GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Master's Thesis

submitted in partial fulfillment of the
requirements for the course "Applied Computer Science"

Rule-based simulation of dendritic Structures from Cupric Chloride Di-hydrate

Elsa Marie Rommerskirchen

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

28. December 2022

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Kurth
Second Supervisor: Dr. Busscher

Contents

List of Abbreviations

$CuCl_2.2H_2O$	Cupric chlorid di-hydrate
GroIMP	Growth Grammar-related Interactive Modelling Platform
ISO	International Organization for Standardization
L-System	Lindenmayer System
NURBS	Non-Uniform Rational B-Splines
RGG	Relational Growth Grammar
ROI	Region of interest
UHT	Ultra High Temperature
XL	extended L-Systems

Chapter 1

Introduction

The reliable and accurate evaluation of food quality is important. As consumption is steadily increasing, production is under pressure to develop faster and, above all, more resilient production and products. The constant advancement of crop production techniques and plant cultivation can lead to manipulation of the outer appearance of food, which makes quality assessment more difficult. The overall organic food quality is determined by two aspects, namely the process quality and product quality.

Processes and product-related aspects of organic food are therefore described on an international level in guidelines given by the International Federation of Organic Agricultural Movements and by the 'European Regulation on organic production and labeling of organic products' stated in the European Commission No. 834/2007 [1]. Due to the constant development of the agricultural processes these regulations are also updated regularly, with the Council Regulation (EC) No. 834/2007 last updated in 2018.

Still the precise definition of appropriate criteria for organic food quality poses a problem. In the evaluation of organic food quality mainly visual indications, like color and decay, are used. While methods exist to analyse individual aspects of food ingredients like nutrient content or pesticide residue, they are not appropriate to evaluate food quality. These single substance measuring methods are more widely used in comparison of different varieties or processes. Which creates the need for a method to determine product quality at a higher level of integration [1].

The basis for such a method provides the so called cupric chlorid di-hydrate ($CuCl_2.2H_2O$) method. The hypothesis of the method is that the process which builds the sample is reflected in the method result, the crystallization picture [2]. The $CuCl_2$ method offers a different approach to determine food quality and it has been developed through multiple studies showing that it can be used for different kinds of fruit and vegetable extracts or juices [3] [2] and dairy products [4] reliably. The application of the method results in a crystallization structure which shows specific patterns depending on the quality and kind of extract used. A crystallization pattern can be described as a

two-dimensional dendritic structure which forms within a Petri-dish.

The challenge of simulating realistic dendritic growth is present in different areas apart from forestry. In neurobiology dendritic structures are used to analyse the neural morphology of real cells [5], or another application example is the simulation of solidification grain structures [6]. Different approaches were used in these studies ranging from numeric simulations [7] to parallel computing [8].

The main question is as to how well these crystallization patterns can be simulated using a rule-based modelling language. Rule-based simulations have been widely used for modeling of organic growth, since they pose the advantage of parallel rule application based on a logical context. Rule-based languages possess a lot of possibilities and freedom in the creation of simulation models.

This thesis examines example images of crystallization structures formed from milk and implements a simulation model depicting both high and low quality.

The implementation has been split into simplification models replicating simplified versions of the input images and a variable model, which creates distinct structures of high and low quality. The implementation of the variable model presented two challenges. The first challenge is the implementation of the general growth rules like round shape, given by the physical space limitation of the Petri-dish, branching and most problematic the collision avoidance. The second challenge was to be able to create characteristic shapes. The main difficulty lies in the fact that, as research has shown, criteria including features of the overall pattern are most representative for the quality. The criteria which describe such an overall pattern are difficult to put into parameters usable for a simulation model.

This thesis is divided in four main chapters. Firstly the basics of the biocrystallization method are explained and the modelling platform GroIMP is introduced. In the following chapter the crystallization structures are analysed using example images of milk with high and low quality and the main visual features, that will be implemented in the model, are described. Afterwards the implementation of a simplification model and a variable model are explained by use of the XL code. The closing chapter evaluates the model and its results and discusses the limitations and alternative approaches.

Chapter 2

Basics

The crystallization structures used as reference for the simulation model are the result of the biocrystallization process of cupric chlorid di-hydrate and milk. The example images were created as part of research done by the section of organic food quality and food culture of the University of Kassel in Witzenhausen, Germany. In the following the basics of the biocrystallization method and image analysis are described and the modelling platform is presented.

2.1 Biocrystallization

The biocrystallization method is being developed and used with the objective of determining organic food quality, where it has the advantage of determining product quality at a higher level of integration, compared to other methods measuring food ingredients [1].

Using the biocrystallization method to determine food quality compromises two difficulties. Firstly how to reliably create crystallization structures and secondly how to confidably categorize the resulting patterns. For both tasks solutions have been developed and extended over the last years.

