Universiteit Leiden
The Netherlands

Opleiding Informatica

Visualizing, Analysing and Constructing L-systems from Arborized 3D Models using a Web Application

Nick van Nielen

Supervisors:
Dr. L. Cao

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

22/08/2023

Acknowledgements

The research conducted and described in this thesis would not have been possible without the invaluable input of my supervisor Dr. Lu Cao. She generously provided the original idea of creating a web-based framework to generate L-systems with A-Frame, the knowledge of 3D model analysis from her paper "L-systems from 3D-imaging of Phenotypes of Arborized Structures", some of the Mammary gland 3D models that were used in her research and the algorithm she used to generate L-systems. Furthermore, I would like to thank my supervisor for her indispensable time, patience and feedback. Due to her input, I avoided unrealistic plans that would make this project too ambitious for the available time. This helped me focus on the tasks that were necessary to reach the final research goals.

Abstract

3D models are of great importance with many different applications in the world of science. In particular, arborized 3D models are an indisputable part in the analysis and visualization of biological concepts. Arborized structures are typically complex, which is why it is common practise to simplify these structures, preferably without loss of important topological features using, for example, Lindenmayer systems (L-systems). Many different applications exist that can realize parts of the process needed to visualize, analyse and create L-system abstractions of arborized 3D models. The focus of this thesis is to combine all these parts into a single web application. To realize all of the aforementioned parts, we design a front-end user interface and a back-end. In the front-end, we use A-Frame and define algorithms to generate and visualize L-systems. In the back-end, we utilize the Vascular Modelling Toolkit's (VMTK) centerline analysis methods to extract data from the arborized 3D models, which can then be applied to L-system generation. The web application provides interactive 3D model visualization with wireframe, mesh, bounding box and vertex components. Using the L-system generation algorithm, a wide variety of structures can be generated from L-system parameters that are based on the analysis data. Furthermore, parameters from a mammary gland and artery 3D model are analysed, extracted and used to generate L-systems. Based on this and related work, we can conclude that the analysis methods are sufficient for arborized 3D models. Next, we find that iteration and branch length standard deviation have a notable effect on the representation of resulting L-systems. Moreover, branch lengths, radii, angles and iteration standard deviation are viable parameters to generate L-systems. Finally, from these key results, we conclude that our method can be used to create a web application to visualize, analyse and create L-system abstractions of arborized 3D models. This in turn provides workflow-improving benefits, easy accessibility and extendibility.

Contents

1 Introduction

In this bachelor thesis we explore the development of a web application that helps visualize and analyze Arborized 3D models and constructs abstractions of these models using L-systems. We will first discuss the context and relevance of our thesis to understand what (Arborized) 3D models are and why we are interested in using (abstractions of) these models in scientific research. Next we talk about our findings regarding current software, its limitations and how it leads to the research question. Subsequently, we determine the goals, requirements and challenges from the research question and aim. Then from this we ascertain our research approach. Finally, we will give an overview of all the remaining chapters and its contents.

1.1 Context and Relevance

3D models are of great importance and have many applications in computer games, art, movies and science. In the various studies of science, they can be utilized to simplify complex scientific ideas. For instance, they have been used to study physiological aspects, like simulating the flow and particle transport of airways [vEHP05], which made it possible to obtain particle deposition patterns and flow characteristics that couldn't be obtained experimentally. They have also been used to analyse the complex anatomy of the canal network in cortical bone [CTSH03] to get a better understanding of the mechanical properties.
It is no coincidence both of these areas are based in the field of biology; 3D models are an unequivocal part in the analysis and visualization of biological concepts. In particular, both articles discuss the usage of 3D models with a tree-like branching structure, or arborized 3D models. These types of structures can be found in plants, but also in vascular structures of animals. Typically, arborized structures are complex due to their high branching order and visibly indistinguishable individual branches, especially in higher branching orders. To realize appropriate visualization and interpretation of these complex structures, it is common to simplify by using only certain sections or by abstraction with minimal loss of features. Generally, computer science methods can be used to extract important features and represent these features in an abstracted model or system. An example of a system that preserves most of the anatomical features of arborized structures is Lindenmayer systems (L-systems) [PL90]. We will use L-systems in our research to describe and build abstractions of arborized 3D models.

1.2 Problem Statement

Many different applications already exist to visualize and analyse (arborized) biological 3D models. A popular platform is 3D Slicer, a platform for medical imaging which provides image registration, interactive visualization, model-based analysis, and more advanced functionality [KPV14]. This can be used together with tools like the Vascular Modelling Toolkit (VMTK) to analyse vascular-like or arborized structures [ISMA18]. When it comes to the abstraction of models, specifically for generating L-systems, L-py can be used for more complex grammar [BCPG10] or other web tools exist that can be utilized for more simple grammar. More general 3D model visualization, processing and editing tools also already exist like Blender [Fou98] and MeshLab [CCC+08].
Web applications like MeshLabJS [oIC16], a web-based version of MeshLab that uses the JavaScript library Three.js [C10] as a rendering engine, can also be created for the purpose of visualization

and analysis of 3D models. An article discussing the efficient visualization of 3D models by web browser [SC13] has shown that web applications can be a well-structured and convenient way of visualizing 3D models. While having clear limitations in handling complex models due to computational and spatial limitations, they can improve accessibility and portability, due to availability on many different devices and browsers. Furthermore, web applications do not have to be downloaded or maintained by the end-user and the 3D models can be stored on and loaded from a database, improving the ease of use and allowing for the possibility to share models between multiple users.
Both system-based and web-based applications can realize parts of the process needed to visualize, analyse and create abstractions of arborized 3D models. However, not many applications exist that combine these parts, which are typically related, namely in anatomical research. Specifically, this type of tool can be realised in a web-based format, which can have all the aforementioned benefits. At the time of writing this thesis, there does not seem to exist such a tool. Therefore, we will concern ourselves in this thesis with the following research question:

How can a web-based tool that is capable of visualizing, analysing and creating L-system abstractions of arborized 3D models be developed?

1.3 Goals, Requirements and Challenges

Now that the main research question has been determined, it is important to establish the goals for our research. With these goals we can set research requirements. As the research question suggests, there are three primary goals in which the application we are interested in developing can be categorized: visualizing arborized 3D models and 3D models in general, analyzing arborized 3D models and generating L-systems. The first primary goal, visualizing (arborized) 3D models, is the most essential part of the web application. Some sort of interactive interface is required to be able to navigate, visualize and interact with the 3D models, but also to assist the user with analysis and L-system generation. The second primary goal, analyzing arborized 3D models, needs functionality to analyze the arborized structure, extract important (statistical) information that can be used for L-system generation and visualize the extracted information. For L-system generation, the final primary goal, it is desired to use a web-based library to interpret and generate L-system grammar, similar to L-py mentioned in section 1.2.
Besides the primary goals, we are interested in the workflow-improving benefits web applications can have mentioned in section 1.2, like accessibility, portability and ease of use. Finally, for the purpose of adding new features during the development or future development of the software, we aim for extendibility.
From the (primary) goals and requirements, we can conclude the recurring challenges that will be faced. Firstly, we acknowledge the obstacle of programming and design. This is mostly present in achieving extendibility and the functionality of the visualization, analysis, navigation, interaction and L-system generation. Secondly, we face the problem of quality and performance, which exists in the optimization of visualization, analysis, interface interaction, accessibility and portability. Noticeably, the challenges of quality and performance as well as programming and design can both be found in most of the goals and requirements that we set for our research. An example of this is ease of use: improving on the functionality in programming and design will ensure a more user-friendly experience and enhancing the performance will reduce delays and improve usability.

1.4 Approach

The goals, requirements and the challenges that accompany them allow for the construction of a general plan before working on the methodology in section 4. For this general plan we establish the programming languages that will be used, the tools that will be used to help against over-complicating the developmental process and the software development methodology. Section 1.3 proposes, for the goal of visualization, an interactive interface is needed. There is a wide variety of web-based rendering engines available that can be used for this purpose. A-frame, a JavaScript web framework for building virtual reality experiences built on top of the rendering engine Three.js [Moz15], was chosen because of its similar performance, cross-compatibility and HTML Entity-Component system. We will discuss more about the capabilities of A-frame and Three.js in section 2.1. As for the programming and design of the analysis, more related work has to be studied, which will be done in section 3. Concerning L-system generation, the JavaScript library lindenmayer.js was selected to interpret and generate L-system grammar in an object-oriented fashion [Bre16]. Additionally, a consideration between static and dynamic website creation has to be made. For the relatively small scale, lesser resource costs and quicker development, static website development has been chosen. As the project grows as well as the demand for dynamic functionality, a dynamic website can be easily developed from the static structure. Finally, having set clear objectives, the waterfall development method was adopted to assure good quality, performance, programming and design [BM12].

1.5 Overview

Concluding the introduction, an overview will be given of all the remaining Section of this thesis. Section 2 and 3 contain the important background information needed to understand the rest of the thesis. In Section 2, the theoretical foundation will be laid by explaining key concepts. Section 3 provides beneficial insights from related work. Next, the design methodology of the research is explained in section 4. Section 5 provides the relevant outcomes of the research. Lastly, section 6 explains and interprets the obtained results and discusses future work.

2 Key Concepts

Before we study related work and describe our methodology of the design of our web application, how it will be utilized to visualise and analyse 3D models and construct abstracted models, we will use this chapter to discuss some important recurring concepts. Section 2.1 explains what the main visualization framework A-Frame and its rendering component Three.js are and how they work. In section 2.2, L-systems are explained and the library Lindenmayer.js we will use to implement them in our web application.

2.1 A-Frame

Before explaining what A-Frame is and how it works, it is important to understand Three.js, which is the component that is at the core of its functionality. Three.js is a web-based JavaScript rendering library used for visualizing 3D graphics with the objective to be cross-browser compatible, easy to use and lightweight [C10]. It uses the web rendering graphics library WebGL, which can use low-level code that can run on a client's GPU for fast physics and image processing. Besides WebGL, Three.js has a broad set of rendering capabilities as SVG, CSS3D and WebGPU are also available renderer addons.

