Development and usage of the instruction language Fcode for neural growth simulations

Master Thesis
September 2008 to March 2009

Tutor: Frédéric Zubler
Supervisor: Prof, Rodney J. Douglas
Abstract

We present a new model for axonal growth. This model mixes a probabilistic and a competitive approach. The model is reactive to its simulated chemical surroundings and is applied in a physical plausible environment named cortex 3d. We show how we simulate the axonal growth of neurons in three dimensions which have similar properties as observed in reconstructed neurons of cats visual cortex layer 2/3 and their look and feel. Along with this we present new measurement approaches for axonal growth. The model has been encoded in a language named Fcode where we established the representation in XML in order to be fast in coding and for further use of Fcode. Moreover, helpful tools have been implemented such as graphical Fmachine designer and a Cortex designer to enhance the usability of Fcode.
# Contents

1 Introduction 7  
1.1 Fcode 7  
1.2 CX3d 8  
1.3 Task 9  

2 Designing a Genome 10  
2.1 Gene Regulatory Network 10  
2.2 XML as descriptive language for cell behaviour 11  
2.3 XML-Genome 12  
2.4 Fcode 12  
2.5 XML as descriptive language for Gene Regulatory Networks 13  
2.6 XML-Schema of the Genome 13  
2.7 A Simple Machine example 13  

3 An axon growth model 18  
3.1 Analysis of reconstructed cells 18  
3.1.1 Reconstruction of cells 20  
3.1.2 Clues of Features 20  
3.2 A growth cone model growth 23  
3.2.1 Existing models 23  
3.2.2 Our model 23  
3.3 Axonal growth model 24  
3.3.1 Existing Neurite Growth models 24  
3.3.2 Approaching a new growth model 25  
3.3.3 Fcodeing the model 29  
3.4 Methodology: How to compare Neurons? 34  
3.4.1 Existing approaches used 42  
3.4.2 Our approaches 45  
3.5 Measurement application 45  
3.6 Simulation setup 46  
3.7 Results 46  
3.7.1 Exploration of production speed of ‘o’ space 47  
3.7.2 Exploration of diffusion constant of ‘o’ space 49  
3.7.3 Exploration of consumption of a growth cone space 51  
3.7.4 Exploration of probability of a the main branch to branch space 53  
3.7.5 Exploration of probability of a sub-branch to die space 56  
3.7.6 Exploration of probability of a sub-branch to branch space 58  
3.7.7 Best matching neurons to reconstructed neurons 61  
3.7.8 Further observations 63  
3.8 Criticism on the model 63  
3.9 Where to head now? 64  

4 Fcode Behavior Designer 65  
4.1 Fmachine composer 65
## CONTENTS

<table>
<thead>
<tr>
<th>5</th>
<th>Cortex Designer</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Building the Cortex Designer</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>Taking it to the next level</td>
<td>71</td>
</tr>
<tr>
<td>6</td>
<td>Summary</td>
<td>72</td>
</tr>
<tr>
<td>A</td>
<td>XML</td>
<td>74</td>
</tr>
<tr>
<td>A.1</td>
<td>XML example File</td>
<td>74</td>
</tr>
<tr>
<td>A.2</td>
<td>The XML Schema</td>
<td>75</td>
</tr>
<tr>
<td>B</td>
<td>Legend</td>
<td>85</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 Gene Regulatory Network .............................................. 11
2.2 XML-Serializable interface ........................................... 12
2.3 Genome class .......................................................... 12
2.4 Starting the main axon from the soma ............................... 13
2.5 grow until a chemical is expressed enough and then fork and recall the same machine again. .......................... 14
2.6 Bifurcation with attraction to the chemical A ...................... 15
2.7 Bifurcation with attraction to the side .............................. 16
2.8 Multiple bifurcation simulation result .............................. 17
3.1 Neuron j1391p2 .......................................................... 18
3.2 Neuron j1591 ............................................................ 19
3.3 Neuron j2187J8 ......................................................... 19
3.4 Neuron j2187J8 .......................................................... 20
3.5 Smoothed neuron ....................................................... 21
3.6 Branching over 30° ..................................................... 22
3.7 Branching under 30° .................................................... 22
3.8 Side branching .......................................................... 22
3.9 Stem orders ............................................................. 23
3.10 Growthcones ........................................................... 24
3.11 An overview of the growth model in action ....................... 28
3.12 cellbase ................................................................. 29
3.13 secrete ................................................................. 29
3.14 startbranch ............................................................. 30
3.15 mainaxon ............................................................... 30
3.16 grow ................................................................. 31
3.17 consume ............................................................... 31
3.18 sidebrancher ........................................................... 32
3.19 kickstartgrow .......................................................... 33
3.20 growandfork ............................................................ 33
3.21 sidebrancher2 .......................................................... 34
3.22 The box-counting method/ Minkowski-dimension .................. 35
3.23 The coastline measurement .......................................... 36
3.24 The Hausdorff-dimension ............................................ 37
3.25 The dilation method ................................................. 38
3.26 The mass radius ....................................................... 39
3.27 The Sholl-Analysis ................................................... 40
3.28 The lacunarity .......................................................... 41
3.29 Depth ................................................................. 42
3.30 Total length ........................................................... 43
3.31 Horton-Strahler number ............................................. 44
3.32 Gaton-Watson ......................................................... 45
3.33 in blue: Measurements of section 3.4.2 applied on reconstructed neurons in red: Mean of the measurements. ........................................ 46
3.34 Altering the production speed of ‘o’ .................................. 48
3.35 Evaluation of Measures on secretion speed changes ............. 49
3.36 Altering the diffusion constant of ‘o’ ........................................ 50
3.37 Evaluation of Measures on diffusion constant changes .................. 51
3.38 Altering the consumption of a growth cone ................................. 52
3.39 Evaluation of Measures on consumption of a growth cone changes .......... 53
3.40 Altering the probability of the main branch to branch .................... 54
3.41 Evaluation of Measures on probability of the main branch to branch changes ...... 55
3.42 Altering the probability of a sub-branch to die ............................... 57
3.43 Evaluation of Measures on probability of a sub-branch to die changes ........ 58
3.44 Altering the probability of a sub-branch to branch ....................... 59
3.45 Evaluation of Measures on probability of a sub-branch to branch changes ...... 60
3.46 Measurements of section 3.4.2 for the best matching neuron ............... 61
3.47 Shape of the best matching neuron ........................................... 62
3.48 Stem orders ................................................................. 63

4.1 Screenshot of the Behavior Designer Tool .................................. 66
4.2 Screenshot of the property editor .............................................. 67
4.3 Screenshot of a machine simulated in a cell ................................. 67

5.1 Screenshot of the Cortex Designer ............................................. 69
5.2 Screenshot of the property editor of a the Cortex Designer, properties of a cell are shown .................................................. 70
5.3 Screenshot of a simple simulated cortex part .................................. 70

B.1 The meaning of the symbols used in Fcode diagrams ..................... 85
List of Tables

3.1 Standard values for the exploration of the models parameter space . . . . . . . . . 47
3.2 Measurement values for the best matching grown neuron . . . . . . . . . . . . . 62
Chapter 1

Introduction

Self-organisation is the ability of a system to create itself without external constructor, only having the intrinsic properties and its environment at hand. It is unknown how biological organisms are able to spawn their complete complex structure of one single stem cell only given the genetic code. If one was to transfer this ability of self-organisation to other fields there would be many applications for it, for instance in robotics self-creating and self-repairing machines, in architecture self-constructing buildings or in computer science self-constructing logical circuits.

In this thesis we are especially interested in the self-organizing capability of the mammalian cortex. The cortex arises from a few cells at the tip of the neural tube, these few cells then build a computational entity that could not be duplicated yet by any software nor hardware that mankind can create. All of this is done without interference of an external intelligent help. The cells "know" what they have to do and where they have to migrate in order to build this complex architecture we observe. Since this architecture supports the computation performed by the human brain, it is worth investigating the process of construction of the cortex in context of morphology. Can we simulate the morphological growth process of a human brain such that we get comparable structures? We want to make a step towards understanding this processes.

We present a model that simulates the morphology of the single computational units of the brain, the neurons. We introduce a genetic description language. Using this language we develop artificial neurons that construct themselves. We implement a new model, axonal growth model, show simulations of it and compare the results to real reconstructed neurons. The idea of self-construction is then taken one step ahead and we show the concept of a Cortex Designer. This Cortex Designer enables a modeler to define a cortex with very few instructions. We demonstrate how we are able to take this abstract description and simulate the growth of this cortex. All of this work is done in the context of SECO.

1.1 Fcode

In this thesis we model biological processes in the context of neural growth. To be more precise, we model axonal growth patterns. In order to do so the biological description language Fcode was used. Fcode was developed by Zubler and Douglas [1] and enables one to program the cell behavior. It was developed in the context of the EU project SECO on self-organisation. The Fcode is analogous to the genetic code for the expression of a certain function, and the Fmachine is the complement of Fcode transcribed into a functional proteomic system. The Fcodes encode minimal behaviors that define the alphabet of the cellular functional language. Each of these minimal behaviors represents a function a cell can execute. An Fmachine is composed of a set of elemental primitive behaviors that are assembled through the inter-connection of their input and outputs. Fmachines are themselves composable in this way, so that a combination of Fmachines is itself a Fmachine. Notice that the Fcode is an inert, non-functional complement of its Fmachine. Transcription of the code evokes the function. The proposed minimal set of primitive behaviors consists of:
• **Detect:** Detect is able to sense chemical concentration, intracellular or extracellular. In the external case a gradient that shows where the chemical concentration is highest in the surrounding area. Both of these values can be read via the corresponding output by another minimal behavior.

• **Move:** This behavior is designed to enable a cell to migrate, or a growth cone to elongate a neurite. Move has 2 input ports speed and direction. Based on these two inputs a random inference factor and in the growth cone case based on the current growth direction a movement direction is calculated. This is then used to move the cell respectively to elongate the neurite in this direction. The length of this elongation can then be read from the output port elongation length.

• **Kill:** Kill decides via an input, namely the probability to kill, whether it should stop the current running Fmachine. As an output it has the port: has been killed which indicates whether the kill has been executed.

• **Instantiate:** Instantiate spawns a new machine and adds it to the parent Fmachine of Instantiate. It has the input probability to instantiate and the output has instantiated

• **Fork:** Fork is a minimal behavior that can spawn a neurite and place a Fmachine in it. There are three cases. First case is that the fork is in the soma which then decides if a new axon or dendrite is started from the soma. Second, the fork is in the growth cone when fork is executed. In this case the current growth cone splits and bifurcates into two new sub-branches each of which contains a machine. In the third case the fork is also in the growth cone, but now it spawns a side-branch on execution. Fork has the input probability to fork which is the basis of the calculation whether it is executed or not. As output it has has forked which tells whether the fork has been executed.

• **Secrete:** Secrete can release a chemical intracellular, extracellular, or embedded in the membrane of the cell. Secrete (negative) consumes a chemical. It has the input secretion speed that decides how much of the chemical is released or consumed per time step.

In addition the Fcode set contains connective filters and logical gates that modify output and input values transmitted between minimal behaviors. So far we use the following connectives:

• **Positive:** Positive has one port: output which always gives the maximal value.

• **Linear Filter:** The Linear Filter has a port input. The value from the input is scaled linearly and given to the port output

• **Step-function Filter:** The Step-function Filter has an input port. The value given into the port is checked whether it excite a certain threshold. If this is the case the maximal value is given to the output.

• **Multiply:** Takes multiple inputs and multiplies their values and gives it to the output.

These are the minimal behaviors, filters and logical gates that have been taken in order to model the axonal growth thus far. There exist a few more but those have not been used in this thesis and we leave the task to Zubler and Douglas to describe them.