2.1.1 Method

The biocrystallization method using cupric chlorid di-hydrate has been standardized for plant products and also tested on dairy products like milk and butter [4]. The main mechanic is the development of crystallization structures from evaporation of a solution of food extract and cupric chlorid di-hydrate ($CuCl_2.2H_2O$).

The solution is created using a specific amount of juice or extract mixed with a defined amount of diluted $CuCl_2$ solution. The ratio of the two components is a crucial part determining the expressiveness of the resulting structure. 6 ml of the mixture is pipetted into a Petri-dish with a radius of 5 cm. The glass plates are then placed in an evaporation chamber ensuring defined

climate conditions during the structure building process. There is an inner and an outer chamber and the conditions, like humidity and temperature, are kept constant in the outer chamber [4] [2].

The structure building process can be divided into two steps, firstly the evaporation of the volatile parts of the mixture till supersaturation, then secondly the crystallization and arrangement of needles in a crystallised state [9].

The evaporation process takes on average around 12-15 hours [4], with the crystallization starting after 12-14 hours from one point and then expanding in a dendritic structure with a hierarchical order of first-, second- and higher level branches [10].

The resulting crystallisation structure shows a pattern specific to the used extract or juice and its quality.

2.1.2 Evaluation

During the analysis of these patterns significant differences for distinct food qualities and additives can be observed. Various approaches have been developed and used to quantify the resulting images in order to reliably classify the crystallization patterns.

Evaluation has been done using computerized image analysis. For example using a structure analysis algorithm, which reflects the spatial linear relationships between grey-scale values of the scanned crystallization structures. For this the length and diameter of parts of the structure were measured and set in relation to the position in the structure, denoted as Region of Interest (ROI) [11].

Using a perceptual strategy the evaluation is done visually by means of defined morphological criteria following the ISO-standardized evaluation panel ISO8587 "Sensory Analysis-Methodology Ranking", which aims at placing test samples in a ranked order [12].

Latest research has shown that visual evaluation using kinesthetic engagement is superior compared to the analytical approach using the ISO panel [13]. The kinesthetic engagement approach is based on Gestalt evaluation, which focuses on the overall pattern instead of atomic features. A Gestalt is an perceptual pattern or structure possessing qualities as a whole that cannot be described merely as a sum of its parts [14]. The crystallisation structure is perceived as a whole and evaluated using given descriptive criteria. These criteria are a fluent interconnected movement, sense of presence in the image, tension in the needle branches from the center to the periphery and consistent dynamic in the filling of the plate [13].

2.2 Modelling Platform

The results of the crystallisation process hold significant importance in determining the quality of food. It has been shown that the overall pattern of the crystallization structure is the main

characteristic used for the evaluation. The aim of this thesis is to be able to reproduce patterns similar to the crystallization structures using a rule-based simulation. For this the software Growth Grammar related Interactive Modelling Platform (GroIMP) is used [15].

2.2.1 GroIMP

GroIMP is a graph-based modelling platform for the simulation of three-dimensional growth [15], which uses the rule-based programming language eXtended L-Systems (XL) [16]. The platform specific programming language XL is a combination of Relational Growth Grammars (RGG) and the imperative and object-oriented programming language Java.

GroIMP provides tools for the simulation of objects in a three-dimensional scene. Basic geometric objects, like spheres, boxes or cylinders, and complex shapes like NURBS (Non-Uniform Rational B-Splines), point clouds or supershapes can be individually created, modified and positioned in a scene using rotation, translation and scaling. Within GroIMP objects are stored as nodes in a graph-structure, so that relationships between any two objects in a scene can be described using ancestor and successor relationships. Features like ray-tracing allow modelling of complex processes like plant growth based on photosynthesis. GroIMP has been used for plant models of different scales ranging from competition models on individual level [17] to molecular models considering genetic aspects [18]. The platform is used in the area of forestry modelling tree architecture or root systems as well as in the area of agriculture simulating apple trees or tomato plants.

2.2.2 Relational Growth Grammars

Relational Growth Grammars are rewriting systems operating on graphs instead of strings [19] and are based on Lindenmayer systems (L-Systems), which were developed by Przemyslaw Prusinkiewicz and Aristid Lindenmayer for the purpose of simulating growth and architecture of plants [20] [21]. L-Systems are replacement rules and have been used in the development of models simulating living organisms focusing on their morphological structure and underlying processes [19]. L-Systems are used for different kinds of models ranging from the level of biochemistry and genetics up to the level of ecological interactions [22].

Replacement rules are given in the form *left-hand side* ==> *right-hand side*, so that with rule application the term on the left-hand side will be replaced by the term on the right-hand side. A example is given in the following section. These rules can be expanded to describe more complex situations using the context of the nodes or attributes as conditions. In every time step the specified rules are applied to all nodes in the graph, which match one of the left-hand sides. Thus the nodes within the structure can be replaced, removed or their attributes can be changed. This way all possible behaviour is defined in the beginning and the graph grows then by itself following the given rule set.