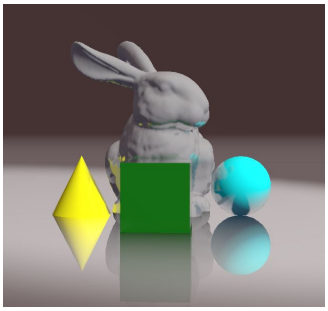


Figure 1: An example of a Three.js scene containing a collection of different 3D objects, including basic geometric shapes and a loaded model. Source: www.threejs.org.

It is possible to create scenes using lights, a camera and geometric shapes. Also more complex (animated) scenes can be created with 3D models. They can be loaded into different meshes with varying material types. Figure 1 shows a scene containing these discussed capabilities. Some material types, like for the floor and sphere in this scene can be set to have reflective surfaces or, conversely, they can be matte like the cube and cone. These features can be implemented using object oriented structures and methods to create games, illustrations and other 3D (VR) experiences.

A-Frame is a web-based JavaScript web framework that focuses mainly on creating 3D VR experiences [Moz15]. It is built upon Three.js, utilizing its rendering capabilities. Therefore, it inherits many of the benefits that Three.js has, like good performance and cross-browser compatibility. Similar to Three.js, A-Frame can be used to create (complex) 3D scenes, like the scene shown figure 1, but with additional VR functionality. Furthermore, it adds a built-in visual inspector that can be used for live scene editing and it runs on declarative HTML with an entity-component system.

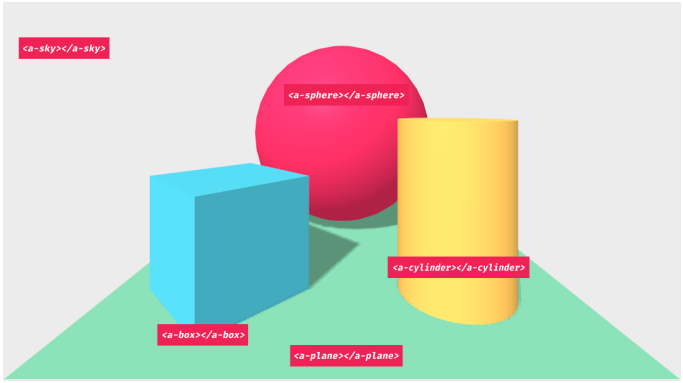


Figure 2: An example of an A-Frame scene containing a collection of different primitive 3D geometric shapes. With each shape the corresponding declaration using HTML elements is shown. Source: www.aframe.io.

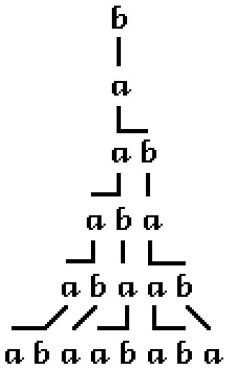
Figure 2 shows some HTML elements for declaring basic shapes in a 3D scene. All of the elements in this scene, or primitives, are added to the main `<a-scene>` element and can have different attributes declared in the HTML, like color or size. This utilizes the main principle of an entity-component system: the primitives are a part of the `<a-entity>` entity element and the components are defined by the HTML attributes which are part of the `<a-scene>` system.

2.2 L-systems

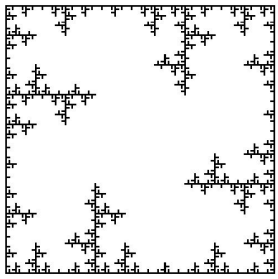
In 1968, the theoretical biologist Aristid Lindenmayer developed a type of formal grammar to describe the growth processes of plants, emphasizing on its topology [PL90]. These so called L-systems are capable of modelling the arborized structure of plants as well as fractals [MM07] and other branching structures, like genetic evolution of organisms [JLR94].
The grammar is similar to the grammar defined Noam Chomsky: they can be generated from an alphabet of symbols that can together form strings [Cho57]. These strings are iteratively rewritten from a primary axiom string using predefined production rules. The important difference compared to Chomsky's sequential approach to applying production rules is the usage of a parallel rewriting method. L-systems apply productions on all symbols at once each iteration. This is a more realistic approach to modelling branching structures, as generally various parts of the structure are developed simultaneously.
There are different types of L-systems, varying in complexity. The simplest class of L-systems is the deterministic, context-free DOL-systems. As illustrated in figure 3a, it introduces the letters **a** and **b**, which is the axiom. Applying the productions

$$b \to a$$
$$a \to ab$$

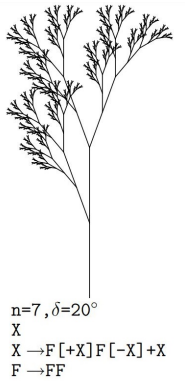
each iteration, meaning that every **b** turns into **a** and every **a** turns into **ab**, the string **abaababa** is generated after $n = 5$ iterations.



(a) An example of the generation of a DOL-system. **b** Symbols are replaced by **a**, which are replaced by the string **ab**.



(b) An example of a Koch curve, where with each iteration some rotational symbols − are added, creating a fractal pattern.



(c) An example of a tree-like structure with tree apex **X**, inter nodes **F**, and rotational symbols − and +.

Figure 3: Three examples of different types of L-systems, increasing in complexity from left to right. Source: [PL90]

Figure 3b shows how geometric symbols + and − can be added to rotate by an angle δ for creating a two-dimensional graphic interpretation of strings. These so called turtle graphics can create fractal patterns, like Koch curves. Finally, L-systems can be further expanded to three dimensions by adding more geometric symbols that can adjust the orientation with a rotational matrix. Branch symbols [and] can also be added, which push and pop the current state of the system on a stack data structure. With these additions, branching structures in two and three dimensions can be generated, like the tree demonstrated in figure 3c.

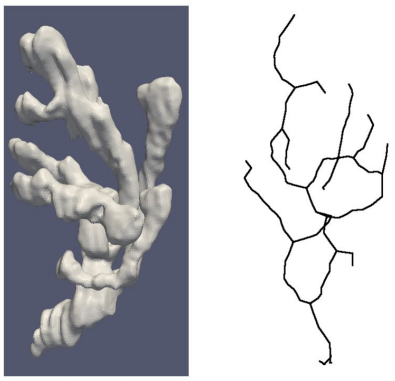
In the research approach section (section 1.4), the usage of Lindenmayer.js was mentioned to interpret and generate L-system grammar. Lindenmayer.js is a JavaScript library that can parse general L-system syntax, including the branch syntax and even context sensitive productions [Bre16]. It does so in an object-oriented fashion: an L-system object is defined and contains an axiom string, productions and finals. Productions are key-value pairs and can be functions, objects or strings. Productions can be applied each iteration by calling the function `iterate(n)`, with n the number of iterations. Finals can be used for interpreting the final L-system string after applying productions. Here, once again key-value pairs are used with functions, objects or strings, which are mainly utilized for drawing the L-system on two-dimensional HTML canvas or building a three-dimensional mesh in Three.js or A-Frame for visualization.

3 Related Work

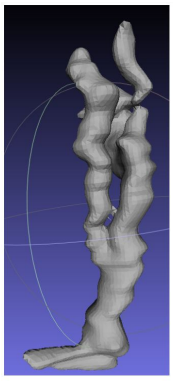
In this section, we discuss the work relating to the developmental process and functionality of our web application. This is divided into two subjects. In section 3.1, the main inspiration for the subject of this thesis is mentioned. This section will also talk about the components that form the foundation of the methodology. Section 3.2 discusses methods used in related articles about the visualization of (arborized) 3D models.

3.1 Arborized Model Analysis

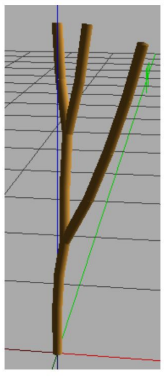
Section 1.1 mentions older, fundamental work that was done on the analysis of airways and the anatomy of cortical bone. An instance of more recent work from the Leiden Institute of Advanced Computer Science (LIACS) on arborized model analysis and work on which the subject of this thesis is based, is "L-systems from 3D-imaging of Phenotypes of Arborized Structures" by Dr. L. Cao and Prof. dr. ir. F.J. Verbeek [VC20]. In this article, the phenotypical changes of the lactiferous duct of newborn mice are analysed under exposure of endocrine disruptors. First, layered gland images were sectioned from mice and then contoured. Next, the 3D point cloud of these contoured models could then be converted to 3D surface models using Poisson reconstruction. The article conveys that 3D models of the lactiferous duct constructed from this process contain a complex tree-like structure [VC20]. In the procedure of detecting morphological changes, a simplification of the arborized 3D model is needed while preserving the important topological features. In this research, the method of center-line extraction was applied to preserve a one-dimensional representation of the original skeletal structure in 3D. Many types of center-line extraction methods were considered, but the polygon-based method [PVS+09] used by the 3D image computing platform Slicer [KPV14] and the 3D geometric analysis tool VMTK [ISMA18] was chosen (figure 4a). Finally, features from the center-line and 3D model were extracted, like number of branches, number of bifurcations and branch length, curvature, torsion, radius, mean and median. From these extracted parameters, L-systems were described with L-py [BCPG10] in a reverse-engineering manner.



(a) A 3D model derived from a mammary gland (left) and its center-line structure (right).



(b) A 3D model instance of the wild-type lactiferous duct control group (WL).



(c) The L-system derived from the WL goup instance given in figure 4b.

Figure 4: Different images from the research conducted in "L-systems from 3D-imaging of Phenotypes of Arborized Structures". Source: [VC20]

In figure 4b, we can see the model from which the L-system in figure 4c is derived. in addition to the extracted branching characteristics, the generated L-system can be used to study the effects of induced phenotypes and simulate mammary gland formation [VC20]. In the section 4, the polygon-based method of center-line extraction and the mentioned parameters will be used in a similar way to analyse more general arborized 3D models and generate similar abstracted L-systems of these models.