### 1.2 CX3d

CX3d (Cortex 3d) is a simulator for neural growth. The simulator constrains a physical environment in which cells and their interaction with the environment can be simulated [1]. In addition a biological growth description language has been developed which is called Fcode [2]. This Fcode describes the behavior of the neural growth. The idea of Fcode is to give the possibility to model complex cell behavior in a fairly simple and abstract manner. This includes migration of cells, branching processes of axons and dendrites, establishment of synaptic connections all of it in interaction with a chemical and physical environment. A model written in Fcode can be seen as the genetic information given to a cell which tells it how to behave if certain
environmental conditions are met. Moreover CX3d consists of a framework for simulations of Gene Regulatory Network along with the possibility of cell splitting. Through these abilities CX3d is able to simulate complex cortex growth models.

1.3 Task

We wish to instruct axonal growth in such a manner that the cells of CX3D grow to form neurons whose morphology is similar to that of real neurons. In order to do this, we started by analyzing mature neurons from cat visual cortex that have been reconstructed in 3D. In order to analyze these cells and compare them to those generated by CX3D, we developed measurements of the similarity between the model and real cells.

To model the growth of axon we also required a growth cone model that could express the movement behavior of a real growth cone. Furthermore we needed a modeling language to express that function, for which we used Fcode. Fcode is a very flexible language which is capable to describe growth processes of neurons on an abstract level. But it had no textual format which made it hard to develop models fastly and easily. Moreover it was not possible to reuse models previously. Therefore it was crucial at this point in the project to have a human readable form of the language, to develop more enhanced cortical/neuronal growth models. Therefore such a textual form had to be developed.

Since XML is fairly human readable and a commonly used self-describing format, we chose to use XML for specifying Fcode. Even though XML is easy to understand and flexible, we found it difficult to write code directly in this form. Instead, we have developed a tool that permits the cell programmer to assemble machines using a mouse driven CAD scheme. We expect that in future we will develop this visual CAD scheme to encompass the design of entire cortical areas and cortical plates. This tool should let the user click together an entire layered cortex of chosen cell types, and specified interconnectivity. The CAD program should then generate the Fcode to be inserted into the precursor cell or cells of CX3D, which cell would then spawn into the desired cortex.

Our first milestone has been to enhance the functionality of CX3D in order to be able to save and load growth models written in Fcode and Gene Regulatory Networks in a genome-like manner. The format used for these persistent forms of the genome is XML.
Chapter 2
Designing a Genome

The modeling language Fcode seemed to be the modeling language that suited best for us in order to model axonal growth. What was missing though was the possibility of having a handy way to develop in Fcode. Fcode was only available as classes in Java, no persistency was given. We decided to develop a persistency model for it to create and test axonal growth models faster. Furthermore we integrated the work of Pfister and Douglas [3] about Gene Regulatory Networks into our persistency model. In order to work with lineages of cells and to spawn different cell types integrated in one model. Even though this has not been used in our work about axonal growth, but it will be helpful for later projects. As persistency language we chose XML as it is human readable, standardized and always transferable to other formats. In the following sections we briefly describe our persistency model and show how it is implemented. We call this XML-Representation an XML-Genome.

The genome of living organisms encodes the construction plan of it. It defines which cell types are followed by each other and it determines which functions these cells will have. This encoding is done in the DNA-strands which is a sequence of genes. We want to model the genome-like structure that defines the cell lineage and the functions of the different cell types. Therefore our model consists of a Gene Regulatory Network designed by Pfister et al. [3], a Fcode [2] and a promoter part. The Gene Regulatory Network part will tackle the lineage of the cells. The Fcode part defines the functions the cells can incorporate. The promoter part builds the link between Gene Regulatory Network and the functional Fcode part. Depending on the cell type the cells have different duties and therefore spawn different Fmachines to have different abilities. As observed in nature we save the information about our genome in a sequential way using XML.

2.1 Gene Regulatory Network

The Gene Regulatory Network can be viewed as expressed genes influencing the expression of other genes. This can be abstracted as a directed graph which shows the interdependencies of the genes. The genes work as promoters and express a functional behavior of the cell they are expressed in, which leads the cell to become a certain cell type.
2.2 XML as descriptive language for cell behaviour

Through analysis, it appeared that most of the minimal behaviors of Fcode where available as code. Each minimal behavior of Fcode was designed as a separate java class. We had previously attempted to create a (non-XML) string based language for these minimal behaviors. And we had even developed a parser for this language. One of our first tasks in moving to XML was to develop a framework that could handle the XML serialization and de-serialization in order to be able to save and load Fcode form an XML-Document of it. This XML-Document format is called the XML-Genome. For each minimal behavior a suitable XML representation had to be found and the Java code had to be changed in order to handle the persistency properly. Moreover, it was necessary to create a schema description in the form of XML-Schema. A schema description is important for the validation of the Genome code, which assists the neuronal genome designer to create documents that have the correct syntax for the XML-Genome, and to ensure that only correct syntax is fed to the XML-Genome interpreter (transcription processor).

Figure 2.1: A gene regulatory network. Shown are regulatory genes influencing the expression of each other regulatory genes. These regulatory genes influence the promotion of functional genes, which then invoke certain abilities of a cell. This defines the behavior of the cell and hence the cell type.

The chosen code representation is that a gene complies with one class that has a list of influencing factors to other genes. The promoters are a separate class that contain conditions under which a certain Fmachine is instantiated inside a cell. The conditions are given a threshold and it is stated whether the conditions are true if the threshold is reached or false if not. These conditions can be combined using the boolean operations *And* and *Or* through this all boolean functions can be expressed.
2.3 XML-Genome

The XML Genome consists of two parts the Fcode part which describes the behavior of the cell and the gene regulatory network part that describes how the cell should behave on division of itself.

2.4 Fcode

XML is a hierarchy oriented description language. This means it consists of parts that consist of parts that consist of parts again. The language Fcode is analogous. It encodes minimal behaviors composed to implicit function. The function is realised by transcription of the Fcode into its complementary Fmachine. The Fmachine can be embedded into another Fmachine again and so forth, in an hierarchical fashion. Multiple of these Fmachines, or their code form, constitute a genetic code. This code is then inserted into a cell, which is able to transcribe code and execute its machine using Gene Regulatory Networks. How the Gene Regulatory Networks are implemented into the Genome code is explained in the next section 2.1. Because both XML and Fcode are hierarchical and also the implementation of Fcode is done in a hierarchical manner it is natural to use a Component Composite Model for implementation. This requirement leads to the following Interface that all XML capable classes must implement in order to be serializable to XML and the reverse:

```java
public interface XMLSerializable {
    public StringBuilder toXML(String ident);
    public XMLSerializable fromXML(Node xml);
}
```

This puts each XML-serializable class into the responsibility to serialize/de-serialize itself and to propagate the serialization/de-serialization process to its components. The XML-Genome is collected in a class which is responsible for creating the XML-File and reading it back in. This means that it starts the XML-serialization and de-serialization. For the reconstruction from XML it uses the help of a XML-Factory class that knows how to de-serialize all the XML-serializable objects. In order to clarify this, the figure 2.3 shall give the reader more overview.

![Figure 2.2: XMLSerializable interface](image)

![Figure 2.3: The UML diagram of the Genome Class](image)
2.5 XML as descriptive language for Gene Regulatory Networks

The Gene Regulatory Network was already implemented in CX3d. It consists of a few classes that had to be integrated into the framework of XML-Persistency. Also the Gene Regulatory Network had to be represented in a suitable XML form. As well as the minimal behaviors the Gene Regulatory Network had to be integrated into the XML-Schema which serves as a validation reference for all the genomes produced.

In terms of XML each gene is a node whereas each influencing factor is a subnode. The promoter is a separate node that contains its condition as a tree-like structures of ANDs and ORs of thresholded values. These classes must therefore also implement the XML-serializable interface and the factory must be able to recognize them in the de-serialization process. As it can be seen in the diagram 2.3, the genome is capable of registering and reconstructing the promoters and the genes.

2.6 XML-Schema of the Genome

The descriptive form of the genome is held in XML, the XML description of the genome has a standardized description, which is held in XML-Schema. This schema is included in the appendix A.2 of the document. All generated XML-Documents will comply to this schema. So does the genome-file of the example machine shown in the next section.

2.7 A Simple Machine example

In order to show the reader what the language Fcode and its XML representation are capable of, we present here an example of a simple Fmachine. This machine is based on recursion and takes into account the self-similarity observed by Binzegger et al. [4].

This is a very simple but nevertheless powerful genome. To give you a better insight, it is shown here schematically:

![Machine schematics](image1)

(a) Machine schematics

![What the machine does](image2)

(b) What the machine does

Figure 2.4: This machine starts an axon from a soma. By detecting in what direction the highest concentration of the chemical E lies. After creating the initial branch of the axon, the growth-cone of the axon gets driven by the machine bifork. This machine is responsible for the recursive growth.
Figure 2.5: This machine drives the growth cone of an axonal branch, into the direction of the chemical gradient of E. Using the minimal behaviors detect and move. Dependent on the concentration of E the axon will bifurcate and start two sub-branches. Both sub-branches have growth cones that again contain an instance of the bifork machine. This mechanism is established by the minimal behaviors fork and detect. The parent branches growth cone is stopped immediately after the bifurcation by the kill behavior.

Obviously this genome contains recursive elements these elements lead to self similarity in the growth. Moreover this growth would never stop this is the reason why one need to stop it somehow. This was done by letting each new branch have a smaller diameter as soon as the diameter went below a certain threshold the growth and the bifurcation was stopped. This part was hard coded in this example. The XML code that describes this machine is listed below A.1. Because the minimal behaviors have some probabilistic parameters, the shape of the neuron is not defined exactly. Instead, the code defines the neuron type in a qualitative manner, and the instances depend on prevailing conditions. The outcome of three simulations are shown on the next few pages:
Figure 2.6: This neuron is attracted by the chemical A which is shown as a red stripe here. The artificial layer of chemical A is horizontally oriented. The chemical A is gaussian distributed.
2.7 A Simple Machine example

Figure 2.7: This neuron is attracted by the chemical D which is shown as a red stripe here. The artificial layer of chemical D is vertically oriented. The chemical D is gaussian distributed.
Figure 2.8: This figure shows the result of simulations of a slightly altered bifurcation genome. The probability to branch is changed in each neuron. Right there is a high probability left there is a low probability.
Chapter 3

An axon growth model

Our goal was to construct a model of axonal growth that is able to grow neurons that look like real neurons. Moreover it had to be biologically plausible. In the next few sections we will show the reader what we achieved.

3.1 Analysis of reconstructed cells

Since we did not have the possibility to observe growing cells in vivo, we have to rely on already grown and reconstructed neurons so far. These cells where reconstructed as described in the following two chapters. In this master thesis only the pyramidal neurons from layer 2/3 were included as data sources namely:

- Neuron: j1391p2

![Figure 3.1: The shape of the neuron](image)

- Neuron: j1591
3.1 Analysis of reconstructed cells

- Neuron: j2187J8

Figure 3.2: The shape of the neuron

- Neuron: j986

Figure 3.3: The shape of the neuron
These were taken as a reference to build a growth model in which the resulting neurons were supposed to have the same key features as described in the section Methodology 3.4.

3.1.1 Reconstruction of cells

In this section we are going to explain how the neurons were reconstructed from the cat visual cortex. The cats were anesthetized and paralyzed. Electrophysiological recordings where taken from a selected neuron and their receptive field was determined. The neurons where stained with HRP. The cat brains were fixated and cut in slices of 80 µm. The slices where magnified and on each slice the parts of the neuron were identified and marked. On each slice parts of the axon, parts of the dendrite, the soma, the buttons and the synapses where manually identified and digitalized. The edge points of the axon and dendrites were then interlinked between the slices. Hence a three dimensional reconstruction of the complete neuron resulted. This data was exported into a format readable by Matlab.