Rule-based languages have advantages for biological modelling, as they are transparent in use, do

not need constant re-compilation and the parallel growth is more intuitive for living organisms and development of their morphological structure [19].

An important feature of XL are queries. Queries are expressions with which a subset of nodes from the graph can be selected. For this predefined functions, like *sum* or *count*, and terms, for example for the edge relations like *successor*, can be used. Using these functions and expressions the graph can be traversed and the node attributes can be analysed. [16]

Application Example

A small application example of replacement rules in GroIMP is given below. Using only two replacement rules the visualization of a Koch snowflake can be created. The code is based on the example code from the Tutorial included in GroIMP [15].

```
protected void init ()
[
    Axiom ==> F(6) RU(120) F(6) RU(120) F(6);
]

public void run ()
[
    F(x) ==> F(x/3) RU(-60) F(x/3) RU(120) F(x/3) RU(-60) F(x/3);
]
```

The first rule initializes the basic triangle form consisting of three cylinder objects given by the turtle geometry command F(). Inside the brackets the length of the cylinder is given as an attribute. The command RU() rotates the cylinders around the specified angle.

The second rule replaces each cylinder object with a sequence consisting of four partially rotated cylinders, each a third of the original length. For this, the value of the attribute length is indicated by the variable x. The second rule can be applied repeatedly to add smaller equilateral triangles. The results of four iteration steps can be seen in the images in figure 2.1.

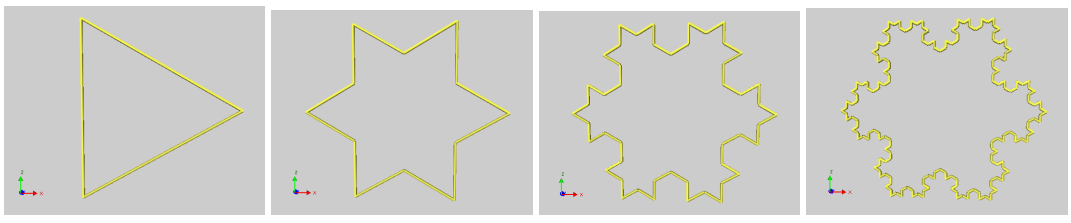

Figure 2.1: Example of Koch snowflake implemented in GroIMP

Chapter 3

Example Images

Since the model aims at recreating the morphology of the crystallization structures, images were used as the basis for the simulation. The images are taken in large numbers during the crystallization process at the section of organic food quality and food culture in Witzenhausen as part of their experimental design. A camera is positioned at the top of the chamber, overlooking all Petri-dishes, and photographs the glass plates every five minutes to document the growth. That way for each structure the whole crystallization process is documented and depicted in images. The images are stored in a data base and used for further analysis.

The scope of the implementation was the simulation of structures from biocrystallization with milk of high and low quality. For this purpose the images given in the following figure 3.1 were used. The image on the left shows the crystallisation structure showing a pattern characteristic for high quality, which was created using pasteurized organic milk. The image on the right shows a crystallisation structure formed from UHT milk representing lower quality. In both cases the mixture was created in a ratio of 200 parts milk to 150 parts $CuCl_2.2H_2O$. In the following section the main characteristics of the example images used as basis for the simulation are described.

3.1 Characteristics of the Images

To interpret these images in a systematic way different criteria have been defined [14] [13] [23]. The criteria focus on different aspects of the images and vary in their range of detail. They range from quantifiable local features to Gestalts or implicit motions in the whole pattern. Single local morphological features can be the amount of side needles or the angles of side needles. Criteria for a kinesthetic approach focusing on implicit motions in the whole pattern are the dynamic, basic tension, connectedness and uniformity.

Structures of high quality for example as seen in the left image in figure 3.1 show a spreading dynamic from the center to the edge of the Petri-dish. The structures looks self-feeding and show