3.2 Visualization of arborized 3D models

To understand the important functionality and design choices behind visualizing 3D models and L-systems, we will look at some related work that incorporates visualization and interaction techniques. An article on 3D Visualization of vasculature structures [PO08] discusses the importance of accuracy, reconstruction time and geometric complexity. It states that the most important model visualization techniques are model-based or model-free approaches. The former relies on model assumptions, which forces the visualization to comply with these assumptions. The latter relies on representing the model data as accurately as possible at the cost of interpretability. Research done specifically on web-based visualization of 3D models [SC13] suggests that there are two essential model rendering techniques. The first technique being server side rendering, which computes the model and scene on the server and sends the result to the client as an image. The second technique is client side rendering, which loads the scene and model on the client, utilizing less resources on the server side. This research also suggests that both the progressive mesh streaming with client side rendering and image compression with server side rendering are highly usable visualization methods, depending on 3D model complexity [SC13].
For a given visualization system with intractability, some key interaction approaches have to be considered. A survey of 3D interaction techniques [Han97] considers interaction techniques developed for object application control, navigation and manipulation in a 3D environment. Object application control is required for parameter adjustment and changing modes, which are a form of interaction between the user and the system outside of the 3D environment. Preferably, 2D interface components are used to improve usability and reduce interaction errors. For object navigation, the 3D viewpoint or camera can be manipulated by the zoom, pan and pivot operations. There are many different camera movement techniques to fly, walk or step around the 3D object. Both zooming and panning are manipulations that can be fulfilled solely by the camera, but pivoting uses a center of rotation, which can be the object itself or some other point in 3D space. It is therefore recommended to implement viewpoint manipulation that can naturally guide the user [Han97]. Lastly, object manipulation techniques such as selection, editing and translating exist. They can be manipulated using 2D mouse input, application control or other 3D input methods. Some object manipulations, like rotation and translation, can be performed in the real world and thus require natural visual feedback to the user in order to provide awareness of the system state and interface capabilities, also known as kin-aesthetic feedback [Han97].

4 Methodology

Here we describe our methodology from the theory and literature we acquired in the previous sections. To visualize and analyse 3D models as well as generate L-systems, we need to design the desired functionality. We will apply design patterns, discuss the design choices and sketch the functionality of our web application using A-Frame in section 4.1. Furthermore, in section 4.2 we discuss how the model analysis is implemented with VMTK, python and the user interface of the web application. Lastly, we reason how we combine Three.js and Lindenmayer.js to generate and visualize L-systems in section 4.3.

4.1 Design

For the design of web-based applications, a modelling method is needed to cover all the requirements established in section 1.3. This can help visualize and document the components and corresponding functionality. UML diagrams have been shown to be a powerful modeling method for covering all required components of web applications [KK02]. Many UML types exist, therefore, an important consideration is the type that best suits the requirements and approach mentioned in section 1.4. In this section, web frameworks that will be used and the static nature of the website structure were discussed. To model the static web components, UML deployment diagrams will be used with static views. The important elements in these diagrams are displayed with a node or rectangle, relations between nodes with regular arrows and dependencies between nodes with dotted arrows. A line indicates a special type of relationship. In this relationship, there is a parent node that contains all the contents of the child node. Nodes that share functionality, forming a component, are surrounded by a colored rectangle, which denotes cohesion. Every node, arrow and component contains a text that describes its class or functionality. This section highlights two distinctive parts that the web application consists of: the client, which views and controls the front-end, and the server, also known as the back-end.

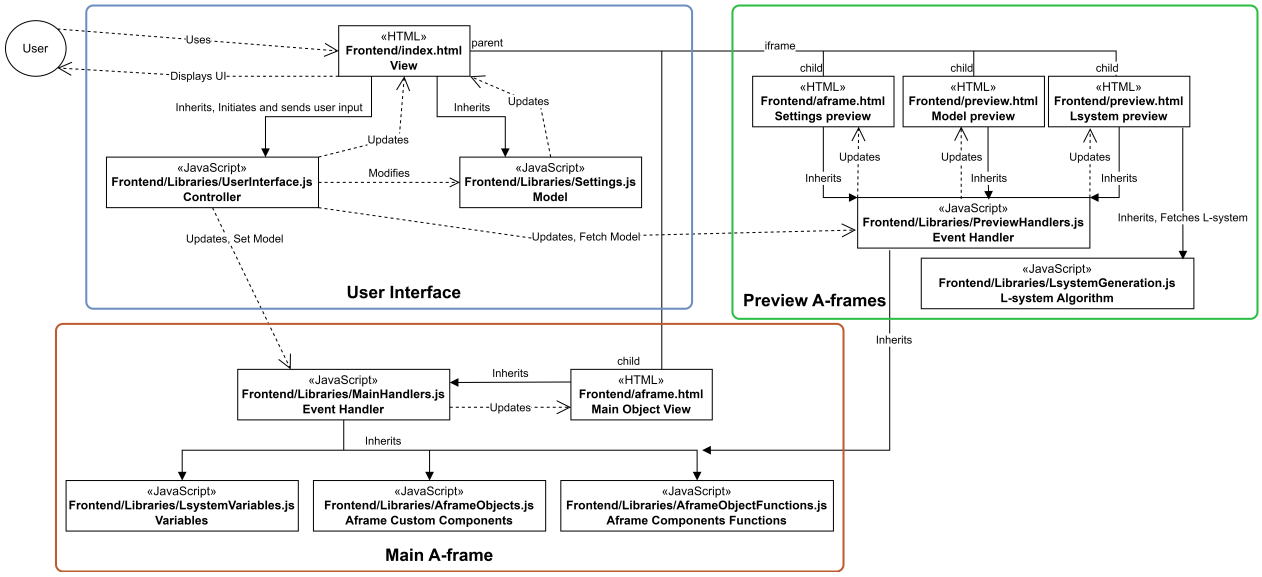


Figure 5: UML Diagram of the front-end layout of the web application.

4.1.1 Front-end

For 3D model visualization and manipulation, an interface is needed for user input, display and storing between static web page states. All this functionality is available on a single page called `index.html`. All the front-end components and the user interface component that contains this page can be found in figure 5. Furthermore, `index.html` is the view node of the Model-View-Controller (MVC) programming paradigm, which is used to display the user interface and is part of the interface component. The MVC metaphor has historically been used, for example in the Smalltalk-80 system, to develop interactive applications [KP+88]. Besides the view node, the states between static page serves are stored in the cookie object data structure altered by the model node in `Settings.js` and updated in the view node. Modifications to this data structure are made by the controller node `UserInterface.js`, which reacts to user input in the view. Some user input does not change the model and is directly updated in the view by the controller.
Inside the view node, options can be selected and settings can be changed. For the 3D model rendering and visualization, two components are used that contain A-Frame with Three.js rendering. These components are loaded within iframes on the main page, instead of redirecting the user to separate pages. The iframes can be compared to the builder design pattern [GHJV93]: they show the same models, but the way they are represented is determined by which functions are used by the JavaScript object files they share. The first component is the main A-Frame. In this component, the main object or L-system view is stored and displayed using the A-Frame web framework inside `aframe.html`. It is controlled by `MainHandlers.js`, the event handlers that wait for events from the controller node in the user interface. Besides handling events, it can receive a 3D model from the controller which can then be loaded into the main object view. For loading L-systems and objects into the view, `MainHandlers.js` uses L-system definitions `LsystemVariables.js`, A-Frame object definitions `AframeObjects.js` and A-Frame component functions `AframeObjectFunctions.js`. The second component is the preview A-Frames. As the name suggests, this is where all the preview nodes of the L-system, A-frame and settings model are shown. This component has been designed to easily expand and remove the number of A-Frame nodes that can be used at once. The settings preview frame is used to display settings changes on the main A-Frame model/L-system and re-uses `aframe.html` to do so. The model and L-system previews use `preview.html` to display what the loaded 3D model or generated L-system will look like. Just like the main A-Frame, the preview iframes use event handler `PreviewHandlers.js`, L-system definitions `LsystemVariables.js`, A-Frame object definitions `AframeObjects.js` and A-Frame component functions `AframeObjectFunctions.js`. Besides handling events from the controller, the event handler can send the preview models from the L-system and model preview frames to the controller, which can be loaded into the main A-Frame. Before the L-system can be previewed or loaded into the main A-Frame, it first has to be generated using the L-system algorithm `LsystemGeneration.js`, explained further in section 4.3. The L-system generation as well as 3D model loading from I/O are initiated by the controller, using the preview event handler. Users might want to interact with multiple user interface components at once. Furthermore, the view and model cannot be stalled while other tasks by the controller are performed. For this reason, asynchronous functions are utilized in most nodes to reduce the number of blocking operations on the web application.

4.1.2 Back-end

Some tasks in the user interface require computations on the server or I/O operations. The base functionality of the user interface can be expanded with these kinds of tasks by adding a back-end. In figure 6, the UML diagram of the user interface and back-end components as well as how they interact are shown. The back-end uses python to run the development server `devServer.py` using flask, a micro web application framework for simple scalability [Gri18].
For the deployment server `deployServer.py`, Gunicorn [GP17] and Nginx [Ree08] are used with flask as an HTTP interface and web server with multi-threading . Flask uses the `Router.py` and `Requests.py` blueprints, which are constantly running application components.
Available HTML pages are served by the router to the user when requested, like the index or iframe pages. Requests from the user interface controller are operated by the request handler. This works similar to the chain of responsibility behavioral design pattern [GHJV93]: each HTTP request that can be handled is arranged in a chain of objects. If the handler cannot process the request, it is passed along the chain. For blocking I/O tasks, a request is sent to the server, which returns a response to the handler. The handler then asynchronously sends a HTTP response to the controller while it waits for input. For non-blocking tasks, like 3D model analysis with VMTK in section 4.2, a new thread is created and more requests can be made by the same controller.