3.1.2 Clues of Features

During this thesis we found interesting structures in the reconstructed cells. The reconstructed cell data was available in a Matlab format. This format had to be exported into a Java readable file. These files could then be imported in an analytical tool that was written in Java developed by us. On the first glimpse all the data looked very jittery. We assume that the jitter is a result of obstacles like blood vessels, neurons, other cells and due to measurement errors. Therefore we decided to apply a low pass filter in order to remove the jitter and make the cells more smooth. This brought a great deal of overview, which figure 3.5 shows.
We aimed at finding a model that would explain the shape of these neurons. The first important observation you can make is that the axon coming out of the soma produces a lot of branches and projects further down to layer 5. As implied by their name the neurons' somas are all situated in layer 2/3. As expected the observed neurons are innervating layer 5. We looked at the branches of the main axon that is going from layer 2/3 to layer 6. We discovered most of the angles are higher than 30° which we found interesting (figure 3.6). We remind the reader to be careful not to be distracted by the two dimensional projections of the neurons shown here. This lead to the assumption that most of the branching progresses from the main axon are not due to bifurcation, but due to side-branching. Furthermore if you look at pictures of stained developing neurons you can see the side-branching effect (figure: 3.8) In this thesis bifurcation means that the growth cone is split into two and both growth cones keep on growing. Important fact is that an existing growth cone splits rather that a new one is built. Both growth cones try to keep the direction the axon has been growing before with minor adjustments so that they can coexist. This leads to a small angle between both growth directions. Side-branching is the process where a new growth cone is built during the growth of a neurite. This growth cone sits on the outside of the membrane of the neurite and starts therefore to grow almost perpendicular to the originating neurite. This results in a bigger angle between the two branches.

By looking at the reconstructed neurons in 3d we saw that the branching process is usually due to side-branching and not due to bifurcation. Inspired by the work of Binzegger [4] we observed that these sub-branches re-branch in a fairly self-similar way. Also considering ideas of branching descriptions by Strahler [5] we came up with the following definition:

A stem-branch is a coherent neurite that never has sharper turns than 30° on a branching point. (a stem-branch can therefore consist of multiple bifurcations).

If there is a branching point where one outgoing branch is higher than 30° then this is not counted to the current Stem-Branch and a new one is built. This leads to a Stem-Branch hierarchy. We speak of Stem-Orders. Where the soma is the origin of the first Stem-Branch of order 0 then the next order are the Stem-Branches of order 1 that originate in the Stem-Branches of order 0 and so on. We found that in the data this usually goes up to a Stem-Branch of order 6 or 7. To illustrate that the first three Stem-Branch orders of cell j986 are shown in the figure below 3.48. This definition of Stem-Branch was taken and the hierarchical model was implemented in the analysis tool which allows to show the different Stem-Orders of a reconstructed cell. Furthermore it can be observed that in general only in the layers that are supposed to be innervated in this case 2/3 and 4, Stem-Branches of order 1 are established.
3.1 Analysis of reconstructed cells

Figure 3.6: Shows branches that have more than $30^\circ$ to the growth direction

Figure 3.7: Shows two branches that have less than $30^\circ$ to the growth direction (bifurcation)

Figure 3.8: A real developing neuron where one can observe the side-branching (white)
What we observed now was that the Stem-Branches of lower order are in general longer than the ones with higher order. The same holds for the number of branches, the higher the order the less sub-branches there are.

All these observed key features lead to the model defined in section 3.3.3. But in order to let neurons grow we had to implement a growth cone that grew the axon in a plausible way.

### 3.2 A growth cone model growth

For modeling a growth cone we had to ask the question how a growth cone works and how we can abstract its behavior into a model. If one observes a growth cone, one can see the filopodia all around the distal end of the neurite (figure 3.10a). This implies that the growth cone is able to detect in its very neighborhood the concentration of the chemical the filopodia are sensitive to. In a winner-take-all manner it is then decided where the axon grows to. The filopodia with the highest influence, meaning the filopodia which senses the highest concentration wins and the axon elongates in this direction.

For the simulation of axonal growth we needed a sufficient model for simulating growth cones in a 3d environment which Cortex 3d is. Since we model on an abstract level there is no need for a model that takes into account the filopodial behavior. It has to be a computationally efficient and fast one.

#### 3.2.1 Existing models

We looked at the model of Li at al. [6] to go more into the direction of actin cytoceleton modeling of a growth cone and it is too precise, not optimized for 3d and does not take into account guidance cues, which made this model inappropriate for our purposes. Moreover we looked at the model of Buettner [7] who modeled the behavior of filopodia in a growth cone. It does not take into account the guidance cues and the movement of the growth cone. For our purposes the modeling of the complex behavior of filopodia goes too deep and would take up too much computing power since we expect that there are multiple axonal branches growing at one time. We needed something more abstract and optimized for three dimensional growth.

#### 3.2.2 Our model

Based on the fact that growth cones follow guidance cues [8] we knew that we needed a model of a growth cone that is able to elongate the axon and follow a gradient in a chemical environment. Therefore we needed two parts for our model: A chemical detection model and
an elongation model. The elongation is pretty simple. Through the minimal behavior move we can give the distal end of an axon the order to elongate along the gradient that was fed into the move port direction. The second part modeling the detection of the chemical had to be implemented into the minimal behavior detect. We needed a way to determine what the gradient of the to be detected chemical is, hence the direction of axonal elongation. This is the reason why we came up with the following model. We did not want to model each filopodia separately, therefore we made the assumption that the filopodia are spread around the distal end of the axon. Each tip of the filopodia on a sphere which has its center at the growing axonal end (figure 3.10b). These tip points measure the intensity of the chemical that the growth cone is sensitive to. In nature there are many more influencing factors for which filopodia decides the direction of the growth but in general the one(s) with the highest concentration is taken. To model these influences we added some random noise to each measured value on the sphere in order to crudely compensate. In our model filopodia with the maximal value is the winner and the direction is calculated by subtracting the position of the distal end from the position of the winning filopodial tip. The branch will therefore elongate into this direction. This model gives a good enough approximation of a growth cone in order to simulate a growing axon following a chemical guidance cue. Through the possibility of adding step- and vector-inverter-filters even the attraction, repulsion and the contact dependency can be modeled [8], which means there should be few limitations in the model we created.

![Real growthcone](https://via.placeholder.com/150) ![Growthcone model](https://via.placeholder.com/150)

Figure 3.10: A growth cone of a real neuron where you can see the filopodia in red and the neurite in green. The growth cone model with a sphere of measuring points (red) around the tip of the neurite.

### 3.3 Axonal growth model

In the next few sections we discuss existing growth models and we describe the growth model we developed.

#### 3.3.1 Existing Neurite Growth models

As Van Pelt et al. summarized nicely in the book Modeling Neural Development [10], there are many approaches throughout the literature in order to model neural growth. Most of the models are developed for dendritic arborisation and not for axons. Since both are neurite growth we do neglect the difference in this thesis and also look at dendritic models for neurite growth in order
3.3 Axonal growth model

One of the most popular models is the BE model and further developments, BES and BEST by van Pelt et al. [11]. These models use a probabilistic approach for modeling the branching process. The model is though missing its biological foundations and justifications. It uses no external guidance cues for where to grow nor does it use an intracellular chemical model for growth. There is no simulation approach in a physical environment such as Cortex 3d given. Furthermore this model only takes into account bifurcation, but as we observed, there is much more side-branching involved in the process.

There are guidance oriented models such as the model of Niell [12] which is influenced by the environment and follows the "synatropic hypothesis" of branching only when there is a potential target for a synapse around. But this model does not take into account intracellular processes and growth cone competition.

Other models use a growth competition and internal diffusion approach such as the work of Van Ooyen [13]. But also this model only insists on bifurcation and has no interaction with its environment, it focuses only on the intracellular processes.

We even looked at models for river basins as in the work of Cieplak [14]. This approach models a probabilistic growth of the tips of a river basin. But it could not satisfy our needs for a competition oriented growth and growth guidance.

What is interesting throughout the literature there is this consensus of self-similarity which encouraged us to make a model that reuses its description in a way as it occurs in recursion or fractals.

As described, there are models that either use guidance, diffusion or growth competition. To our knowledge there is no model that does combine these ideas. Though it was suggested to combine these to make more complicated models by van Pelt [10].

3.3.2 Approaching a new growth model

Our goal was to combine the idea of side-branching, consumption production (growth competition), growth guidance and self-similarity (recursion) into one model which could be simulated in a physically plausible environment such as Cortex 3d.

In order to reach this goal we used the Fcode language to construct our model. After having established a plausible growth cone behavior and after identifying key features found in the axonal data we began to create a plausible model for axonal growth. To make use of these properties we wanted the model to consist of very few Fmachines that make use of re-invocation of themselves which leads to recursion and therefore to fractal behavior (self-similarity).

Inspired by the work of Van Ooyen [13] we defined that the growth of the simulated axons should contain a consumer producer principle where the soma produces a chemical 'o' that the growth cones need and consume in order to be able to enlarge the axon. The chemical 'o' is released cell internally. This chemical 'o' represents some sort of tubuline that is needed for elongation of neurites.

The primary bases for our model is the growth of Stem-Branches of order 0 to 7 where Stem-Branch of order 0 is treated specially and will be explained in the following paragraph. The other Stem-Branch orders from 1 to 7 are designed self-similar and make use of recursion to spawn each other but all Stem-Branches have in common that they are attracted by a certain chemical in the environment.

For Stem-Branches of order 0 (which is usually only 1, the main axon) we took into account the work of Kalil et al. [15]. The first side-branches do usually not appear before the growth cone of the Stem-Branch of order 0 has reached or is close to its target layer. This fact made
it necessary to implement some sort of plausible delay before the first Stem-Branches of level 0 begin to grow. But already in this process of down-growth the future growth cones are established as it is suggested by Kalil et al. [15]. Even though they have been placed they do not start to grow instantaneously. This delay is implemented by waiting for ‘o’ to reach a certain threshold of concentration at each growth cone.

The fact that only certain layers are innervated by the axon is also addressed by our model. The probability for having an outgrowing Stem-Branches of order 1 is dependent on the chemical environment of the layer to be innervated. Due to the re-invocation of the same machine that grows Stem-Branches of order 1 all the further spawned Stem-Branches are attracted by the same chemical as their parent. In other words Stem-Branch of order 0 spawns a tree of Stem-Branches in a certain layer which is also known as innervation.

The speed of the growth of each growth cone depends on the chemical ‘o’ that is produced in the soma. But not only that chemical ‘o’ has to be produced, it also has to be diffused along the already grown part of the axon to its growth cones (If the reader is interested in the methodology of diffusion please read Zubler and Douglas[1]). Furthermore there is the aspect of competition between the growth cones since there is only one production place and there might be multiple growth cones growing in parallel. These two reasons lead to the fact that the more growth cones there are the slower the Stem-Branches grow. This should induce the effect of shorter and shorter Stem-Branches the higher the order is. Since there are more branches of order higher than of order 0 and the higher ones are spawned later on. This means the competition increases with order which leads to shorter Stem-branches in higher orders.

It moreover needs to be clarified how to stop the growth cones that they do not grow forever. In the Stem-Branch of order 0 this is done by detection of the target layers’ chemical environment. As soon as the growth cone reaches a high enough concentration the growth cone is stopped. The higher order Stem-branches have a certain probability to stop growing. This probability is time-dependent which implies that the slower they grow the shorter they become.

Now remains the topic of how to decide when to branch in a Stem-branch of order higher than 0. This is again probability dependent. But it is also influenced by the concentration of the growth cones chemical which it is attracted to and the availability of the chemical ‘o’. Through these factors it is defined if a branch can currently spawn sub-branches or not.