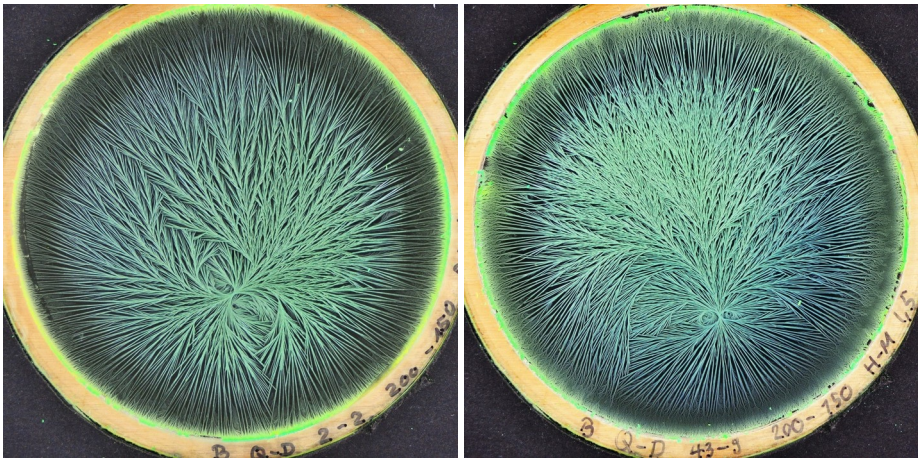

Figure 3.1: Crystallisation structures from $CuCl_2$ method with milk of high quality (left) and low quality (right) (Source: Faculty of Organic Agricultural Sciences of the University of Kassel)

tension. The branches should have an overall fluent connection with a distinct direction towards the edge. There are no noticeable changes in direction and overall the structures seems uniform. The kinesthetic criteria have been proven to be superior in indicating quality, but they also are more difficult to summarize in parameters usable for a simulation model.

Firstly it was necessary to determine the basic features characteristic for the crystallisation structures, which should be implemented in the model. The main features of the image are the center of the structure, the branches emerging from the center and the growth to the edge of the Petri-dish. The growth towards the edge can be separated into three phases. The criteria of kinesthetic engagement were taken into account in the development of the differences between the distinct quality.

3.1.1 Center

The center is the point of the structure, where the growth starts. The branches grow out of the center in two opposing directions with multiple side branches on each side. The division in two opposing directions and the way the branches spread creates a axial symmetry along the center. The center can form randomly at any position on the glass plate, but for the analysis the images are positioned in a way that the center is in the lower middle region of the image. This simplifies the comparison and recognition of patterns.

Several centres and thus several structures can form on a single glass plate, but in the simulation only single structures are implemented.

3.1.2 Phases

The structures show a distinct pattern in the development towards the edge, which can be divided into three phases, as seen in the cutout in figure 3.2.

Figure 3.2: Segment of input image showing phases of structure (Image source: Faculty of Organic Agricultural Sciences of the University of Kassel)

The growth in the inner region is characterized by frequent and random branching while the needles spread out in different angles creating a crowded pattern. As the inner region offers less space, side branches only grow over short distances and form numerous short side branches.

The second phase is distinctly visible in the images through the contrast of colour, as the internodes decrease in diameter towards the edge the background color becomes more prominent. Another characteristic of the second phase is that the internodes grow straight on without branching or bending.

The third and last growth phase starts close to the edge and is very narrow. It is in contrast to the second phase as the internodes branch again randomly.

3.2 Differences in Quality

Based on the criteria used for the kinesthetic evaluation low quality structures can be described as less prominent or less pronounced compared to high quality structures. The structure of a low quality extract shows a comparatively very low spreading dynamic. The branches seem weak, limp and resilient with no individual initiative. There is more variance in the structure with full and overgrown regions next to empty ones. In general the overall appearance of the structure is not uniform or regular. One major difference between the samples and therefore an indicator of quality is a higher variance in the single morphological features throughout the entire structure.

It is visible that the overall orientation of the structure is uneven with one side almost overgrowing the whole plate, while the opposing side of the structure is barely differentiated. Compared to the symmetry seen in the sample of high quality, the structures of low quality appear irregular. It appears as if the center is oriented differently, so that no symmetry can emerge.

The distinction into phases also is more variable compared to the sample of higher quality. In high quality crystallisation structure the second phase starts earlier and the third phases starts further to the edge compared to low quality structures and the transition between the phases is more coherent in the higher quality. This variance in the lower quality leads to a more crowded and fuller looking area near the edge. In low quality structures the growth does not always reach the edge but sometimes stops beforehand. These cases are not taken into account and are not included in the model.

Chapter 4

Implementation

The aim was to implement a rule-based simulation model in XL, which creates dendritic structures showing characteristics of the crystallisation structures from the $CuCl_2$ method. The focus is put on depiction of the morphology of the structure, whereas temporal development and underlying chemical processes are not included. A special emphasis has been the display of distinct qualities as seen in the example images.

Since the crystallization structures show a dendritic growth pattern, the model represents a graph-like structure based on nodes and internodes. Nodes are points, where a branch forks or changes its direction. There are different kinds of nodes in this model, ones that are for branching within the structure and ones that mark the start of new growth phases. Nodes are not always represented by visible objects in the scene, but can be of purely conceptual use. *Internodes* are implemented as an extension of the cylinder objects and thus always have a visible representation. They can be modified using their default attributes diameter and length or additional attributes.