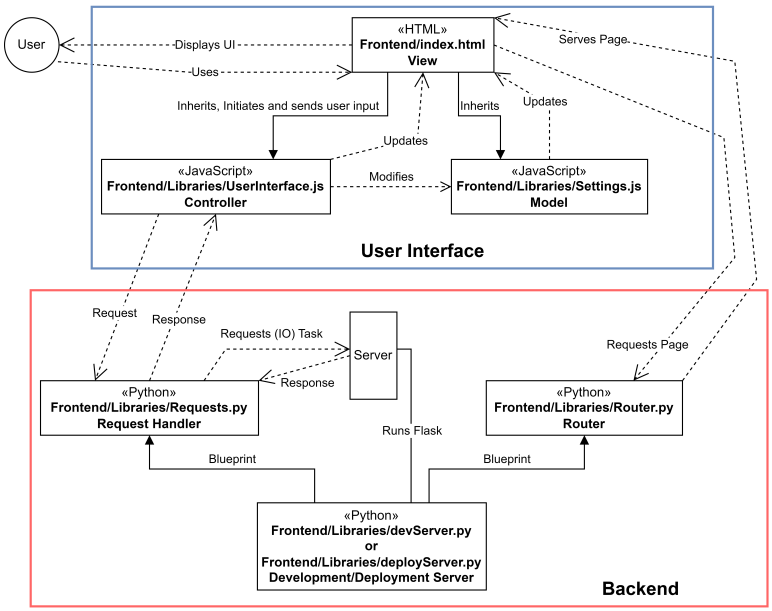


Figure 6: UML Diagram of the back-end communicating with the user interface of the front-end of the web application.

4.2 Model analysis

Section 3.1 discussed related work on arborized model analysis using VMTK tools and important 3D model features that can be extracted. Using the well-defined back-end structure from figure 6, these VMTK tools that run on python are deployed in the request handler node. During the analysis, endpoints on the model are selected in the front-end view node. The endpoints and 3D model name are then send with an analysis request to the request handler by the controller node. In the request handler, three steps take place: verification, instance creation and the analysis itself. For verification, the request parameters are verified by checking for correct 3D model filename and type. If either the filename or file type are incorrect, the model cannot be found in the objects folder and no analysis will be possible, so an error will occur.
For the second step, an analysis instance is created on a new thread to prevent request blocking and fatal errors to occur on the development or deployment thread. Fatal errors can arise due to nondeterministic VMTK methods and therefore the inability to catch all errors during the analysis process. Because most VMTK methods are nondeterministic, it is hard to predict whether the analysis instance is still operating or in an infinite loop.
The analysis process itself and its structure can be explained by the diagram in figure 7. Endpoints and the 3D model file path, which consists of the objects folder, filename and extension are transferred to `vmtkCenterlines()`, which is used to compute the model centerline. This is then transmitted to `vmtkCenterlinestoNumpy()`, to calculate the centerline mesh points. The centerline is also used in `vmtkCenterlineGeometry()` to calculate the length, turtuosity, torsion and curvature of each centerline. With `vmtkBranchExtractor()`, the centerline is split into branches and grouped. Bifurcations are grouped in the blanking scalar field extracted from the centerline. For the final data extractions, the split centerline is used to calculate branch length, turtuosity, torsion and curvature with `vmtkBranchGeometry()` and bifurcation angles with `vmtkBifurcationReferenceSystems()`. Extracted data from the analysis is compressed into a string buffer, which is returned from the thread when the analysis is complete. If any error occurs during the analysis or the data extractions take too long, the request handler returns an error. Otherwise, the string buffer output is channeled through the request handler to the controller.

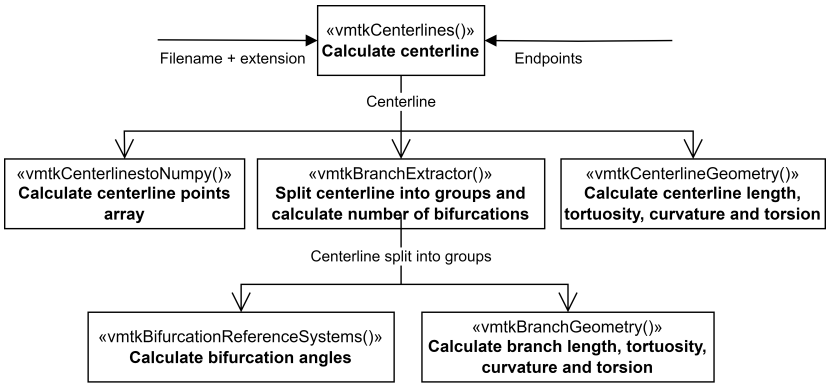


Figure 7: Diagram of the analysis process of an arborized 3D model. In this process, VMTK functions are used to extract, analyse, group and split centerlines. The corresponding branches from centerline splitting can then be analysed further with additional VMTK functions.

4.3 L-systems

In section 4.1, the L-system algorithm node was introduced in the preview A-Frame of the front-end design. This section discusses the details of the algorithm, how Lindenmayer.js and Three.js are used to generate L-system models and how parameters extracted in the analysis are interpreted by the algorithm. First, the contents of the L-system algorithm node are discussed in section 4.3.1 and 4.3.2. The former is about the L-system string generation and the latter about L-system grammar interpretation and generating 3D meshes. Second, from the analysis method in section 4.2, a method for importing parameters into the L-system algorithm is discussed in section 4.3.3.

4.3.1 L-system string generation algorithm

To understand how algorithm 1 works, a formal definition of the L-system and a set of L-system variables are needed. The L-system can be defined by the triplet G = (V, ω, P). Here:

- V Is the alphabet consisting of the variables I, B and constant D. I Is a branch segment, B is a branch segment candidate. D Is a dead branch, which is a branch segment candidate that will not expand any further.
- ω Is the axiom. For a standard system the axiom is III[/&B][/&B]. The number of I variables can be chosen.
- P Is the set of production rules, these are defined within the algorithm. For each variable, a new set of symbols and variables are returned.

Variables I and B share the following parameters:

- current iteration $\in \mathbb{N}_{\geq 0}$. This keeps track of the current iteration in the algorithm
- centerline index $\in \mathbb{Z}$. This determines for the variable which centerline, with a unique centerline id, it is a part of.

Parameters unique to I are:

- radius $\in \mathbb{R}_{\geq 0}$ Of the cylinder geometry.
- length $\in \mathbb{R}_{\geq 0}$ Of the cylinder geometry.

Parameters unique to B are:

- lifetime $\in \mathbb{Z}_{\geq 0}$ Of the variable
- maximum centerline length $\in \mathbb{N}_{\geq 0}$. This determines the maximum lifetime of both I and B. If the centerline length (in iterations) exceeds this number, B will turn into a dead branch D. From literature [Par16] was inferred that this follows a right-skewed distribution.

Furthermore, L-system parameters, italicized in algorithm 1, are defined as follows:

- stemLength $\in \mathbb{N}_{\geq 0}$ Defines the number of branch segments in the stem of the L-system. This determines the number of I variables in the axiom.
- radialIncrement $\in \mathbb{R}$ Determines how much the thickness of each I grows each iteration.
- upwardsAngle, upwardsAngleSpread $\in [-180, 180]$ Are the mean respectively standard deviation of the normal distribution of upwards growing branch segments I.
- lineLength, lineLengthStd $\in \mathbb{R}_{\geq 0}$ Are part of the cylinder geometry length distribution of I. Similar to the maximum centerline length, a right-skewed distribution is used for this L-system parameter.
- nb_axes_prob Is the probability a bifurcation will occur each time a new segment is built.
- iterations, iterationsStd $\in \mathbb{N}_{\geq 0}$ Sets the maximum centerline length of B to a value in this right-skewed distribution.
- bifurcationAngle, bifurcationAngleSpread $\in [-180, 180]$ Are the mean and standard deviation of the bifurcation angle normal distribution. This angle is set for each part B of the bifurcation.
- maxOrder $\in \mathbb{N}_{\geq 0}$ Determines how spread out the branches will be from the stem at the starting stages of the algorithm. A higher maxOrder will ensure a longer lifetime for B variables at lower branch orders.

Concluding the definitions, helper functions are utilized during variable and parameter creation. The helper functions include:

- max duration, Which uses the L-system parameter maxOrder to calculate the lifetime of B variables using the current branch order.
- random integer, Which calculates a random integer between a given integer minimum and maximum.
- (skewed)distribution, Which return a random number around a mean with a given standard deviation and possibly a shape parameter, describing the skewness of the distribution.

From these definitions can be concluded that algorithm 1 builds the scaffolding of the L-system with B variables and replaces B variables with I variables. I Variables are constant in the sense that they don't change to other symbols, only the segment radius changes each iteration. The lifetime of B ensures that not many bifurcations occur at the start of the generation, which reduces intersections between segments in later generation stages. Besides it's lifetime, B variables have a certain chance to create a bifurcation or turn into a dead branch each iteration. As the variable and parameter definitions suggest, among other things, the rates at which bifurcations occur, lifetimes, number of iterations and radius increments can be preset and adjusted by the user. In the next section, the purpose of the remaining parameters will be explained.

Algorithm 1 Pseudo code for L-system algorithm

```
Initialize stem with stemLength branch segments and the first bifurcation
for number of iterations do
    for each I do
        Increment radius by radialIncrement
        return I
    end for
    for each B do
        if max centerline length < current iteration then
            return D
        end if
        if time is shorter than maximum duration of B then
            Increment time by 1 and iteration by 1
            ^ ← angle upwardsAngle with standard deviation upwardsAngleSpread
            I ← length lineLength with standard deviation lineLengthStd, radius radius,
                store iteration and centerline index
            return ^IB
        else
            Set time to 0, increment iteration and order by 1
            if random chance of branching > nb_axes_prob or bifurcation has dead branch then
                / ← angle with random integer between -60 and 60
                return [/B]
            else
                branches ← new array
                for each branch in bifurcation do
                    set max centerline length to iterations with standard deviation iterationsStd
                    set centerline index to random integer
                    / ← angle with random integer between -60 and 60
                    & ← angle bifurcationAngle with standard deviation bifurcationAngleSpread
                        and iteration correction
                    push [/&B] to branches
                end for
                return branches
            end if
        end if
    end for
end for
```

4.3.2 L-system grammar interpretation algorithm

Now the L-system object string has been generated in section 4.3.1, an interpreter is needed to create a 3D model corresponding to this object string. Geometric symbols used in algorithm 1, like ^, & and / with the given angles or [and] are used as defined in the standard L-system grammar in section 2.2. However, each variable symbol I requires a process to determine branch segment position, length and intersection with other segments.
Algorithm 2 shows how the radius and length of variable I, defined in algorithm 1, are used to build the L-system model with cylinder geometries. currentSegment Determines the position and orientation of each variable I. From this, a copy is made called newSegment that is used to build the cylinder geometry with a radius, length and number of radial segments with L-system parameter radialSegments. Each segment is built from the center of the cylinder, thus currentSegment has to be shifted by half a segment length before copying to newSegment.
Before the geometries are pushed, the L-system model is checked for intersecting segments with newSegment if pruneIntersections is true. If pruneIntersections is false and boundingEllipsoid is true, intersection of newSegment with a bounding ellipsoid, which is preset with L-system parameters, is checked. If none of these conditions hold, newSegment is pushed to the L-system model and centerlinelengthsObject is used to keep track of the number of centerlines and the total centerline lengths. To count the total number of branches, branchCounter is incremented by 1 for each variable I.