To have more flexibility there are six parameters through which the model can be altered. These are not altering the model's general behavior but in the expression of growth properties it has. These factors are:

- **production speed of ‘o’**: This factor influences how fast the chemical ‘o’ is being produced in the soma. This chemical will be needed in the axonal tips by the growth cones to elongate the axon.

- **diffusion constant of ‘o’**: The diffusion constant decides about the viscosity of the internal chemical ‘o’ and how fast the material propagates throughout the cell.

- **consumption of a growth cone**: Defines how much of the chemical ‘o’ a growth cone consumes per length of elongation.

- **probability of a the main branch to branch**: This probability and the concentration of the target layers chemical define whether the main branch builds a sub-branch.

- **influence of ‘o’ on the sub-branch probability**: This defines how much the concentration of ‘o’ influences the probability of a sub-branch to branch.

- **probability of a sub-branch to die**: This probability and the concentration of the guidance chemical define how long an axonal branch can grow. This factor only influences Stem-Branches that have a higher order than 0.

- **probability of a sub-branch to branch**: This factor, the concentration of ‘o’ and the concentration of the chemical the axonal tip is guided to, decide whether a new branch and hence a growth cone is established or not.
The following figure of the growth model in action should help the reader to understand the model before we go into detail. This is an example of an innervated layer which had the chemical environment C (in blue) and the axons target layer A which is at the bottom of the figure.
Figure 3.11: The machine M1 secretes permanently in the soma, M2 initiates the outgrowth of the main-axon which will grow down to layer A (M3). The speed of the growth-cone will be limited by the substance availability of the microtubule substance ‘o’. While growing down the axon will encounter a chemical environment of layer C. The main-axon’s growth cone is sensitive to layer C and will prepare growth cones for side-branches (M4). These growth cones will grow in the direction of the chemicals C as soon as enough of microtubule ‘o’ is available. Also their growth speed will depend on the concentration of ‘o’ (M5). Furthermore these side-branches are capable of recursive side-branching, they also place growth cones while growing using the same machine as the first one (M4). This will lead to many growth cones growing at once and competing for the resource ‘o’. The growth cones will not grow forever and have a certain probability to stop growth (M6).
3.3 Axonal growth model

3.3.3 Fcoding the model

To give the reader a broader understanding how the model was implemented we explain now how the machines are designed in Fcode and what they are supposed to do. Additionally we relate it to the explanation of the model’s intention in the upper section 3.3.2. In the appendix B is a legend of what the different symbols mean, if it is not instantaneously clear to the reader.

Figure 3.12: **cellbase:** First there is the need to place some starting machine into the soma of a cell. This starting machine is the cell base machine which you can see in the figure 3.12. It basically does nothing more than to spawn two other machines, the secretion machine 3.13 and the branch starter machine 3.14. Both are explained further down.

Figure 3.13: **secrete:** The secretion machine is the producer part of the model. It is producing the chemical ‘o’ according to the defined speed in the model and according to the diffusion constant it is then diffused cell internally throughout the cell which can then be consumed (figure 3.17) by the growth cone.
3.3 Axonal growth model

Figure 3.14: **startbranch**: In order to let the axon grow we need an initiating outgrowth of a growth cone. This is established by this machine. As soon as it has spawned the growth cone, the machine is removed from the cell. The started machine in the outgrowing growth cone is the main axon machine 3.15 which corresponds to an order 0 Stem-Branch.

Figure 3.15: **mainaxon**: The main axon machine grows a Stem-branch of order 0, the root of all other Stem-branches to come. It basically consists of two sub-machines. First a growing machine 3.16 which grows where the concentration of A is highest. Second a side-branching machine 3.18 that decides on the concentration of C whether to place a growth cone at the current position or not. Furthermore it contains a mechanism to stop the growth of the Stem-branch when the layer with the chemical environment A is reached.
Figure 3.16: **grow** This machine does the work of the growth cone as we know it. It detects the gradient where the growth cone should head to and grows there, with the speed that is possible at the current concentration of 'o'. Furthermore on growing it consumes material. In order to know how much has to be consumed the move behavior is connected to the consume machine since a higher elongation per timestep should use more material than a shorter one. It also contains a time-dependent mechanism which tells an outer machine when it is time to stop the growth of this axon branch. The option *probability to die* can be altered on instantiation of this machine.

Figure 3.17: **consume**: The consumption machine is fairly simple as it just acts as a sink for the chemical 'o'. This is simulating the material needed for the growth. The consumption is influenced by the input, this input is connected to a move behavior which tells how much it has grown. This value is then taken in order to compute how much must have been consumed in the grow process of the move. On instantiation of this machine it can be defined how much is consumed per growth length.
3.3 Axonal growth model

Figure 3.18: **sidebrancher**: The side-brancher machine has a simple task, it places growth cones in the branch dependent on the defined probability to branch and on the chemical concentration of the chemical environment that the future side-branch has to innervate. As declared before the branches will wait until a high enough concentration of the chemical 'o' is reached. This waiting to grow is addressed by the machine kick start grow 3.19. This introduces already a competition-based model between the currently growing growth cones and the waiting ones. Being aware that multiple of the kick start grow machines are instantiated and all of these work according to the same rules and have the same parameters. Moreover all growth cones start perpendicular to the current branch.
Figure 3.19: **kickstartgrow**: The kick start grow machine waits for the chemical 'o' to exceed a certain threshold. When it is reached the grow and fork machine 3.20 which will fully grow the Stem-Branch. This Stem-Branch will innervate the layer it is spawned in.

Figure 3.20: **growandfork**: The grow and fork machine consists of two sub-machines. One is the grow machine 3.16 which will elongate the current branch in the direction of chemical gradient of the layer to be innervated. The other machine is a side-branching 3.21 machine again. This Side-branching machine places growth cones on the currently elongated Stem-Branch. These growth cones start to grow medially without a delay time. As it is easy to see this machine actually produces a Stem-Branch of a higher order.
3.4 Methodology: How to compare Neurons?

In order to compare the artificially grown neurons and the measured neurons we needed to measure them. Since the artificial and the natural neurons can be exported into the same format it is possible to apply these measurements to both. Axons build tree-like structures it therefore seems natural to apply measurements that are optimized to compare trees on our data.

Throughout science tree structures are very common, they exist in biology, computer science, geology and many more fields. Therefore there are lots of measurement techniques. Fractal measurements applied to tree structures seem to be the most common measurements for neurite growth. Many things in biology seem to be self-similar or statistically self-similar as Fernández claims [16] which corresponds to the fractal property. There are many measurements...
proposed by scientists for measuring neurons and their neuritic trees as fractals. In order to give the reader a brief overview we explain the most common measurements in the next sections.

### The Box-counting method/ Minkowski-dimension

The Box-counting method is widely used in neuroscience and fractal analysis [17] [4]. This method calculates the Minkowski- dimension [16] which is also called the Box-counting dimension.

![Box-counting method diagram](image)

Figure 3.22: To apply the box-counting method, a grid is laid over the neuron (in blue). In this grid all boxes are counted that are needed in order to completely cover the structure. The grid is made smaller, half the size and applied to the structure and the boxes are counted again (red). These numbers are put into a log-log plot with the axis log of box size in x and the log of box count in the y axis. The gradient of the resulting line represents the fractal dimension D [16]. This method is usually applied in 2d but can be extended to 3d.

### The coastline measurement

The coastline measurement is also known as the caliper method. It measures the coast length of the object at different resolutions [18] [19].
Figure 3.23: The coastline measurement takes a rulers of a fixed length. The first ruler is taken, one end is put on the border of the object of interest, then the ruler is tilted until the end is as well on the border of the object. A new ruler is added the one side adjacent to the end of the last one and tilted again till it reaches a point on the coast of the object. This is repeated until the complete object is wrapped in rulers. The count of the rulers needed is taken. The method is repeated with a ruler of smaller size and one can determine the fractal dimension D by adding the measurements to a plot where log of ruler size is the x axis and log of the ruler count is the y axis. This is a 2d method.

**The Hausdorff-dimension**

A very general measure for the complexity of an object is the Hausdorff-dimension [16]. It is a rather theoretical measure but related to the more practical box-counting method.
Figure 3.24: To determine the Hausdorff-dimension one covers the complete object with circles (blue) of diameter $d$. The circles are counted. One now investigates how quickly the count of circles grows when making this diameter smaller. In theory diameter $d$ goes to 0. The growth of circles on diameter reduction yields Hausdorff-dimension. This measure is calculated in 2d.

**The dilation method**

The dilation method is smoothing out the surface of the object of interest [20] [16] [17] [21].
3.4 Methodology: How to compare Neurons?

Figure 3.25: In the dilation method the border of the object is smoothed out. This is achieved using a convolution kernel applied to the object. Now this grayscale picture is taken and a binary decision is taken on each pixel. If it contains a gray color it is turned into black if not into white. One can now alter the radius of the convolution kernel and plot this in a diagram where the x axis is the log of the kernel diameter and the y axis is the log of the area covered divided by the diameter. This plot yields the fractal dimension by getting the gradient of the line [17]. This is a 2D method.

The mass radius

The mass radius Method [20] [17] [16], the Sand box method [20] [17] and the cumulative-mass method [19] [20] refer to the same method.
3.4 Methodology: How to compare Neurons?

Figure 3.26: In order to apply the mass radius, the center of gravity and the gyration radius (rotation radius) need to be computed (black circle). Having computed that, each pixel in the region of the circle is taken as origin for a circle of radius $r$ (blue circles). In each of these smaller circles the occupied pixels are counted. This is called the cluster mass. The cluster mass is averaged over all circles. The value is added to a plot with the x axis showing the log of radius of the circle and the y axis log of average cluster mass. This is repeated for smaller circles (red) in order to obtain multiple values in the graph. Calculating the gradient will yield the fractal dimension of the object\[17].

The Sholl-Analysis

Another widely used analysis method is the Sholl-Analysis. It was designed for 2d applications but can easily be extended to three dimensions. This method was developed in the context of neuroscience and is therefore especially designed for analysis of neurons \[19] \[16].
3.4 Methodology: How to compare Neurons?

Figure 3.27: In order to apply the Sholl-Analysis a circle around the center of the cell is drawn. In the Sholl-Analysis the center of the cell is defined as the center of the soma. After drawing the circle of radius $r$, the number of intersections with the cell are counted and recorded. The radius $r$ is altered and the procedure is applied again. These recordings are put into a log-log plot for radius versus intersection count where the slope of the resulting curve can be extracted and be used as a measure for comparison. [19]

The lacunarity

The lacunarity measure expresses the "gapiness" of an object but not exclusively it is also an expression for the non-uniformity of it [17]. There are many ways to calculate lacunarity we will focus on the one explained by Smith et al.
3.4 Methodology: How to compare Neurons?