To determine the scale of the structure the example images were used as reference. The Petri-dishes seen in the images have a diameter of 10 cm which matches the visualized ten by ten grid of the coordinate system in GroIMP. Thereby 10 length units in GroIMP correspond to 10 cm. Some specific measurements were taken on the example images in order to deduce reasonable values for attributes of the objects in the model. For example the diameter was measured at a few points within the structure to get the corresponding value for the diameter of the internodes in GroIMP. The distance from the edge at which the different phases start was as well measured at random points to estimate a threshold for the growth rules in the model.

The process of implementing a simulation model was divided into two approaches. The first approach consists of simplifications of the example images in XL. For this the images were used as templates and a reduced one-step deterministic model was created for each image. The models will always create the exact same structure depicting a simplified replication of the example image.

The second approach is a general simulation model which grows over time until it is fully devel-

oped into a crystallization structure. It combines options for high and low quality in one model. This model utilizes randomness and a function of the normal distribution enables variance in the growth. Thus the model is able to create different looking structures in each run.

4.1 Simplifications

The simplification models are hard-coded dendritic structures, which replicate the basic pattern of the input images. Both qualities are implemented/realised in their own model. The models only consist of internode objects and rotation operations. The length, position and rotation of every internode is specified in the code and thus predefined before the execution of the model.

A pivotal point was to determine the level of detail. Since these simplifications are specified manually, a higher level of detail increases the effort exponentially. Whereas the higher level of detail might not increase the expressiveness. It is important to present the overall pattern and capture differences that define distinct qualities, without including unnecessary detail.

To manually create the replications each internode is manually positioned and scaled in the XL code according to the example image. So it was necessary to determine the length and rotation of the internodes efficiently. Therefore the example images were loaded as textures into the background of the GroIMP scene, so that the structure could be directly compared during each step of construction.

4.1.1 Textures

Textures can be defined by the user and applied to the surface of objects. In this case the texture was added to a parallelogram, which was positioned in the background of the scene. The size and position of the parallelogram were chosen to fit the coordinate grid, so that the edge of the growth structure in the image stayed within the raster.

An image can be added as a texture to an object using a self-defined customisable shader. The image for the shader can be selected using the register 'Panel' in the user interface, there under the entry 'Explorers' the option 'Images' can be selected. In the new window using the button 'Object' the option 'New' can be selected. When the file directory is opened an input image can be chosen and uploaded. Thereupon the image can be chosen in the attribute editor of the shader. In the XL code a shader object can be initialized and set as the texture for any visible object in the scene.

4.2 Variable Model

The main task was to create a comprehensive model, which creates structures of characteristic form for crystallisation structures of high and low quality. For the model multiple modules, growth rules and calculations were implemented to simulate different growth behaviour.

The modules, growth rules and functions are the same for both kind of quality. A set of specific parameters is used to influence the growth and by that depict the difference in quality. To select the parameter set, buttons in the interface are used.

Models in GroIMP are controlled using auto-generated buttons in the user interface. Two default buttons named *Reset* and *Stop*, which reset and stop the simulation, respectively, are always present. For each public function in the code two additional buttons are created. The first is named after the function and runs the function once. The second button is a *Run* version, which executes the function repeatedly until the *Stop* button is pressed.

To execute the model one button selecting the quality has to be pressed first. Then the user can use the buttons named "grow" or "Run grow" to apply the replacement rules creating the structure.

In the following section the code of the modules, parameters and functions used in the simulation is shown and explained in detail.

4.2.1 Modules

Modules are objects defined in the XL code, they can, like java objects, inherit attributes and methods from existing objects. Modules are the basis on which the replacement rules are deployed. Using different modules can make it easier to distinguish which rules should be applied in specific situations. The following modules were defined for the simulation.

```
1  module Origin() extends Sphere(0.01);
2  module Radius() extends Circle(RADIUS);
3
4  module Center() extends Sphere(DIAMETER);
5  module Node extends Sphere(0.025);
6  module Decease();
7  module Phase2(float diameter);
8  module Phase3(float diameter) extends Sphere(0.01);
9  module Internode(super.length) extends F(length, DIAMETER, 7);
10 module Branchless(super.length, super.diameter) extends F(length, diameter, 7);
```

The first two modules are not directly part of the resulting structure, but are used for calculations and visualization. The module *Origin*, which extends the object class *Sphere* with a radius of 0.01 length units, will be positioned at the coordinate system's origin and is used in calculations to determine the distance of objects from the edge. It is implemented as a Sphere object, so that the predefined distance-function can be used in the calculations.

The module *Radius* has the sole purpose to give a visual representation of the edge of the Petri-dish like seen in the example images in figure 3.1. Hence it extends the predefined geometric object

class *Circle*, which creates a two-dimensional circle with the specified radius of 5 length units.

The module *Center* also extends the object class *Sphere* and represents the center of the structure, from where the structure starts growing. The position of the center is influenced by the parameters determining the quality.