Algorithm 2 Pseudo code for building L-system model

```
branchCounter ← 0
centerlinelengthsObject ← [ ]
currentSegment ← new Three.js mesh
for each I do
    translate currentSegment by half of length
    newSegment ← copy of currentSegment
    build cylinder geometry for newSegment with radius, length and radialSegments
    translate currentSegment by half of length
    if pruneIntersections and segment intersects with L-system model created
            so far then
        do not push segment to L-system model
    else
        if boundingEllipsoid and segment intersects with bounding Ellipsoid then
            do not push segment to L-system model
        else
            increment branchCounter by 1
            add length to the integer at centerline index in centerlinelengthsObject
            push segment to L-system model
        end if
    end if
end for
```

4.3.3 Parameter extraction from model analysis

In table 1, an overview is given of all the L-system parameters used in this section. Except the number of bifurcations, the centerline points, branch and centerline Curvature, Tortuosity and Torsion, all parameters extracted in section 4.2 are used in the L-system algorithm. First, the branch probability is set to 100%, because there are never two single segments between each bifurcation. Furthermore, max order has not been included in the analysis, so it is set to 0. Next, the mean number of iterations is set, which is multiplied by two because it takes the algorithm at least two iterations to build a segment. The mean iterations cannot exceed 50, as the L-system model would grow too large and it cannot be less than one, as no segments would be built. Then, the upwards angle is imported, which is not allowed to be larger than 30 degrees or smaller than -30 degrees for practical purposes. After that, the bifurcation angle mean is divided by two for each segment in the bifurcation. Finally, the remaining parameters seen in table 1 are imported from the analysis data without any further adjustments. After importing the parameters and saving them on the client side, the L-system generation can commence.

Parameter name (Units)
Segment Radius (Pixels)
Radial Increment (Pixels)
Bifurcation Probability
Radial Segments
Iterations
Iterations std.
Max Order
Segment branch Length (Pixels)
Segment branch Length std. (Pixels)
Upwards Angle (Degrees)
Upwards Angle std. (Degrees)
Bifurcation Angle (Degrees)
Bifurcation Angle std. (Degrees)

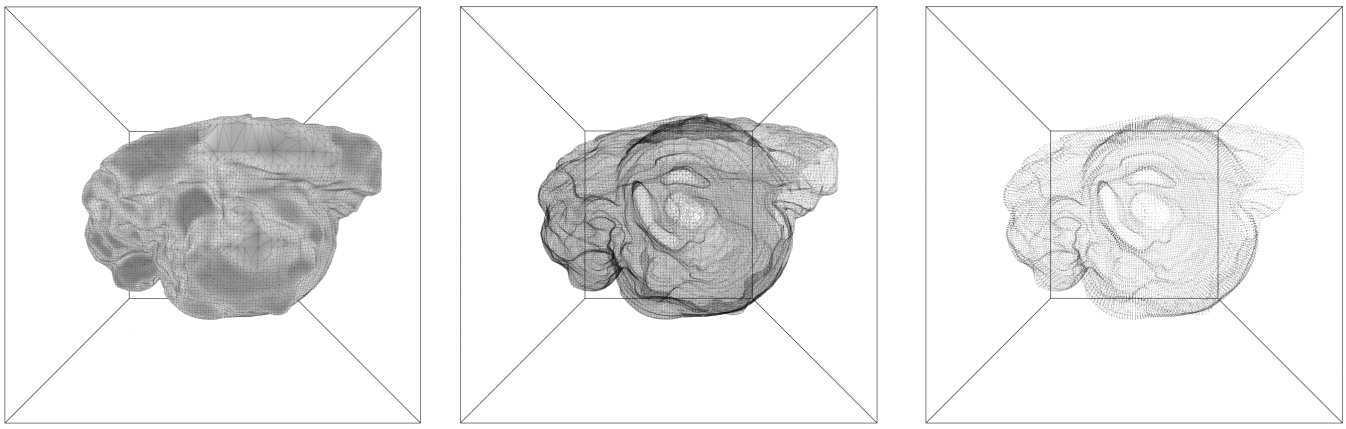
Table 1: A list of all available L-system parameters used in algorithm 1 and 2.

5 Results

This sections displays the results from the web application we created in our methodology. Section 5.1 and 5.2 display the primary goals of visualization and L-system generation of the web application. The first section shows general model visualization capabilities in the user interface and the second section discusses how the L-system Generation algorithm combined with selected parameters manifests itself visually. Finally, section 5.3 conducts experiments on three different arborized 3D models. This section mentions how centerlines and parameters are extracted and what kind of L-systems are generated from these parameters.

5.1 Model Visualization

Before looking at the other results, it is important to get an overview of the main visualization components of the user interface, discussed in section 4.1. These components were tested on multiple browsers and devices. The main components are a bounding box, the model mesh, wireframe and vertices. Figure 8a shows all the components in one image, figure 8b only shows the wireframe and figure 8c only the vertices. The dark-coloured wireframe and vertices are positioned on the mesh surface to improve visibility. Furthermore, the orbit view can be used to pan around and zoom in on the 3D object. The bounding box helps guide the users orientation around objects where the sides are hard to distinguish from one another.



(a) View with a mesh, wireframe and vertices turned on.

(b) View with only the wireframe.

(c) View with only the vertices of the model.

Figure 8: Object views of a Xenopus Embryo 3D model in the user interface. In each view, the model is bounded by a bounding box. Model source: fishnet

An important drawback from client-side visualization is the limited resources. For 32-bit and 64-bit users, 1 gigabyte respectively 64 gigabyte of RAM is available. This limits the complexity of the 3D models that can be rendered. Furthermore, the user must posses sufficient graphical processing power to use the orbit view around the object. Although, most semi-modern processors contain adequate build-in graphical processing units.

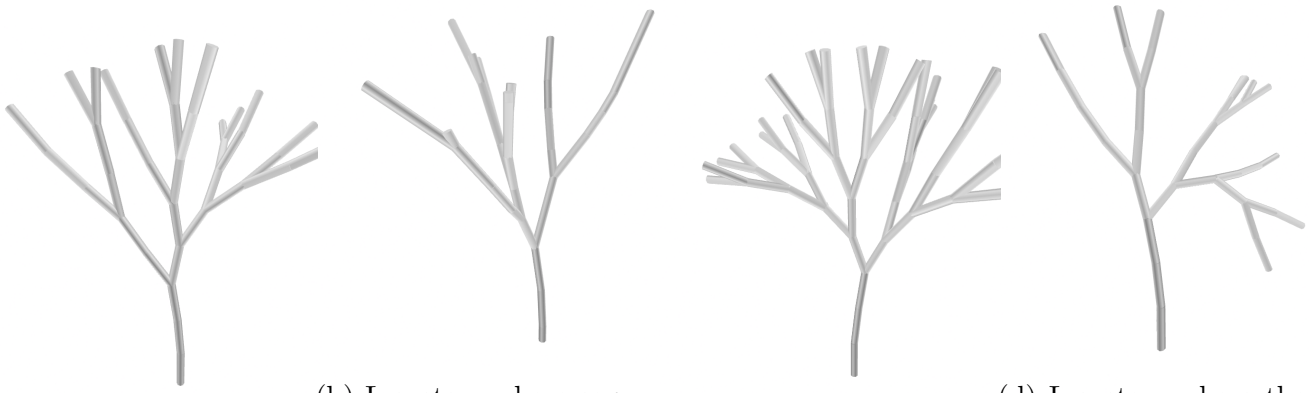
5.2 General L-system Generation

With the algorithm 1 from section the methodology section and the discussed user interface where parameters can be inserted, many different types of L-systems can be generated. Table 2 shows the standard L-system parameters that are supplied by the web application. Figure 9 demonstrates, for larger L-systems with more iterations, how some of the individual parameter adjustments can affect the results of the generated L-systems.

Parameter name (Units)	Value
Segment Radius (Pixels)	0.03
Radial Increment (Pixels)	0.0001
Bifurcation Probability	40%
Radial Segments	8
Iterations	4
Iterations std.	0
Max Order	0
Segment branch Length (Pixels)	0.3
Segment branch Length std. (Pixels)	0
Upwards Angle (Degrees)	5
Upwards Angle std. (Degrees)	2
Bifurcation Angle (Degrees)	17.5
Bifurcation Angle std. (Degrees)	0

Table 2: Standard L-system parameters supplied by the web application and L-system algorithm. We exclude L-system stem data, since it does not influence the generation process of the L-system itself.

It can be of interest to the user to create a wide variety of L-systems or generate L-systems closely resembling a 3D model. Depending on the choice of parameters, many varieties can be formed. The L-system in figure 9b shows how the branch lengths are normally distributed when adding segment branch length standard deviation. Compared to the base model in figure 9a, it has both shorter and longer branches and on average approximately equal branch lengths. Using the standard parameters but with iteration standard deviation within close range of the number of iterations, results in similar L-systems to figure 9a, except with equal segment lengths. Figure 9c illustrates how increasing the branch probability causes the L-system algorithm to increase the number of bifurcations. This makes the L-system appear fuller in contrast with the base model. In the last L-system shown in figure 9d, adding the Bifurcation angle standard deviation generates bifurcations with smaller angels and larger angles, which can tilt the branches more downwards.