Figure 3.28: The lacunarity describes the space filling property of an object. If the lacunarity is small the object fills a lot of space. To measure it, a box of size $r$ is defined. This box is then applied to the object and the mean number of mass/the pixel count is taken. Application in this sense means sliding the box over the picture such that all possible box positions are covered. The box is ten shrunk and the procedure is repeated. In order to contribute to the scaling, the mean is divided by the variance at each box size this yields the lacunarity value. These values are then added to a two dimensional graph expressing the relation of log radius to lacunarity value. [17]

There are many problems with these fractal measurements. First of all the fractal dimension that most of these methods compute is only somewhat descriptive [16]. Moreover the fractal dimension computed on biological systems tends not to be linear over more then two orders of magnitude[16]. The linearity is a description self-similarity which mathematical fractals fulfill. Another criteria for us was that the methods to compare axonal trees must be compatible for a tree dimensional problem. Any projections in a two dimensional space would destroy the properties of the grown neurons which would make the comparison even more unreliable. But most of the methods in literature are only made for two dimensional problems. Even though some have been redefined and used in three dimensions, the majority of these measures would not make sense in three dimension or simply takes up a vast amount of computing power to calculate. So Fernandéz, Smith, Jelinek and we come to a similar conclusion that these methods might not be very accurate for comparing neurite growth. [20] [16] [17]. There are also measurement methods that are not designed to measure fractal behavior, such as caulescence [22] used in botany, a measurement based on neurite length ANL (Average Neurite Length) [23] or asymmetry measurements such as NGA (Neurite Growth Asymmetry) [23], NISA (Neurite Initiation Site Asymmetry) [23]. A lot of the described measurements are already implemented and available for application in the tool L-Measure. [24] But not all of them make sense to apply to our neurons. We needed comparison methods that are applicable to three dimensional structures, that are shape oriented and that allow comparison from artificially grown neurons to the reconstructed ones. Moreover they should measure the key features we could identify like the observation of the Stem-Branches. As for the time limits of these thesis only a few measurements could be taken. We decided for historical reasons but also since the measurements seemed useful for comparison, to base our approaches mainly on our observation of the Stem-Branches and on the work of Binzegger et al. [4]. In the following two sections these comparison methods are explained in more detail.
3.4 Methodology: How to compare Neurons?

3.4.1 Existing approaches used

The following measurements have been used by Binzegger [4] to measure axons.

Depth:
The depth of an axon is defined by the longest path from the soma to any of the tips of the axon where each segment (from one branching-point to another) is counted as one unit.

![Diagram showing depth measurement](image)

Figure 3.29: The depth measures the numbers of steps from the root of the tree to reach the leaf that is furthest away from it [4] where each step is defined as going from one branching point to another.

Total length

The total length of a neuron is determined by measuring the length of all parts of an axon. This means summing up the distance along the axon from one branching point to another (where also tips and the axonal root are counted as branching points). [4]
3.4 Methodology: How to compare Neurons?

Figure 3.30: The total length of a tree is defined as the sum of the length of all its branches.

**Horton-Strahler number**

Is a measurement for complexity of trees originating in measuring the complexity of river basins. The Horton-Strahler number is defined as follows: an empty tree has the number 0.

\[
\text{if } S(T_L) \neq S(T_R) \text{ then } \max(S(T_L), S(T_R)) \\
\text{else } S(T_L) + 1
\]

where \( S(T_L) \) denotes the left subtree and \( S(T_R) \) denotes the right subtree of the current node.
3.4 Methodology: How to compare Neurons?

Figure 3.31: The Strahler-number is an indicator for the complexity of a tree. If we have a high complexity the Strahler-number is high as well. The Strahler-number is calculated bottom up, the leaves are assigned the value 1 each, then the leaves’ parents are calculated. If the right subtree and the left subtree have the same number then the number of the left subtree is taken and increased by one. If the two subtrees are not equal then the subtree with the maximal Strahler-number is taken. This number is the number of the current tree to be calculated. The number calculated for the root branch is the Strahler-number of the complete tree.

Galton-Watson

Galton-Watson is actually a model that describes the growth of a tree structure with the three probabilities

- \( P_{el} \) for probability to elongate a branch by 1 micron
- \( P_{br} \) for probability to branch
- \( P_{dt} \) for probability of the branch to die

We applied the model in reverse and calculated the probabilities that would lead to such a tree as we measure it.
3.5 Measurement application

Figure 3.32: The Galton-Watson measure used here is based on the Galton-Watson model. The model is based on the three probabilities, $P_{el}$ for probability to elongate a branch by 1 micron, $P_{br}$ for probability to branch, $P_{dt}$ for probability of the branch to die. For the measurement these values are computed. In order to do this the branchings are counted (number of branchings in red), the total length is measured (number of elongations in black) and the ending branches are counted (number of deaths in blue). These numbers are normed by the sum of length, branchings and endings. This yields the three probabilities.

These numbers already give some key figures about the shape of the axon.

3.4.2 Our approaches

The above mentioned measurements take into account the complete axon at once. We wanted to make more sophisticated measurements that are based on the Stem-Branch definition. The three measurements we came up with are defined as following:

- **Average length of the Stem-Branch orders $B_{len}$**: Is measuring the average length of the Stem-Branches on each order. This yields usually 7 numbers one for each of the orders 0 to 7.

- **Average sub-branches of the Stem-Branch orders $B_{sb}$**: Is measuring the average number of sub-branching (only side branching no bifurcation) of each Stem-Branch order. This yields also approximately 7 numbers for each of the orders one.

- **Average branching density of the Stem-Branch orders $B_{dens}$**: Is measuring the average branch density of each order. This yields usually 7 numbers one for each of the orders 0 to 7. where branch density is defined as $B_{dens} = B_{sb}/B_{len}$

3.5 Measurement application

We measured the reconstructed neurons in order to get reference values for our simulations. We will first state the results for the classical values from Binzegger [4]. Since Binzegger grouped the neurons somewhat different for different measurements we redid the measuring but only focus on the neurons of layer 2/3. We obtained the value for depth $19 \pm 3$, total length $65135 \pm 21379$, Horton-Straler number $6 \pm 1$ and the Galton-Watson probabilities $P_{el}: 0.9938 \pm 5.3050e-04$, $P_{br}: ...
3.6 Simulation setup

For simulating the growth of neurons we used the software cortex 3d which is based on Java. The computer on which the growth was simulated had 1.6 GHZ processor speed and 1 GB memory and was running openSUSE linux 10.3. Each of the grown axons had 15 minutes computing power in order to develop.

3.7 Results

An important result are the effects of changes of the parameter on the axonal growth. This shows best the capabilities of the model. We will go through each of these values and describe the effects of it. For each parameter we present three simulation tries, one with the parameter having

0.0031 ± 2.6525e−04 and \( P_{\text{rel}} \): 0.0031 ± 2.6938e−04. When we applied average length of the Stem-Branch orders, average sub-branches of the Stem-Branch orders and average branching density of the Stem-Branch orders we got the results as shown in the graph below.

These measurements have been taken as a reference for the artificially grown neurons. The focus on the matching of the values for \( B_{\text{len}}, B_{\text{sb}} \) and \( B_{\text{dns}} \). Our intention was to match the values in the graph from the reconstructed neurons with the artificial ones.

Though before we did that we tried to explore the parameter space for the model to get a better impression on which parameter has what influence on the outcome of the growth.
a low value, a moderate value and a high value. We keep the values for the other parameters constant during these changes. Our standard values are stated in the table 3.1.

<table>
<thead>
<tr>
<th>parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>production speed of 'o'</td>
<td>0.3</td>
</tr>
<tr>
<td>diffusion constant of 'o'</td>
<td>10000.0</td>
</tr>
<tr>
<td>consumption of a growth cone</td>
<td>0.01</td>
</tr>
<tr>
<td>probability of a the main branch to branch</td>
<td>0.0367</td>
</tr>
<tr>
<td>probability of a sub-branch to die</td>
<td>0.006</td>
</tr>
<tr>
<td>probability of a sub-branch to branch</td>
<td>0.054</td>
</tr>
</tbody>
</table>

Table 3.1: Standard values for the exploration of the models parameter space

The intention is to give the reader an overview over the parameter space of the model.

After having explored the parameter space we present an optimized value for the axons in order to match the results we have from measuring the reconstructed axonal trees.

### 3.7.1 Exploration of production speed of 'o' space

We changed the value of production speed of 'o' from 0.009 over 0.609 to 5.409. The other parameters will be kept constant as stated in the table 3.1. The next figures show the measurements of section 3.4 applied to the grown neuron. Moreover the shape of the neuron is shown in a two-dimensional projection to give the user a visual impression of the looks of the neuron.
3.7 Results

(a) Length

(b) Branches

(c) Branch density

(d) Neuron id: 1231887505047 Value: 0.009 (green curve)

(e) Neuron id: 1231888409629 Value: 0.609 (red curve)

(f) Neuron id: 1231896303532 Value: 5.409 (black curve)

Figure 3.34: Altering the production speed of 'o'
3.7 Results

Figure 3.35: The measures described in section 3.4.1 applied to the three grown neurons.

The production speed of ‘o’ defines how fast the growth cones and how many growth cones can elongate the axon at once. If you use high values it floods the cell with ‘o’ and drives the growth cones at a very high speed which yields long and a very developed axonal tree. Furthermore the axon gets more complex in structure because of the many more growth cones numerous side branches can be grown which also leads to a higher Strahler-number. If you go to very high values, the branches will grow at a higher speed and can therefore get longer. The branching also increases, but both elongation and branching seems to scale linearly which leads to a moderate branching density.

3.7.2 Exploration of diffusion constant of ‘o’ space

We changed the value of the diffusion constant of ‘o’ to the values 300, 20300 and 180300. Simulations with these values raised the following neurons.
3.7 Results

(a) Length
(b) Branches
(c) Branch density

(d) Value: 300 (green curve)
(e) Value: 20300 (red curve)
(f) Value: 180300 (black curve)

Figure 3.36: Altering the diffusion constant of ‘o’
Figure 3.37: The measures described in section 3.4.1 applied to the three grown neurons.

As you can observe, if the diffusion constant is too low the axon will hardly grow. As a result of the low diffusion constant the viscosity of the chemical ‘o’ is high. This effect causes that the material is hardly distributed in the cell which means very few of ‘o’ reaches the growth cones. The growth cone can not elongate, it misses the materials needed. Hence the axonal length is low. There is furthermore no side-branching happening, because the threshold that allows the outgrowth of the growth cones of the first order Stem-Branches is not reached during simulation time. This fact is responsible that the Horton-Strahler will stay very low and that the branching probability stays 0. If you go higher with the diffusion constant of ‘o’ the axon will grow faster and also side-branches more, as one can easily observe in the figure 3.36e. The measured values go up again to higher values and also the complexity of the axon increases. The axon starts to behave more as one expects of an axon. If one goes even higher the effect diminishes since the computational border is reached. The chemical can not distribute instantaneous and the growth cones will not go faster. The diffusion constant of ‘o’ basically determines the reaction-speed, how fast the growth cones can react on the produced chemical ‘o’ in the soma.

3.7.3 Exploration of consumption of a growth cone space

The model allows the modification of the growth cones consumption per length it grows. This effect is demonstrated in the following figures. The values taken are $3.0 \times 10^{-4}$, 0.0203 and 0.1803.
3.7 Results

(a) Length

(b) Branches

(c) Branch density

(d) Value: $3.0E-4$ (green curve)

(e) Value: 0.0203 (red curve)

(f) Value: 0.1803 (black curve)

Figure 3.38: Altering the consumption of a growth cone
These results clearly show what happens if you take out the competition aspect of this model. This is what happens if you reduce the consumption of the growth cones to a low value. They will not use up the produced material and therefore even the higher order branches can grow very long, since supply of ‘o’ never ceases. The total length of the axon increases, the Galton-Watson numbers show a stronger elongation factor and the depth is high since there is still a lot of side-branching happening. If one goes up with the consumption of a growth cone value, it can be observed that the branch-length is again behaving in a competitive manner and that the Stem-Branches of higher order get shorter than the ones of lower order. The complexity of the neuron is increasing and the probability to die increases. If one goes to even higher values, one can observe that all the Stem-Branch orders get shorter. Especially order 1 and 2 are affected by this which is caused by the increased competition. The competition gets a stronger limiting factor as it has been with lower values of growth cone consumption. This shortage of branches also affect the length, the depth and the Strahler-number as all these values get smaller.

### 3.7.4 Exploration of probability of a the main branch to branch space

The parameter probability of the main branch to branch was explored using the values 0.001101, 0.037801 and 0.6617. This should give the reader an insight what happens on alternation of this parameter.