The module *Node* is used for nodes placed within the growing structure at branching points, where the growth continues consistently. Whereas the modules *Decease* is used to stop a branch from growing. The modules *Phase2* and *Phase3* mark where the growth pattern changes from the inner region to the outer region of the structure. They are added to the structure through a replacement rule when the size of the structure grows over a specified threshold. The thresholds for both modules are checked as conditions in the growth rules and their values are defined by the chosen parameter set. The occurrence of the modules *Phase2* and *Phase3* are thus dependent on the selected quality of the model.

The modules *Internode* and *Branchless* both extend the turtle command F and are visualised as cylinders. They differ in their functionality and are used at different parts of the structure. *Internodes* are placed within in the inner region, where the structure branches. Modules of type *Branchless* on the contrary are used in the outer region following *Phase2* and *Phase3* objects or *Decease* objects. As the name suggest, they are used in areas where the structure does not branch. The same structures could be created without this distinction by instead defining the context and conditions for the left-hand side of the replacement rules, but the different modules aim to make the code more easily readable.

4.2.2 Constants

Some values, which are used for basic computation or visualization, are defined in the beginning of the simulation independent of the user. Since the values will not be changed throughout the simulation, they are set as constants to increase readability of the code.

```
1 const float RADIUS = 5.0;
2 const float DIAMETER = 0.05;
3 const int NUMBER_OF_BRANCHES = 8;
```

The float constant RADIUS is set to the value 5.0 and is used in the model for calculation purposes. The growth rules to create the distinction of the structure into phases depend on the distance to the edge of the Petri-dish, which is calculated for each node using the radius.

The constant DIAMETER is defined as the float value 0.05 and is used in the module definition as the diameter of the *Internode* objects within the structure. It is used as initial value in growth rules

where the diameter decreases repeatedly for multiple consecutive objects.

The constant NUMBER_OF_BRANCHES determines the amount of branches growing directly out of the center of the structure. It is initialized with the int value 8, similar to the amount counted in the example images.

4.2.3 Parameters

The following parameters are set dependent on which button the user selects and determine the quality the structure depicts. There are two possible parameter settings creating high or low quality. The parameters influence different aspects of the simulation including position and orientation of the center, internode growth, phases and other values.

```
1 const float START_POSITION_DISTANCE;
2 const float INITIAL_BRANCHING_ANGLE;
3 const float INITIAL_BRANCHING_STD;
4 const float COLLISIONFREE_ANGLE;
5 const float BENDING_STD;
6 const float LENGTH_MEAN;
7 const float LENGTH_STD;
8 const float SURROUNDING_START;
9 const float PERIPHERY_START;
```

The parameter START_POSITION_DISTANCE determines the distance between the center and the origin. It is defined using the *normal* function, which computes a random number from a normal distribution with a given mean and standard deviation. The mean and standard deviation are defined in the parameters and depend on the chosen quality. Values greater than the threshold defined by the difference between the parameter Surrounding start and the constant RADIUS will not result in growth, as the position is too close to the edge. A smaller distance increases the area where the branches can grow, thus giving an advantage in development of a fuller structure. The value returned by the *normal* function could exceed the area set for the inner part of the structure, which would prevent the structure from growing. Therefore the value of the parameter is tested using the parameter SURROUNDING_START in an additional step.

The parameter INITIAL_BRANCHING_ANGLE is used to choose the general direction of the first branches growing from the center. The branches grow in two opposing directions creating a kind of symmetry. The parameter specifies one value, a second value is calculated to be the opposite site by adding 180. Reasonable values for the parameter can lie in the range of 0 to 180. High quality structures form the branches orthogonal to the edge, so that both sides have equal conditions leading to a balanced symmetry. For low quality structures one side faces the border

and thus is underdeveloped leading to a less balanced appearance.

The parameter INITIAL_BRANCHING_STD determines the range of directions in which the first branches grow from the center. It is used as the standard deviation in a normal function with a mean of zero. The resulting variance is used as rotation angle around the previously defined INITIAL_BRANCHING_ANGLE. A reasonable value can be chosen from the range between 0 and 90, as the standard deviation is set around a mean of zero and is computed for both sides of the branches emerging from the center.

The parameter COLLISIONFREE_ANGLE is used in the growth function to avoid collisions between the branches. The replacement rule tests for collisions using a cone to check if the area in front of the current node is empty before the branch grows in that direction. The value of the parameter COLLISIONFREE_ANGLE defines the half-opening angle of the cone. The value should be positive and values greater than 30 results in unnatural looking structures. As the values is used as the half-opening angle, a value of 30 would result of a cone with an opening angle of 60, which would prevent a majority of growth in the structure. Due to the nature of the collision checking function, collisions cannot be excluded, but a higher value for the parameter decreases the amount of collisions.