(a) Base L-system containing parameters from table 2.

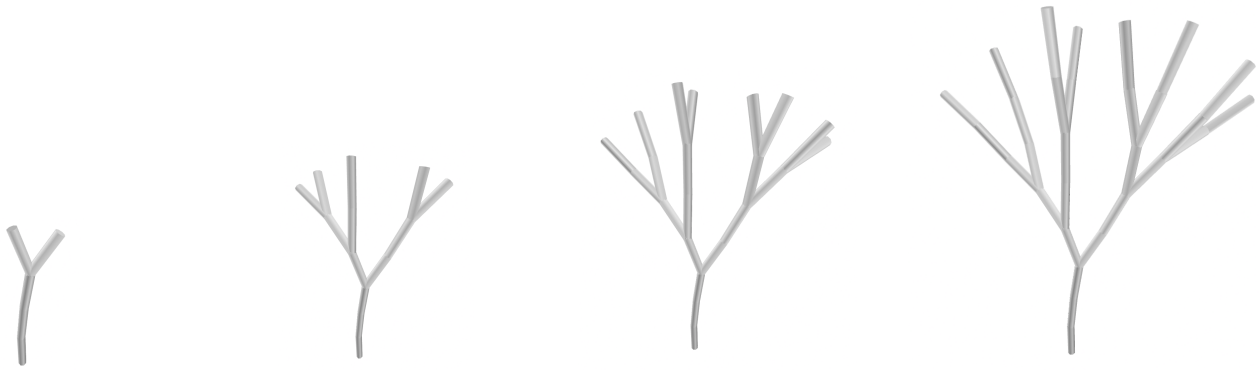
(b) L-system where segment branch length standard deviation is set to 0.15 pixels.

(c) L-system where the branch probability percentage is set to 70.

(d) L-system where the bifurcation angle standard deviation is set to 20 degrees.

Figure 9: A variety of randomly generated L-system with varying parameters. The models are generated using the standard parameters from table 2 with nine iterations instead of four and an additional parameter adjustment.

The "show iterations" functionality in the user interface can, for each iteration of the algorithm, display how the L-systems develop. The development from the first iteration to the ninth iteration can be seen in figure 10. The system starts with a stem and the first bifurcation and develops further from that point on. Furthermore, from both figure 9 and 10, the radial fashion in which the tree-like structures develop can be observed. However, it can be noted that the depth of these radially developing centerlines from the stem is not exactly nine iterations: algorithm 1 indicates how the lifetime of branches and the maximum branching order determine the initial growth of the system.



(a) First iteration

(b) Fifth iteration

(c) Seventh iteration

(d) Ninth iteration

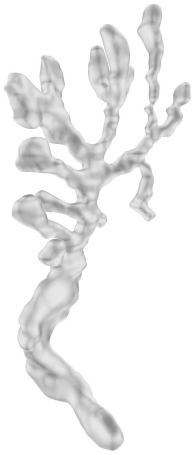
Figure 10: The developmental process from left to right of a randomly generated L-system using the standard parameters from table 2 and nine iterations (starting from one, not zero).

5.3 L-system Generation from Arborized 3D Models

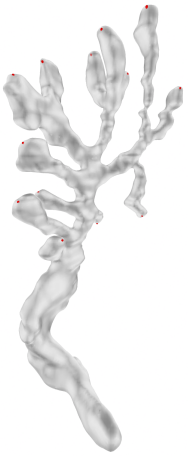
During the design process, the mammary gland model in figure 11a was used to test the visualization of this model, the analysis methods, parameter extraction and the corresponding L-system generation. The results of these tests can be seen in section 5.3.1. To verify whether arborized 3D model analysis and parameter extraction works on more complex cases, tests were additionally conducted on an artery model in section 5.3.2. The results are presented similar to the analysis process one would go through while using the web application. First the endpoint selections will be shown on the model, comparable to the visualization of endpoints in the user interface. Besides endpoints, the calculated centerline from the analysed 3D model will be shown. This centerline model can then be layered on top of the original model to check its accuracy and coverage. Next, the extracted data from the 3D model and centerline will be shown in a table, as it would be shown in the user interface after analysis. Lastly, L-system data will be extracted from the table, which can then be used to randomly generate L-systems. Three generated L-system examples with the standard extracted data and three examples with varying branch length standard deviation, iteration standard deviation and branch probability will be shown to demonstrate numerous model abstractions. The results from this process can then be interpreted by comparing the data and generated L-systems to the original 3D model and centerline model. The overall interpretation of the models has been done from a computer science perspective. Most of the morphological interpretations are derived from parameter changes to the L-system generation algorithm and biological interpretations are omitted. Furthermore, it is important to note that all the 3D models, centerlines and L-systems are shown from a pre-selected perspective. Since the models orient themselves beyond a two-dimensional plane in three-dimensional space, it can be difficult to determine centerline and branch counts. Thus, the number of centerlines and branches shall be stated where applicable.

5.3.1 Mammary Gland

For the first experiment, a simplified mammary gland 3D model was used. The mesh of the model as it would be shown in the user interface can be seen in figure 11a. The tree-like structure, starting at the thicker bottom stem and splits to branches that vary in thickness, contains few bifurcations and consequentially centerlines. For this reason, it was used to test and verify the analysis and the L-system parameter extraction methods during the development of the web application. In this section, the results of the final analysis and L-system generation from the extracted parameters are found.



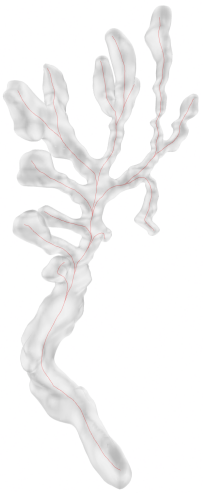
(a) The model mesh as it is visualized in the web application.



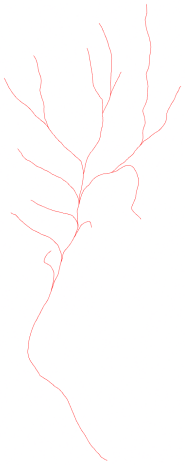
(b) 13 Endpoint selections (in red) layered on top of the model mesh.

Figure 11: Model mesh of a breast gland from newborn mice, used in the research conducted by Dr. L. Cao and Prof. dr. ir. F.J. Verbeek [VC20]. In the left image the model itself can be seen and in the right image the endpoint selections used for centerline creation this experiment are shown.

For the preparation of the analysis, endpoints are selected. In figure 11b, the selected endpoints are shown in red. The stem at the bottom was picked as the source point of the centerline structure. Endpoints are chosen based on distinct branches and where they approximately end in the mesh structure. Using this method, 13 endpoints were selected.



(a) The computed centerline model illustrated inside of the original mesh.



(b) The calculated one voxel thick centerline model without the original mesh.

Figure 12: The resulting centerline (in red) from the analysis on the mesh in figure 11a shown inside of the mesh (left) and by itself (right).

After selecting the endpoints, the analysis itself can start. From the analysis the centerline in figure 12 was extracted. The centerline model inside of the mesh in figure 12a shows no disconnected centerlines. Furthermore, the centerlines seem to start at the source point and roughly end at the selected endpoints when comparing the model to figure 11b. Additionally, the one voxel thick centerline model in figure 12b appears two-dimensional. This is due to the near two-dimensional shape of the 3D model, as can be seen in figure 11a, with a few exceptions of branches slightly overlapping.

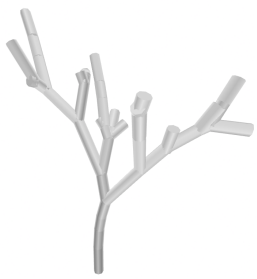
Attribute name (Units)	Mean	Std.
Branch Length (Pixels)	63.96	39.20
Point Radius (Pixels)	14.46	4.13
Centerline Length (Pixels)	582.56	123.60
Stem Length (# Iterations)	4.32	
Bifurcations	11	
Centerlines	13	
Branches	24	
Iterations	4.79	1.93
Bifurcation Angle (Degrees)	62.81	19.45

Table 3: Attributes extracted from the analysis of the model mesh in figure 11a. Only the important attributes used for L-system generation are shown in this table. Thus, attributes like curvature, torsion and tortuosity are omitted. Branch length and centerline length both exclude the stem length. In the first column, the attribute name an units are shown. The second and third column show the mean and the standard deviation if applicable.

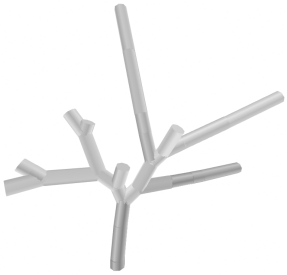
The data collected from the analysis both from the model itself and the centerline model can be seen in table 3. A few observations can be made from this data. Firstly, from the 13 endpoints selected in figure 11b, 13 centerlines have been created, meaning there are no incorrectly analysed centerlines. Secondly, the number of bifurcation does not entirely match the number of bifurcations that can be found in figure 12b, because there is also one trifurcation mistaken for a bifurcation. Another observation is that the total number of iterations, which is in terms of how many mean branch lengths it takes to build one mean centerline, is approximately 9 ± 2. Four of the total number of iterations consist of only the stem. This can also be observed from the model in figure 11a. Finally, most of the varying data has a relatively high standard deviation, but the branch length, excluding the stem length, has the highest standard deviation. Figure 12b also displays this large variety in branch lengths between bifurcations or bifurcations and endpoints.

The parameters from table 3 were then used to randomly generate L-systems. First, three L-systems were selected from many similarly generated systems using the given parameters, which can be found in figure 13. During this process, some L-systems were created with colliding branches due to randomness of the algorithm. These results were left out by using the prune intersections option, as the original model does not contain intersections. Furthermore, models with similar data extracted from the original mesh were selected for the best representation. The three models were all generated with ten iterations, because building each segment needs at least two iterations.