**Figure 3.39:** The measures described in section 3.4.1 applied to the three grown neurons.
3.7 Results

(a) Length

(b) Branch density

(c) Branches

(d) Value: 0.001101 (green curve)

(e) Value: 0.037801 (red curve)

(f) Value: 0.60617 (black curve)

Figure 3.40: Altering the probability of the main branch to branch
3.7 Results

Figure 3.41: The measures described in section 3.4.1 applied to the three grown neurons.

This factor influences the first order branches the most. If the probability of the main branch to branch is kept very low, there will hardly be any first order branches or none at all which then leads to a branchless axon that is just going to its target layer without branching. This has an influence on all measured factors, all values stay very low. If you increase the probability of the main branch to branch, there will be branching and a more complex structure will arise which leads all measured values to a moderate level. One expects higher values to give even more branches of the first order, this effect though can not be seen in the data. The reason for this is, that the axon has only 15 minutes to grow and can therefore not fully develop all the axonal tips. The growth cones are placed in preparation, but never have access to enough of chemical ‘o’ in order to start their growth. If we let the axon grow for longer we would expect the that more Stem-Branches of order one will appear because their prepared growth cones will eventually reach the threshold for outgrowth as soon as the competition of the other branches diminishes somewhat. This will happen as soon as some of the active growth cones at minute 15 stop and therefore are taken out of the competition. Then eventually the threshold will be reachable. Hence first order Stem-Branches start to grow more.
3.7.5 Exploration of probability of a sub-branch to die space

We changed the value of probability of a sub-branch to die from $1.8e - 4$ over 0.00618 to 0.10818. The other parameters will be kept constant as stated in the table 3.1. The next figures show the measurements of section 3.4 applied to the grown neuron. Moreover the shape of the neuron is shown in a two-dimensional projection to give the user a visual impression of the looks of the neuron.
3.7 Results

(a) Length

(b) Branches

(c) Branch density

(d) Value: $1.8 \times 10^{-4}$ (green curve)

(e) Value: $0.00618$ (red curve)

(f) Value: $0.10818$ (black curve)

Figure 3.42: Altering the probability of a sub-branch to die
If the probability of a sub-branch to die is low, the growth cones will grow over a longer time which leads to longer Stem-Branches in all orders. The growth rate is highly increased, there is less concentration of ‘o’ in the axon. Therefore the branching process is slightly decreased per length of the axon which leads to a lower density of branching. Moreover the axon becomes very long, though the complexity of branching is not very high, the Strahler-number stays at a moderate value and therefore also the depth. As you go up with the probability the axonal branches get increasingly shorter with order. A diminishing effect on length can be seen at higher order Stem-Branches. The density of branching goes up and the branching gets more complex and therefore the Stahler-number and depth go up. But the total length of the axon is reduced compared to the case with a small probability. By setting the probability of a sub-branch to die to a very high value, newly spawned branches almost instantaneously die and can not branch further. Therefore the neurons hardly builds any Stem-Branches higher than order two which leads to a diminished complexity again and a shorter axon in total, with less depth and a smaller Stahler-number.

3.7.6 Exploration of probability of a sub-branch to branch space

To explore the space of the parameter probability of a sub-branch we apply the three values $0.00162$, $0.10962$ and $1.02762$. The results we optioned are stated on the following pages.
3.7 Results

(a) Length

(b) Branches

(c) Branch density

(d) Value: $0.00162$ (green curve)

(e) Value: $0.10962$ (red curve)

(f) Value: $1.02762$ (black curve)

Figure 3.44: Altering the probability of a sub-branch to branch
3.7 Results

Figure 3.45: The measures described in section 3.4.1 applied to the three grown neurons.

As expected, all the three figures show that the first order Stem-Branches grow out, since this probability is not affected here, the effect should show itself from the second order branches on up to the highest order ones. As one can observe in the figure 3.44d, if the value is small there will grow very few Stem-Branches of order 2 or higher. These branches did not have any branches of their own caused by the propagated small probability of the sub-branches to branch. This also has an effect on the total length of the axon, the depth and the Strahler-number as all three values stay very low.

As soon as one goes up with the branching probability, Stem-Branches up to order 6 appear. It is to note that we observe that through the increasing competition at higher order branching, the branches start to get shorter. The effect, that there is less branching in higher order Stem-Branches, is due to two reasons. First of all the branches get shorter, therefore there is less space to branch and second the branch density decreases in higher order branches. The decrease in branch density with higher orders is expected since the probability to branch is paired with the availability of ‘o’ at the evaluation point of branching. Due to increased competition in higher Stem-Branch orders the concentration of ‘o’ is lower at further stages of development. In general the further you go away from the soma the lower the concentration of ‘o’ is. This is caused by the diffusion and that chemical ‘o’ is used on its way to the far ends of the axon.

If you go even higher with the probability to branch, the branch density massively increases. Obviously there are more branches generated and therefore the values of average sub-branching of the Stem-Branch order increases. This has the effect of increased competition, since many more growth cones spawn in a shorter area of space. This increase in competition of the chemical ‘o’ leads to the effect that the branches get shorter.

In general one can say that the measures of depth, total length and Strahler-number increase with the increase of the probability of a sub-branch to branch.
Note that this parameter allows us to grow more compact or widespread neurons. This effect can be deduced by looking at the shape of the axonal growth in our figures 3.42. This variation of neurons can be observed to a certain extent in nature as Caserta et al. [19] mention.

### 3.7.7 Best matching neurons to reconstructed neurons

After having explored the parameter-space of our model, we present the best matching results with the parameters **production speed of 'o'** set to 0.45, **diffusion constant of 'o'** set to 10000.0, **consumption of a growth cone** set to 0.0075, **probability of a the main branch to branch** set to 0.0367, **probability of a sub-branch to die** set to 0.0060 and **probability of a sub-branch to branch** set to 0.054. We found these measurements by doing a manual gradient decent on the values. In order to get a feeling for the parameters. By using these values for our parameters we obtain the following measurement graph.

![Graphs showing measurements of section 3.4.2 for the best matching neuron](image)

**Figure 3.46: Measurements of section 3.4.2 for the best matching neuron**

As one can observe there is this huge difference in branch density in the order 6. This could be due to the fact, that there have not been many other order 6 branches spawned. Since the neuron has only 15 minutes to grow artificially it is not yet developed completely and the effect we see in this order could be due to that. We expect the effect to average out with longer development times. Though the curves observed in natural neurons are not exactly followed by the artificially grown one, the tendency of decay can be observed as it is expected due to competition. Generally it can be said that the branches of each order are slightly too long, a reason for that might be the lack of retraction in our model. Furthermore there seem to be to few of Stem-Branches order 1, this is again in relation to the growth time, the neuron has not had time yet to fully grow, there are still growth cones that have not yet started to innervate the layer and therefore its numbers will adjust with further growth time.

To give the reader a better impression what the neuron looks like when fully grown, we show the shape of it, projected in the x-y space:
3.7 Results

Figure 3.47: Shape of the best matching neuron

As expected the neuron innervates the layer as it should. It spreads out nicely in the layer and therefore generates a lot of surface which could be used for other neurons to synapse with it. These properties can be observed in the reconstructed neurons as well. It is not yet fully developed and has therefore some more side-branches to come. On the other hand the branches bend unnaturally and create a palm tree shape which is not the case for the reconstructed neurons. But this has not yet been part of our investigations.

Furthermore we applied the measurement methods of Binzegger [4] which yields the following results:

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>15.0</td>
</tr>
<tr>
<td>Total length</td>
<td>32299</td>
</tr>
<tr>
<td>Horton-Straler number</td>
<td>5</td>
</tr>
<tr>
<td>Galton-Watson $P_{st}$</td>
<td>0.99544</td>
</tr>
<tr>
<td>Galton-Watson $P_{br}$</td>
<td>0.00222</td>
</tr>
<tr>
<td>Galton-Watson $P_{dt}$</td>
<td>0.00234</td>
</tr>
</tbody>
</table>

Table 3.2: Measurement values for the best matching grown neuron

One can observe that the depth is not as high as in the reconstructed neurons. We guess that for most of the artificially grown or real grown neurons the main axon has the most segmentation and therefore decides about the depth of the neuron. Therefore if we would let the neuron grow for a longer time it would start to get a higher number. The Strahler-number is in the boundaries we observed in natural neurons and so are the Galton-Watson numbers. They seem to follow them nicely. Since there are many branches of order 1 missing due to the growth time, it is only logical that the overall length is too short which explains why it is only about half the length of the reconstructed neurons. We would expect for longer growth times that this number goes up drastically and that the overall length for our simulated neurons will be higher than the natural ones. This relates again to the fact that most Stem-Branches are a bit too long. As we stated
we think this might relate to the missing retraction or to the missing influences of a more natural environment.

### 3.7.8 Further observations

We first would like to point out that our model complies with the results of Fenstermaker [25]. This applies that if you leave away the environmental chemical of layer 2/3 you get the following shape of the axon:

![Figure 3.48: The effect of neglecting the chemical guidance gradient](image)

As one can easily predict and observe, there was no side-branching happening. Even though the model was not designed to match the results of Fenstermaker [25] the growth seems to comply to the naturally observed behavior. If there is no guidance chemical available, the axon will not branch.

Moreover another interesting observation is that the branching happens more often closer to the soma. This means that the observed terminal branches are longer than the ones closer to the center (terminal branches not Stem-Branches) which also complies with the measurements and the observations made by Van Pelt et al. [26]. This also arises naturally from the model and is not explicitly designed.

### 3.8 Criticism on the model

With the developed model it seems to be possible to grow artificial neurons in a physical environment as given by Cortex 3d. The model tries to be biologically plausible though there are some points that are not satisfactorily fulfilled. One critic on this model is that it totally neglects bifurcation. In a further development step this has to be added. Another point is that the bending of the axonal branches seems somewhat unnatural, there has to be investigation about this issue to make this look more natural. All the neurons have been simulated in a static environment with artificial chemical gradients respectively layers that where exactly horizontal and had a Gaussian distribution in the vertical direction. Moreover there where no obstacles around for the growing branches to dodge, neither other cells nor other branches. We only designed the neurons to innervate layer 2/3 and not the layer 4 also. The innervation of layer 4 is clearly observable in the reconstructed neurons and therefore the model does not comply
with the reconstructed ones in this point.

This model only grows axons and neglects the dendrites totally, but it is very likely that these also contribute to the competitive behavior of the growth cones. Furthermore, retraction has not been considered nor used. Electrophysiology and its influences on the growth and its influences on the operational changes of the morphology have not been considered in the model.

Nevertheless there are many points in favor of the developed model. One of the main advantages of this model is that through using Fcode, further developed models for other aspects of the growth can very easily be combined with this one. This will yield even more powerful models in the future. Moreover this model is able to produce neurons that look like natural axons and complies with the measurements from the reconstructed neurons of cat visual cortex. It yields more sophisticated results than other models known from literature. The model integrates many ideas that have been suggested in literature. It also shows behaviors that are naturally arising from the model and that can also be observed in naturally grown neurons. As we expected the model is far from perfect, it is at this stage not capable of growing neurons that look exactly like reconstructed neurons. But the model seems to approach the problem of axonal growth from the right direction. This fact makes our model an interesting approach to neural growth.

### 3.9 Where to head now?

The question remains now, what has to be done in order to improve the stated model. We are going to keep the Fcode language as modeling language since it turned out to be very powerful and modular.

The model has to be tested in an environment which is more naturally inspired. Therefore multiple cells should be grown at once in a compound and the layers should arise naturally from these cells which includes a working gene regulation, cell specification and migration of cells. On the morphological point of view, the model is not complete in the sense that it is only innervating one layer instead of two, in a further development step this has to be changed. But this step is straightforward in adding another ‘sidebrancher’ machine to the main axon. The tricky part will be the finding of the right amount of innervation which requires to explore the additional parameters of the second ‘sidebrancher’ machine. Another open point is the missing electrophysiology. We need a working electrophysiology during the growth since these factors have shown to be important for the further development of the morphologic properties of a neuron [27]. Moreover it contributes to the main goal of growing functional models of the cortex. Furthermore retraction seems an important mechanism that will need to be investigated by future models.