The parameter BENDING_STD sets the standard deviation for the normal function determining the branching angle. It specifies the range of the bending angle for branches. The standard deviation is set around a mean of zero, so the resulting angle can be positive or negative. Values used for the parameter should be positive and should not exceed 30, whereat the boundary for high values is not as distinct and might vary. The value is used in the turtle command RU, which rotates around the local up-axis, causing a turn to the left for negative values and to the right for positive values. Larger values create a more angular structure, whereas too small values prevent the structure from spreading over the whole area.

The parameter LENGTH_MEAN is used as the mean value for the normal function calculating the length for *Internode* objects within the structure. Negative values are not reasonable and larger values make the structure appear empty, as over the length of an *Internode* no branches emerge.

The parameter LENGTH_STD defines the standard deviation used in the normal function calculating the length of *Internode*. Using the normal function can result in negative length values, which would create a cylinder growing in the opposite direction. To avoid this, negative values are set to zero. A reasonable value for the parameter LENGTH_STD should be determined depending on the parameter LENGTH_MEAN.

The parameter SURROUNDING_START defines the threshold for how far from the edge the second growth phase starts. It is used as a condition in a growth function to place *Phase2* objects. The distance was measured in the example images and used as a reference value for the parameter. The value has to be positive and significantly smaller than the constant RADIUS. Additionally the value should be greater than the parameter PERIPHERY_START.

The parameter PERIPHERY_START defines the threshold for how far from the edge the third growth phase starts. It is used as a condition in a growth function to place *Phase3* objects. The parameter has to have a positive value and should be smaller than the value set for the parameter SURROUNDING_START. For models of low quality the parameter is chosen from a normal distribution, so that the variance creates a less uniform structure.

Both qualities have their own buttons setting the parameters for the simulation to the values given in the following table 4.1.

Constant	high Quality	low Quality
START_POSITION_DISTANCE	normal(1.6, 0.5)	normal(2.5, 0.5)
INITIAL_BRANCHING_ANGLE	-90	0
INITIAL_BRANCHING_STD	30	40
COLLISIONFREE_ANGLE	10	5
BENDING_STD	15	10
LENGTH_MEAN	0.3	0.4
LENGTH_STD	0.2	0.3
SURROUNDING_START	2	1.3
PERIPHERY_START	0.25	normal(0.3, 0.1)

Table 4.1: Parameter settings for high and low quality simulation

4.2.4 Functions

The main functionality of the model is determined through the methods defining the growth of the structure. The following section can be roughly divided into these three parts: growth functions, auxiliary functions and mathematical functions.

Init

At compilation the pre-existing init() function is executed. The default object Axiom, which is always present in any project, serves as the starting point.

```
1 protected void init (){
2     Axiom ==> Origin Radius() RL(90) Axiom;
3 }
```

The first rule defined in the init() function is executed before any quality is chosen by the user. In this function the objects of the modules Origin and Radius are added to the scene while the object Axiom remains unchanged.

Using the turtle command RL(90) the local intern coordinate system is rotated by 90 degrees, so that all following turtle commands are based on the plane spanning between the x- and y-axis.

Center

The replacement rule in the following function determines the position of the center.

```
Axiom ==> RU(-45 + Math.random() * 2 * 45) M(START_POSITION_DISTANCE) Center();
```

To position the center first a random angle, which lies in the lower quarter of the circle, is chosen using the turtle command RU() and the predefined function Math.random(). The distance to the center is set to the parameter determined by the chosen quality. The turtle command M() moves the specified distance in the previously determined direction and then the Center object is placed there. After this rule application the object Axiom is no longer in the scene.

Realistically the center could be positioned anywhere in the circle, as the crystallisation starts randomly. But since the images are always oriented with the center in the lower region during the image analysis process, the position is set in the same range.

The following function determines the growth of branches emerging from the center. The amount of branches is specified by the constant NUMBER_OF_BRANCHES. The rule execution depends on the return value of the function cangrow(), which is explained in a following paragraph.

```
c:Center, (cangrow(c)) ==>
    {long branchCounter = count((* c-minDescendants->n:Internode *));}
    if(branchCounter < NUMBER_OF_BRANCHES)(
        {int angle = (int) normal(0,INITIAL_BRANCHING_STD);}
        if(branchCounter % 2 == 0)(
            c [RU(INITIAL_BRANCHING_ANGLE + angle) Internode(0.3) Node]
        )else(
            c [RU((INITIAL_BRANCHING_ANGLE + 180) + angle) Internode(0.3) Node]
                    )
    )else(
    );
```

On the right-hand side of the replacement rule, the If clause determines the direction in which the branches will grow, alternating in direction. The directions are determined by the parameters INITIAL_BRANCHING_ANGLE and INITIAL_BRANCHING_STD and are set using the function RU().