Additionally, they vary in branch lengths, number of centerlines and number of branches. The branch lengths range from frequent short branches in figure 13a to frequent long branches in figure 13b or a mixture in figure 13c.



(a) L-system with 16 centerlines and 43 branches

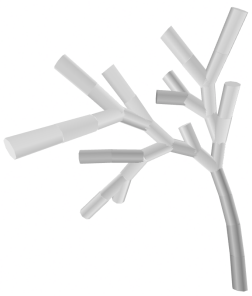


(b) L-system with 9 centerlines and 30 branches

(c) L-system with 13 centerlines and 37 branches

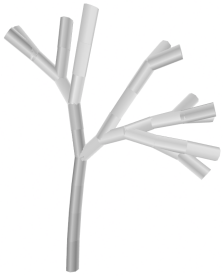
Figure 13: A wide variety of randomly generated L-systems based on the data from table 3 and setting the number of iterations to 10. The upwards curve of the stem was set to the standard value of 5 with a standard deviation of 2.

Lastly, figure 14 shows three L-systems with 13 centerlines and varying iterations, iteration standard deviation and no branch length standard deviation. These models were again selected from a pool of randomly generated L-systems. Figure 14a shows a model where the the branch length standard deviation has been removed, which removes extra long branches since all the branch segment lengths are equal. The iteration standard deviation has been reduced by one iteration in figure 14b. Consequently, the variety in centerline lengths reduces. In figure 14c, the variety in centerline lengths has been completely removed and the branch probability has been reduced to retain a diverse amount of centerlines and bifurcations. Due to the constant branch lengths, the overall shape of the L-systems in figure 14 is more compact compared to figure 13.



(a) L-system with 38 branches, 10 iterations and no line std.

(b) L-system with 33 branches, 8 iterations, iteration std. 1 and no branch length std.



(c) L-system 35 branches, 10 iterations, no iteration std., no branch length std. and 66% branch probability.

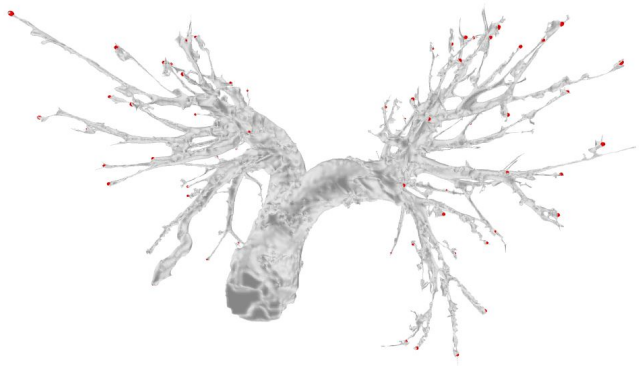
Figure 14: A wide range of randomly generated L-systems, all with 13 centerlines, based on the data from table 3. The systems differ in number of iterations, iteration standard deviation, branch probability and have no branch length standard deviation. The upwards curve of the stem was set to the standard value of 5 with a standard deviation of 2.

5.3.2 Artery

For the second experiment, an Artery 3D model was used. The mesh of the model and how it would be visible in the user interface can be seen in figure 15. This model has a more complex tree-like structure starting at the much thicker bottom stem and quickly decreasing in thickness. Some branches are hardly connected to each other with thin segments. Furthermore, the structure has a high branching order and therefore contains many bifurcations and centerlines. Thus, it can be used to test the analysis and the L-system parameter extraction methods on a more complex structure. In this section, the results of the first unsuccessful analysis attempt as well as the results from the second successful analysis and L-system generation from the extracted parameters are demonstrated.

Figure 15: The model mesh of a complex arborizing artery as it would be visualized in the web application. Model source: www.sketchfab.com

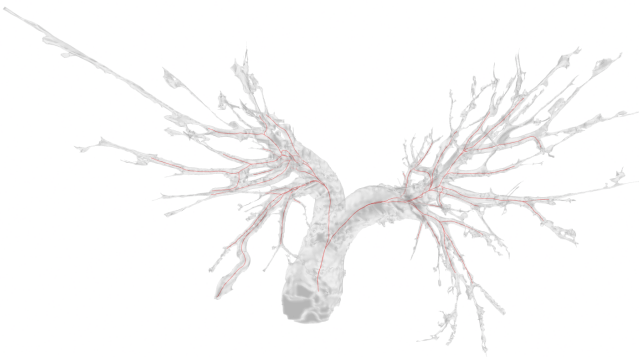
For the first analysis attempt, all the endpoints were selected at the very end of each centerline. Centerlines that were too thin or disconnected from the mesh, were left out of the endpoint selections. In figure 16a some of the 96 selected endpoints from the front view of the mesh are shown. Contrary to expectations, figure 16b demonstrates a cloud of red noise that can be observed around the main centerline model. It was found that many of the selected endpoints were disconnected from the source point in the stem as a consequence of the centerline extraction algorithm failing to compute centerlines through the smaller sections of the mesh.
After concluding that some sections were not suited for the VMTK algorithm, a second attempt was made where only the thicker parts of the mesh were selected as endpoints. For the analysis, 40 endpoints were chosen. Figure 17a shows the resulting centerline model in the original mesh from the analysis of these points. Most of the centerlines of this model cover about half to three quarters of the mesh, leaving a portion of the model for further analysis. The standalone centerline model of the second anaylis can be seen in figure 17b. What can be observed from the model itself as well as this centerline is that it contains relatively many long branches compared to the mammary gland model in figure 11a.



(a) 96 Endpoint selections (in red) layered on top of the model mesh.

(b) The resulting one voxel thick centerline model from the analysis after endpoint selection.

Figure 16: The first attempt at analysing the artery model. The endpoint selections of the mesh in figure 15 used for centerline creation are shown on the left and the resulting centerline model from the analysis are on the right (in red).



(a) The computed centerline model illustrated inside of the original mesh.

(b) The calculated one voxel thick centerline model without the original mesh.

Figure 17: The resulting centerline (in red) from the second analysis attempt on the mesh in figure 15 shown inside of the mesh (left) and by itself (right). For the centerline analysis, 40 endpoints were selected.

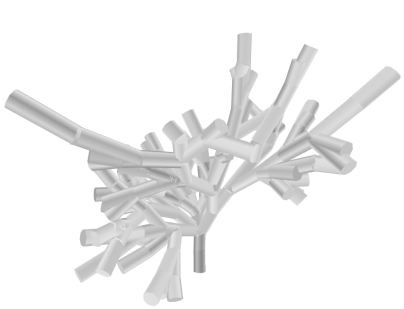
The data collected from the second analysis both from the model itself and the centerline model can be found in table 4. Once again, some observations can be made from this data. First, from the 40 endpoints that were selected, none of them were incorrectly analysed, as can be seen by the number of centerlines in the table. Second, compared to table 3, the number of bifurcations is correct since there are no trifurcations in this centerline model. Next, the total number of iterations is approximately 9 ± 2, where only two of the total number of iterations consist of only the stem. Furthermore, most of the data has a relatively high standard deviation just like in table 3, even though there is more data available. Relatively, the bifurcation angle standard deviation contains

the highest and the point radius the second highest standard deviation. Contrary to the bifurcation angle standard deviation, the relatively large branch length and point radius standard deviations can be observed by the alternation between long and short branches together with thick inner branches close to the stem and thin outer branches further away from the stem.

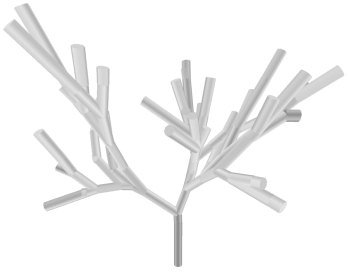
Attribute name (Units)	Mean	Std.
Branch Length (Pixels)	17.25	12.70
Point Radius (Pixels)	6.39	4.48
Centerline Length (Pixels)	150.04	20.86
Stem Length (# Iterations)	2	
Bifurcations	32	
Centerlines	40	
Branches	72	
Iterations	6.82	1.20
Bifurcation Angle (Degrees)	44.40	37.68

Table 4: Attributes extracted from the analysis of the model mesh in figure 15. In the first column, the attribute name an units are shown. The second and third column show the mean and the standard deviation if applicable.

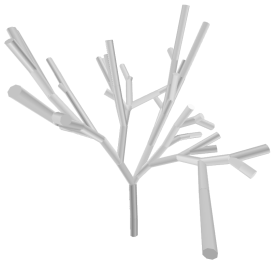
For the L-system generation from the attributes acquired in table 4, three models were generated. The first models in figure 18a shows an abundance of colliding branches and centerlines, indicating that with the given 14 iterations, too many branches generated. On top of that, the preset branch radius was too high compared to the branch lengths, which made it hard to distinguish between branches. The remaining L-systems would therefore be generated with intersection pruning and a decreased segment radius and iteration count. Figure 18b and 18c indicate how these changes affect the models. Additionally, the two models vary in branch lengths, iterations, bifurcation angles and number of branches.



(a) L-system with 96 centerlines and 188 branches.



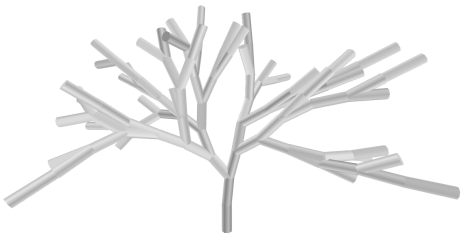
(b) L-system with 46 centerlines, 113 branches and 2.39 segment radius.



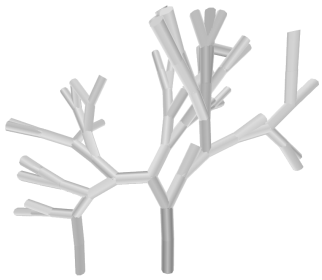
(c) L-system with 38 centerlines, 94 branches and 2.39 segment radius.

Figure 18: A wide variety of randomly generated L-systems based on the data from table 4. The first The upwards curve of the stem was set to 0. The iteration count of the first figure was set to 14 iterations and 12 iterations on the remaining figures.