To address the appearance of the neurons it could be interesting to have a look at the curvature of the growing branches to show the relation between Stem-Branch order and curvature. This could also lead to more natural looking neurite branches. There might also be a need for a model that considers multiple chemicals to steer the development of the axonal tree. Furthermore a mathematical description of the growth interactions of our model might be helpful to better judge the consequences that changes in the model have on the axonal growth which means it would make sense to deduce the mathematics from our model. Investigations have to be done in these directions.

And there will be a need to show that our crude approximations of biological processes will hold at the microbiological level. This stays one of the challenges to be tackled in the future.
Chapter 4

Fcode Behavior Designer

In this chapter we describe a tool that assists the modeler in creating Fmachines. The Behavior Designer shall be a practical tool where you can design Fcode machines and complete XML-Genomes. The Behavior Designer should allow the user to drag and drop minimal behaviors into a machine and connects the behaviors to each other. It should also be possible to alter values of the parameters of the minimal behaviors. The user should be able to save his genome into a file and reload genomes back into the Designer to further refine them. Moreover it shall be possible to simulate the machine directly from the tool.

4.1 Fmachine composer

As mentioned in the task 1.3 section. The need arose to create a tool where you can visually develop in the Fcode language. This implied the creation of a GUI. In order to make such a GUI, a design has been worked out that should allow the manipulation stated as follows:

The idea was that the user is given a sheet where he/she can compose Fmachines. These Fmachines are registrable as part of the XML-Genome. The composing is really easy which means using drag and drop to get minimal behaviors in place and clicking on ports to connect those with each other. Moreover it is possible to edit the properties of each minimal behavior by double clicking on the visual representation of it. This property window displays all the options the user has to alter the parameters of the minimal behavior (figure 4.2). Moreover the user is able to put already created Fmachines into the currently edited Fmachine. He/She can run a simulation of the currently created machine. To give the reader a visual impression of the tool, screenshots have been added (figure 4.1).

As a code paradigm the pattern code behind has been taken in order to code the GUI which means that the GUI is sending all functional related requests to a back end class that will then alter the to be composed Fmachines in the way the user intends to. Moreover it has been decided that visual representation of the minimal behaviors are actual wrappers and during the process of creation of the machine the functional minimal behavior classes are already created and linked to each other. The user is able to register a developed machine in the genome class 2.3 which will collect all the different machines composed. On execution of the save function the genome class will simply serialize all the Fmachines to a user given file. By using the simulation command the Fmachine is automatically registered in the genome and the genome class then instantiates a new Fmachine of the given type. This Fmachine is then loaded into a single virtual cell and executed during the simulation (figure 5.3).
Figure 4.1: Screenshot of the Behavior Designer Tool
4.1 Fmachine composer

Figure 4.2: Screenshot of the property editor

Figure 4.3: Screenshot of a machine simulated in a cell
Chapter 5

Cortex Designer

In this chapter we are going to present the Cortex designer. The Cortex Designer is a prototype that allows the user in a fairly simple manner to describe how his/her cortex has to look like in the end. Therefore there should be a simple way of defining the cortex. It should be graphical and intuitive. It should be possible to add layers, put cells into layers and add connections from one layer to another. Moreover in another partial project that is done by Sabina Pfister it shall be possible to design the lineage of the cells. When a user has designed his/her cortex he/she shall be able to simulate it. The tool should construct the necessary XML-Genome. This XML-Genome should be loaded into a stem cell which defines the starting point of the simulation in CX3d. This cell should then with the help of a Gene Regulatory Network create all the cells needed in the order that was defined by the lineage tool. These cells should then build layers and innervate the layers defined in the Cortex Designer. For all of this just one genome file should be created and the user should not need to do any more definitions than he/she has done in the Cortex Designer.

5.1 Building the Cortex Designer

The Cortex Designer arose from the wish that it shall be possible for a user to give minimal instruction and the Cortex Designer shall then be able to use that information to automatically grow a cortex. As stated earlier this is a prototype which will need refinement especially in the direction of growing a cortex in a biologically more plausible way.

The Cortex Designer needed to be designed that the user could define the shape of the cortex in a very abstract way. The user is given a sheet which enables him/her to drag and drop elements on the cortex. It turns out to be useful in this prototype to give the user three possibilities.

- Drop a layer on the cortex which should be created by secretion.
- Add a cell type to a layer where those cells should migrate to
- Add a patch to a layer which an axon should innervate.

The user can connect the cells to patches in order to tell them which layer has to be innervated by what cell type. Moreover the cells, patches and layers have option windows (Figure 5.2) much like the ones described in the section 4.1 where the user can define the properties of the items.

As for the cell it is possible to define which color (for convenience reasons) and the amount to be spawned. For the patch it is possible to set the size of it which means how intense the layer will be innervated by the given axon. For the layer can be defined what color it has and what chemical it consists of. All this is savable to a file in order to reload certain designs. Furthermore it should be possible to connect the to be created lineage tool of Sabina Pfister to the Cortex Designer. This lineage tool will take care of the definition how the Gene Regulatory Network is built in order to spawn cells. It moreover defines how the stem cells change to the defined cell types and these cells will then populate the defined layers. For the time of this thesis this feature was not implemented and therefore the cells are supposed to be spawned instantaneous on the very start of a cortex growth simulation. Also the layers are placed by artificial means and
give a Gaussian shape of density at the position described in the Cortex Designer. This layer construction should be later replaced by a more natural way of development and there shall be cells used to secrete chemicals. These cells shall then form the layer. As the very core of the Cortex Designer, it should be possible to simulate the described cortex. In terms of development this means an interpreter had to be created that takes the users description, composes the description into Fmachines and then places it into the initial cells. These Fmachines must take into account the connection of the cell to different layers, the color of the cell and the layer the cell shall migrate to. This assembling of the description is based on predefined machines that are modified slightly and composed together during the translation task. Then the genome is loaded into the stem cell and the simulation is being executed. As for now without the Lineage Designer, the Gene Regulatory Network is not available. In this version the stem cell actually is multiple cells that have already a defined cell type, those cells will be spawned in the user defined numbers directly in the simulation environment and are imprinted with the cell type specific Fmachine to define their behavior.

Figure 5.1: Screenshot of the Cortex Designer
Figure 5.2: Screenshot of the property editor of a the Cortex Designer, properties of a cell are shown

Figure 5.3: Screenshot of a simple simulated cortex part
5.2 Taking it to the next level

Especially this part of the thesis is supposed to be extended since it targets at self-construction respectively organisation. The Cortex Designer misses many wanted features though. First of all there should be a Lineage Designer where the cortex creator can define the lineage of the cells or even an automatic Lineage Designer which matches best the desired setup of the cortex. The patches are furthermore not at all natural looking, they do not seem to spread into the right direction and do not target any other cells. Furthermore the patches in pass through layers should not be innervated by a second axon but rather have the same axon that passes trough the layer innervating it. At the current state each layer is created by a separate axon. Moreover dendrites are completely missing. The electrophysiology and hence the synapsing between neurons need also to be integrated in order to have a functioning self-constructing cortex. The automatic construction of layers is another important point, how to create the guidance queue that the axons need to grow and how to integrate the influence of glial cells on the process. Lastly all the growth should be described in a way that is biologically plausible which is not the case yet. This should eventually lead to self-constructing functioning neural-circuits that have functions as observed in the real cortex which can then be used to do computations with it and analyze their behavior under varying conditions and inputs.
Chapter 6

Summary

In this master thesis we present an axonal growth model. Throughout the literature many neurite growth models have been suggested [11] [12] [13] [14]. These models are not designed or are not capable of reproducing axonal-like shapes. Models often lack a biological explanation such as the purely probabilistic approaches of Van Pelt [11] They do not take into account chemical environment and are therefore not able to react to their surroundings. None of the models are designed to be used as a model for simulation and growth of artificial neurons. Most of them are created for a two-dimensional space and are not three-dimensional. All of them are not established to act in a physical environment. Another important aspect is the branching process. We found not even one model using side-branching, all are based purely on bifurcation, even though we observed more side-branching in reconstructed axonal trees than bifurcation. Side-branching seems generally to be neglected. Our model bases on side-branching rather than bifurcation which we observed to happen more frequently in the neurons we analyzed which also lead to the notion of Stem-Branches. Stem-Branches allowed us to view the branching process from a different point then the common branching notion. The model is context sensitive to its chemical environment, for instance it can follow external chemical guidance cues. It is entirely based on the biological programming language Fcode developed by Zubler et al. [2] which allows a reuse and recombination of partial aspects of neurite growth. Our approach also integrates the computational aproach of Van Ooyen [13] and uses intracellular diffusion to transport raw materials for growth from the soma to the axonal tips where the growth cones lie. This is where the probabilistic approach of Van Pelt et al. [11] inspired us. Dependent on these chemicals and on a base probability to branch, to die and to elongate the growth of the axon is steered. And lastly the complete model is simulated in a physical plausible environment that is given by the application CX3d developed by Zubler et al. [1]. It can easily be seen that for the growth to be simulated we needed a model for growth cones. Where more elaborate models of growth cones go into depth, we stayed superficial for simplicity and computational efficiency reasons [6] [7]. We achieved this with a fast and simple but still biologically inspired model which we present in this thesis.

We wanted to compare neurons grown with this model to reconstructed neurons from cat visual cortex. Therefore measurements for comparison were needed. Many of the measurement methods suggested by Binzegger et al. [4] were taken for this task. Other comparison methods were not as convincing as the ones eventually used. Moreover we took the idea of the side-branching further and developed methods to measure branch lengths, counted the average branching of Stem-Branches and measured the density of side-branches. These methods enabled us to compare the artificially grown neurons to the reconstructed ones. Since the model has parameters that have influence on its behavior and on the shape of the grown neurons, we explore the parameter space of the model and show the reader interesting properties of these parameters. We present a grown neuron that matches the reconstructed neurons best and point out that our model has unintended emerging properties that matches behavior observed in natural neurite growth. As the initial main goal of the thesis, it was purposed that an XML representation of Fcode is implemented. This goal was reached and is shortly discussed here. But it is intentionally not the main part of the thesis. Also the second goal of integrating the Gene Regulatory Network for creating a cell lineage has
been added into the XML description. The complete description we call XML-Genome. Such genome-files can be loaded into an artificial cell of CX3d. The cell will then behave like a stem cell and initiate the cell lineage and the growth that has been defined using the Fcode language.

Out of this initial task the idea arised to test how far you can get with Fcode in describing neural growth model and how fast you can develop one. The results described in this thesis is what has been achieved in this terms.

Furthermore a development tool has been created throughout the thesis to faster develop growth models in Fcode such as the Fmachine designer which is a tool for creating Fcode in a visual way where you can drag and drop in Fcode and link the code axioms together to create more complex machines.