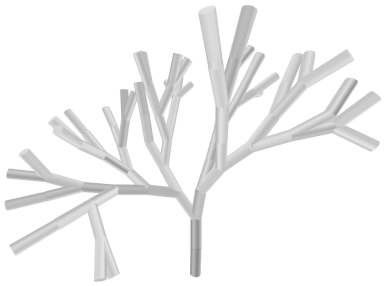
The last three L-systems that were generated without branch length standard deviation are shown in figure 19. The first of the three L-systems in figure 19a, displays frequent outliers that grow in a more horizontal direction. In the second L-system, the iteration standard deviation was reduced, which is demonstrated by the reduced number of outliers and radial development in figure 19b. The iteration and branch length standard deviation in the last L-system were replaced by branch probability in figure 19c. This lead to equal centerline lengths and no more cut off centerlines compared to the systems generated before.



(a) L-system with 38 centerlines, 97 branches, 2.39 segment radius with 12 iterations, no branch length std.



(b) L-system with 40 centerlines, 85 branches, 2.39 segment radius, no branch length std. and one iteration std.



(c) L-system with 38 centerlines, 86 branches, 2.39 segment radius no branch length std, no iteration std. and 76% branch probability.

Figure 19: A wide range of randomly generated L-systems based on the data from table 3. The systems differ in iteration standard deviation, branch probability and have no branch length standard deviation. For all of the systems, the number of iterations was set to 12. The upwards curve of the stem was set to the standard value of 5 with a standard deviation of 2.

Lastly, what can be observed from all the L-systems that have been generated from the mammary gland and artery models, is the increased number of branches compared to the number of centerlines. The data in table 3 and 4 show that the original models require less branches to obtain the same number of centerlines compared to what is produced by the algorithm in figure 13, 14, 18 and 19.

6 Conclusion

To conclude this thesis, we assess in this section whether the main goals have been reached, answer the main research question and discuss important findings from the results. After the conclusion and discussing findings and limitations in section 6.1, we propose suggestions for future work in section 6.2.

6.1 Conclusion and Discussion

In the introduction of this thesis, we explored the importance of visualizing and abstracting 3D models. In particular, abstracting arborized 3D models while preserving important topological features using L-systems. Both 3D model visualization in a web-environment and generating abstracted L-systems from these models are no new concepts [VC20, SC13]. In this thesis, we have focused on combining these concepts and generalizing this functionality for numerous arborized structures. For this, we established primary research goals in section 1.3 and the main research question: "How can a web-based tool that is capable of visualizing, analysing and creating L-system abstractions of arborized 3D models be developed?". Based on this question, the research goals and requirements, key concepts were established in section 2 and related work was gathered in section 3. Using this together with additional software development theory, all the required parts were designed in the methodology in section 4. Furthermore, in section 5, some key results from the web application developed in the methodology section were shown.
To begin with, the visualizing capabilities were shown in section 5.1. These capabilities comply with the primary goal of visualization that was set in the introduction, similarities with other visualization tools like MeshLab [CCC+08] and literature that was discussed in section 3.2. On the other hand, the ability to visualize 3D models is limited by the users (web) resources, which restricts the complexity of models that can be rendered.
Secondly, arborized 3D model analysis methods were discussed in section 4.2 and partially demonstrated in section 5.3, but not all extracted data was used, as shown in section 4.3.3. The extraction methods are identical to related work mentioned in section 3.1 and attains all requirements from section 1.3. Therefore, it has been illustrated that the functionality is sufficient to perform proper statistical analysis on branch-like structures and extract useful L-system parameters. An important condition for the analysis method, which can be noted from section 5.3.2, is a sufficient branch radius. However, no specification was given on which radii satisfy these conditions. Another important condition, stated in the VMTK documentation, requires the 3D model mesh to be close-ended or to utilize the open profiles functionality [ISMA18].
Next, L-system generation from standard and extracted parameters was shown in section 5.2 and 5.3. Changing the upwards and bifurcation angle standard deviation did not seem to result in much of a difference compared to the related work it was derived from [VC20]. In contrast, the iteration and especially branch length standard deviation did have a significant impact on resulting models. Removing the branch length standard deviation made the overall shape look less chaotic and more comparable to the centerline models in both examples. Depending on the type of abstraction one would be interested in, either the iteration standard deviation to add more variety to centerline lengths or the original branch probability can be used for abstractions with equal centerline lengths. Thus, extracted parameters are viable to generate L-systems, particularly branch lengths, radii, angles and iteration standard deviation.

From these key results, we can conclude that the method described in this thesis could be applied to develop a web application that is capable of visualizing, analysing and creating L-system abstractions of arborized 3D models. It has been shown that the web application is capable of being a powerful tool that combines essential parts used in biological research. In addition, easy accessibility and portability can provide workflow-improving benefits and the applied web development methods contribute to extendibility.

6.2 Future Work

In the previous section, limitations regarding the web applications have been discussed. From these limitations, future work can be derived.
Concerning visualization, larger and complex models could be supported with server side rendering [SC13] or by rendering only specific sections of the 3D model. The benefit of this is that the client requires less resources, but at the cost of more server resources and loss real-time navigation and interaction.
Besides visualization, more (arborized) 3D model analysis functionality and methods can be added for specific and general use cases. This could include automated endpoint selection found in Slicer [KPV14], model pre-processing methods for open-ended model analysis and error reduction [ISMA18] or support for analysis of multiple similar 3D models at once.
For L-system generation, additional statistical research can be conducted on the distributions of the parameters and how the data can be applied optimally, particularly when dealing with relatively high standard deviations. This also includes the right estimate of iterations, which consistently failed during the testing phase. Moreover, further experiments on L-system parameters and resulting abstractions may give rise to models that are more consistent and in accordance with the requirements.
To conclude, VR features can be utilized that are readily available within the A-Frame framework. This includes VR interaction methods with the user interface and new visualization capabilities like a free-floating camera that can change the way 3D structures are analysed and perceived.

References

[BCPG10] Frédéric Boudon, Thomas Cokelaer, Christophe Pradal, and Christophe Godin. L-Py, an open L-systems framework in Python. In DeJong, Theodore, Da Silva, and David, editors, *6th International Workshop on Functional-Structural Plant Models*, pages 116–119, Davis, CA, United States, 2010.

[BM12] Sundramoorthy Balaji and M Sundararajan Murugaiyan. Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management*, 2(1):26–30, 2012.

[Bre16] Tom Brewe. A l-system library using modern (es6) javascript with focus on a concise syntax. http://https://github.com/nylki/lindenmayer, 2016.

[C10] Ricardo C. Three.js 3d javascript library. http://github.com/mrdoob/three.js, 2010.

[CCC+08] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool., 01 2008.

[Cho57] N. Chomsky. *Syntactic Structures*. Janua linguarum (Mouton, Paris).: Series Minor. Mouton, 1957.

[CTSH03] D.M.L. Cooper, A.L. Turinsky, C.W. Sensen, and B. Hallgrímsson. Quantitative 3d analysis of the canal network in cortical bone by micro-computed tomography. *The Anatomical Record Part B: The New Anatomist*, 274B(1):169–179, 2003.

[Fou98] B Foundation. Blender. org—home of the blender project—free and open 3d creation software. http://blender.org, 1998.

[GHJV93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*, pages 406–431. Springer, 1993.

[GP17] WSGI Gunicorn-Python. Http server for unix. *URL: http://gunicorn. org*, 2017.

[Gri18] Miguel Grinberg. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[Han97] Chris Hand. A survey of 3d interaction techniques. *Comput. Graph. Forum*, 16:269–281, 12 1997.

[ISMA18] Richard Izzo, David Steinman, Simone Manini, and Luca Antiga. The vascular modeling toolkit: A python library for the analysis of tubular structures in medical images. *Journal of Open Source Software*, 3:745, 05 2018.

[JLR94] Christian Jacob, Aristid Lindenmayer, and Grzegorz Rozenberg. Genetic l-system programming. In *PPSN*, pages 334–343, 1994.

[KK02] Nora Koch and Andreas Kraus. The expressive power of uml-based web engineering. In *Second International Workshop on Web-oriented Software Technology (IWWOST02)*, volume 16, pages 40–41. Citeseer, 2002.

[KP+88] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[KPV14] Ron Kikinis, Steve D. Pieper, and Kirby G. Vosburgh. *3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support*, pages 277–289. Springer New York, New York, NY, 2014.

[MM07] Jibitesh Mishra and Sarojananda Mishra. *L-system Fractals*. Elsevier, 2007.

[Moz15] Mozilla. A-frame, a web framework for building virtual reality (vr) experiences. http://github.com/aframevr/aframe, 2015.

[oIC16] Visual Computing Lab of ISTI CNR. A javascript, client-side, mesh processing tool. inspired by the well known meshlab. https://github.com/cnr-isti-vclab/meshlabjs, 2016.

[Par16] Emmanuel Paradis. The distribution of branch lengths in phylogenetic trees. *Molecular Phylogenetics and Evolution*, 94:136–145, 2016.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 1990.

[PO08] Bernhard Preim and Steffen Oeltze. 3d visualization of vasculature: an overview. *Visualization in medicine and life sciences*, pages 39–59, 2008.

[PVS+09] Marina Piccinelli, Alessandro Veneziani, David A Steinman, Andrea Remuzzi, and Luca Antiga. A framework for geometric analysis of vascular structures: application to cerebral aneurysms. *IEEE transactions on medical imaging*, 28(8):1141–1155, 2009.

[Ree08] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

[SC13] Bartosz Sawicki and Bartosz Chaber. Efficient visualization of 3d models by web browser. *Computing*, 95(1):661–673, May 2013.

[VC20] Fons J Verbeek and Lu Cao. L-systems from 3d-imaging of phenotypes of arborized structures. *Fundamenta Informaticae*, 175(1-4):327–345, 2020.

[vEHP05] Caroline van Ertbruggen, Charles Hirsch, and Manuel Paiva. Anatomically based three-dimensional model of airways to simulate flow and particle transport using computational fluid dynamics. *Journal of Applied Physiology*, 98(3):970–980, 2005. PMID: 15501925.