As an extension to the model and Fmachine designer a Cortex Designer has been prototyped. This tool lets the user drag and drop a graphical description of the cortex, including cells and axonal wiring. This is held in a very abstract way and will then be translated into Fcode which lets eventually a cortex grow according to the user given description. This is done in a self-organizing way, including migration of cells to layers where they belong, axonal growth to defined layers and patch building of axons in these layers.
Appendix A

XML

A.1 XML example File

This is the XML description of the example mentioned in section 2.7 (Note that if the chemical value is changed from the value E to another value, the Fmachine will search for that chemical.)

```
<genome>
  <machine name="start">
    <detect name="detect0" chemical="E" location="extra_celular" />
    <fork name="fork0">
      <machine name="bifork"/>
      <machine name=""/>
    </fork>
    <positive name="positive0"/>
    <kill name="kill0" reference="this" />
  </machine>
  <machine name="bifork">
    <kill name="kill1" reference="this" />
    <move name="move1"/>
  </machine>

  <vectoriallink tag="detect0.gradient_fork0.direction">
    <from machine="detect0" output="gradient" />
    <to machine="fork0" input="direction" />
  </vectoriallink>

  <link tag="fork0.hasForked_kill0.probabilityToKill">
    <from machine="fork0" output="hasForked" />
    <to machine="kill0" input="probabilityToKill" />
  </link>

  <link tag="positive0.output_fork0.probabilityToFork">
    <from machine="positive0" output="output" />
    <to machine="fork0" input="probabilityToFork" />
  </link>
</genome>
```
A.2 The XML Schema

This is the XML-Schema which defines the XML-Genome:

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:documentation>
      This is a document that describes
      the fCode machine language in the cx3D project
    </xs:documentation>
  </xs:annotation>

  <!-- promoter reader starts here -->
  <xs:complexType name="evaluatable"/>
  <xs:complexType name="booldecision">
    <xs:complexContent>
      <xs:extension base="evaluatable">
        <xs:attribute name="substance" type="xs:string"/>
        <xs:attribute name="threshhold" type="xs:decimal"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

```xml
<detect name="detect1" chemical="E" location="extra_cellular" />
<fork name="fork1">
  <machine name="bifork"/>
  <machine name="bifork"/>
</fork>

<!-- Links -->

<link tag="detect1.concentration_fork1.probabilityToFork">
  <from machine="detect1" output="concentration" />
  <to machine="fork1" input="probabilityToFork" />
</link>

<link tag="fork1.hasForked_kill1.probabilityToKill">
  <from machine="fork1" output="hasForked" />
  <to machine="kill1" input="probabilityToKill" />
</link>

<vectoriallink tag="detect1.goverc_move1.direction">
  <from machine="detect1" output="goverc" />
  <to machine="move1" input="direction" />
</vectoriallink>
</machine>
<mainmachine>null</mainmachine>
</genome>

A.2 The XML Schema

This is the XML-Schema which defines the XML-Genome:
<xs:element name="and" type="promoterand"/>
<xs:element name="or" type="promoteror"/>
<xs:element name="lessthen" type="booldecision"/>
<xs:element name="greaterthen" type="booldecision"/>
</xs:choice>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="promoterand">
<xs:complexContent>
<xs:extension base="evaluatable">
<xs:choice maxOccurs="unbonded" minOccurs="1">
<xs:element name="and" type="promoterand"/>
<xs:element name="or" type="promoteror"/>
<xs:element name="lessthen" type="booldecision"/>
<xs:element name="greaterthen" type="booldecision"/>
</xs:choice>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="argument">
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="value" type="xs:decimal"/>
</xs:complexType>
<xs:complexType name="promoter">
<xs:sequence>
<xs:element name="argument" type="argument" maxOccurs="unbounded" minOccurs="0"/>
<xs:choice maxOccurs="unbonded" minOccurs="1">
<xs:element name="and" type="promoterand"/>
<xs:element name="or" type="promoteror"/>
<xs:element name="lessthen" type="booldecision"/>
<xs:element name="greaterequalsthen" type="booldecision"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
<!−− Gene regulatory network definition starts here−−>
<xs:complexType name="genesubstance">
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="color" type="xs:decimal"/>
</xs:complexType>
<xs:complexType name="influencingfactor">
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="value" type="xs:decimal"/>
</xs:complexType>
<xs:complexType name="gene">
<xs:sequence>
<xs:element name="genesubstance" type="genesubstance" maxOccurs="1" minOccurs="1"/>
<xs:element name="influencingfactor" type="influencingfactor" maxOccurs="unbounded" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="badalproduction" type="xs:decimal"/>
<xs:attribute name="name" type="xs:string"/>
</xs:complexType>
<!−− The definitions for the machines start here−−>
<xs:simpleType name="position">
<xs:restriction base="xs:string">
  <xs:enumeration value="membrane"/>
  <xs:enumeration value="inner"/>
  <xs:enumeration value="outer"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="support">
  <xs:restriction base="xs:string">
    <xs:enumeration value="inner"/>
    <xs:enumeration value="outer"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="input">
  <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
<xs:complexType name="output">
  <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
<xs:complexType name="vectorialinput">
  <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
<xs:complexType name="vectorialoutput">
  <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
<xs:complexType name="link">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="from">
      <xs:complexType>
        <xs:attribute name="machine"/>
        <xs:attribute name="output"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="to">
      <xs:complexType>
        <xs:attribute name="machine"/>
        <xs:attribute name="input"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="weight"/>
  <xs:attribute name="tag"/>
</xs:complexType>
<xs:complexType name="vectoriallink">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="from">
      <xs:complexType>
        <xs:attribute name="machine"/>
        <xs:attribute name="output"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="to">
      <xs:complexType>
        <xs:attribute name="machine"/>
        <xs:attribute name="input"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="weight"/>
<xs:attribute name="tag"/>
</xs:complexType>
</xs:complexType>
<xs:complexType name="behaviouralElement" abstract="true">
  <xs:attribute name="name"/>
</xs:complexType>
</xs:complexType>
<xs:complexType name="machineelement" abstract="true">
  <xs:attribute name="name"/>
</xs:complexType>
</xs:complexType>
<xs:complexType name="behaviouralElement" abstract="true">
  <xs:attribute name="name"/>
</xs:complexType>
</xs:complexType>
<xs:complexType name="machineelement">
</xs:complexType>
<xs:complexType name="machineelement">
</xs:complexType>
<xs:complexType name="behaviouralElement">
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>
    <xs:attribute name="machine2" type="xs:string"/>
  </xs:extension>
</xs:complexType>
<xs:complexType name="fork">
  <xs:extension base="machineelement">
    <xs:attribute name="machine1" type="xs:string"/>...
```xml
<xs:extension base="behaviouralElement">
  <xs:attribute name="chemical" type="xs:string"/>
  <xs:attribute name="support" type="support"/>
</xs:extension>
</xs:complexContent>
<xs:complexType name="synapse">
  <xs:complexContent>
    <xs:extension base="behaviouralElement"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="replicate">
  <xs:complexContent>
    <xs:extension base="behaviouralElement"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="die">
  <xs:complexContent>
    <xs:extension base="behaviouralElement"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="fluorescentprotein">
  <xs:complexContent>
    <xs:extension base="behaviouralElement"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="kill">
  <xs:complexContent>
    <xs:extension base="behaviouralElement">
      <xs:attribute name="reference" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="instantiate">
  <xs:complexContent>
    <xs:extension base="behaviouralElement">
      <xs:sequence>
        <xs:element name="argument" type="argument"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="machinetype" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="FanIn" abstract="true">
  <xs:complexContent>
    <xs:extension base="behaviouralElement"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="average">
  <xs:complexContent>
    <xs:extension base="FanIn"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="maximum">
  <xs:complexContent>
    <xs:extension base="FanIn"/>
  </xs:complexContent>
</xs:complexType>
```

--- Logical elements definition -->
<xs:complexType name="minimum">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="addition">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="and">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="or">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="xor">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="not">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="positive">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="vectorialAdd">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="vectorialAverage">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="vectorialAveragesumming">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="vectorialMaximum">
    <xs:complexContent>
        <xs:extension base="FanIn"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="vectorialMinimum"/>
<xs:complexType name="Filter" abstract="true">
  <xs:complexContent>
    <xs:extension base="behaviouralElement">
      <xs:attribute name="shift" type="xs:decimal"/>
      <xs:attribute name="steepness" type="xs:decimal"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="InverseVectorFilter" abstract="true">
  <xs:complexContent>
    <xs:extension base="behaviouralElement"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="linear">
  <xs:complexContent>
    <xs:extension base="Filter"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>

<xs:complexContent>
  <xs:extension base="Filter"/>
</xs:complexContent>
<xs:complexType name="inverseStepfunction">
  <xs:complexContent>
    <xs:extension base="Filter"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="MachineInstance">
  <xs:complexContent>
    <xs:extension base="machineelement">
      <xs:attribute name="machineType"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Machine">
  <xs:complexContent>
    <xs:extension base="machineelement">
      <xs:attribute name="machineType"/>
    </xs:extension>

    <xs:choice maxOccurs="unbounded" minOccurs="0">
      <xs:element name="chemical">
        <xs:complexType>
          <xs:attribute name="value"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="location">
        <xs:complexType>
          <xs:attribute name="value" type="position"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="support">
        <xs:complexType>
          <xs:attribute name="value" type="support"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>

    <xs:element name="in">
      <xs:complexType>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:element name="input" type="input"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="out">
      <xs:complexType>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:element name="output" type="output"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:choice maxOccurs="unbounded" minOccurs="0">
      <xs:element name="detect" type="detect"/>
      <xs:element name="morph" type="morph"/>
      <xs:element name="fork" type="fork"/>
      <xs:element name="move" type="move"/>
      <xs:element name="secrete" type="secrete"/>
      <xs:element name="replicate" type="replicate"/>
    </xs:choice>
  </xs:complexContent>
</xs:complexType>
<xs:element name="kill" type="kill"/>
<xs:element name="flourecentprotein" type="flourecentprotein"/>
<xs:element name="linkweightmodifier" type="linkweightmodifier"/>
<xs:element name="debugoutput" type="debugoutput"/>
<xs:element name="die" type="die"/>
<xs:element name="attach" type="attach"/>
<xs:element name="synapse" type="synapse"/>
<xs:element name="instanciate" type="instanciate"/>
<!−−logics−−>
<xs:element name="average" type="average"/>
<xs:element name="maximum" type="maximum"/>
<xs:element name="minimum" type="minimum"/>
<xs:element name="and" type="and"/>
<xs:element name="or" type="or"/>
<xs:element name="xor" type="xor"/>
<xs:element name="not" type="not"/>
<xs:element name="positive"/>
<xs:element name="vectorialadd" type="vectorialadd"/>
<xs:element name="vectorialaverage" type="vectorialaverage"/>
<xs:element name="vectorialminimum" type="vectorialminimum"/>
<xs:element name="vectorialmaximum" type="vectorialmaximum"/>
<xs:element name="vectorialaverageandsomming" type="vectorialaverageandsomming"/>
<xs:element name="addition" type="addition"/>
<!−−filters−−>
<xs:element name="linearfilter" type="linear"/>
<xs:element name="gaussianfilter" type="gaussian"/>
<xs:element name="sigmodfilter" type="sigmod"/>
<xs:element name="stepfunctionfilter" type="stepfunction"/>
<xs:element name="heavysidefilter" type="heavyside"/>
<xs:element name="inverselinearfilter" type="inverseLinear"/>
<xs:element name="inversegaussianfilter" type="inverseGaussian"/>
<xs:element name="inversesigmodfilter" type="inverseSigmod"/>
<xs:element name="inversestepfunctionfilter" type="inverseStepfunction"/>
<xs:element name="inverseheavysidefilter" type="inverseheavyside"/>
<xs:element name="inversevectorfilter" type="InverseVectorFilter"/>
<!−−CustomMachine−−>
<xs:element name="minstance" type="MachineInstance"/>
</xs:choice>
<!−−establish all links−−>
<xs:choice maxOccurs="unbounded" minOccurs="0">
  <xs:element name="link" type="link"/>
  <xs:element name="vectoriallink" type="vectoriallink"/>
</xs:choice>
</xs:sequence>
<xs:attribute name="name"/>
<xs:complexType name="MachineCollection">
  <xs:sequence>
    <xs:element name="machine" type="Machine"
      minOccurs="1" maxOccurs="unbounded" />
    <xs:element name="mainmachine" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:element type="MachineCollection" name="MachineCollection"/>
</xs:schema>
Appendix B

Legend

Figure B.1: The meaning of the symbols used in Fcode diagrams
Bibliography