The first branches are composed of an object *Internode* with a fixed length of 0.3 followed by a *Node* object. The center object remains unchanged, whilst the if-clause of the rule is executed. When all branches from the *Center* are set as indicated by the variable branchCounter, the *Center* object is removed.

Branching in the structure is created using the symbols "[" and "]". The opening bracket stores the current state of the turtle on a stack until it is retrieved by the closing bracket.

Growth rules

Growth inside the structure is determined by the following rules. There are three different replacement rules defined for Node objects. Which rules are executed depends on the functions cangrow() and die(). The purpose of the function cangrow() is to test the distance between the current node and the edge and the function die() tests the amount of other nodes in a specific distance.

The first growth function is applied to nodes which are in the inner part of the structure. Depending on the position of the node the function calculateBending() calculates the angle at which the branch forks. To avoid collisions of *Internode* objects the amount of nodes and internodes in front of the current node are counted in the if-clause. The collision avoidance test uses a query which counts the amount of nodes and internodes which lie within a distance of one length unit and are in front of the node. The width of the area tested for collision depends on the parameter COLLISIONFREE_ANGLE.

```
1  n:Node, (cangrow(n)) ==>
2      {float m = calculateBending(n);
3      de.grogra.vecmath.geom.Cone cc = cone(new Point3d(location(n).x + DIAMETER, location
           (n).y + DIAMETER ,location(n).z) , rotateVector(direction(n), m) ,
           COLLISIONFREE_ANGLE);
4      }
5      if (empty((* f:Internode, ( ((distance(n, f) < 1) && (f in cc)) ) *))
6       && empty((* nn:Node, ((distance(n, nn) < 1) && (nn in cc)) *))
7      ) (
8          Node [RU(m) Internode(determineLength(n)) Node]
9      ) else (
10         [n]
11     );
```

The structure only continues to grow in that direction if the resulting set is empty. The growth of the node results in a branching at that point. In case the query returns a non empty set, no branches emerge from this node at this time step. Instead the node is set in brackets, which changes the

constellation of the underlying data graph, so that changes in following replacement steps do not influence the existing structure.

The next replacement rule initiates the second growth phase towards the edge of the glass plate. The return value of the function cangrow() indicates that the distance to the edge is below the specified threshold, so a new type of node is instantiated. If the structure grows too close to the border the *Node* objects get replaced by objects of the type *Phase2*. *Phase2* objects have no visual representation, since they are used as transition points between growth phases.

```
1  n:Node, (!cangrow(n)) ==> Phase2(DIAMETER);
2
3  n:Node, (die(n)) ==>
4      if(Math.random() > 0.5) (
5          Decease
6      )else(
7          n
8      );
9
10 Decease ==> Branchless(0.1, 0.04) Branchless(0.1, 0.03) Branchless(0.1, 0.02);
```

The last rule applicable to Node objects uses the return value of the function die() as a condition. When the function die() returns true, the node will with a 50% chance be replaced by an object of the type *Decease*, otherwise it stays unchanged. *Decease* objects will develop into a group of three continuously thinning *Internode* objects, which do not evolve further. This rule aims at further decreasing collisions by removing nodes from too densely filled areas. Since rules are applied in parallel, multiple nodes influencing each other would both be removed if not for the probability condition.

Phases

The change in appearance from the center towards the edge is characteristic for the crystallization structures. Near the edge the structure stops branching and instead spreads out into *Internodes*, which get thinner towards the edge. Close to the edge the needles branch again, but more intensely compared to the inner region of the structure. This creates a very distinct pattern, which varies for the two qualities and is thus dependent on parameters. To recreate this pattern the following three rules are used.

```
b:Phase2(x) ==> RP(new Point3d(2*location(b).x, 2*location(b).y, 2*location(b).z), 0.8)
    [RU(-7.5 + normal(0, 2.5)) Branchless(0.25, x*0.8) Phase3(x*0.8)]
    [Branchless(0.25, x*0.8) Phase3(x*0.8)]
    [RU(7.5 + normal(0, 2.5)) Branchless(0.25, x*0.8) Phase3(x*0.8)];

c:Phase3(x), ((distanceToEdge(c) > PERIPHERY_START)) ==> Branchless(0.1, x*0.8) Phase3(x
    *0.8);

c:Phase3(x), (distanceToEdge(c) <= PERIPHERY_START) ==> [RU(-7.5 + normal(0, 2.5))
    Branchless(distanceToEdge(c), x)]
    [Branchless(distanceToEdge(c), x)]
    [RU(7.5 + normal(0, 2.5)) Branchless(distanceToEdge(c), x)];
```

This growth pattern starts with a *Phase2* object. The object *Phase2* branches into three identical parts, which are oriented at slightly different angles. From this point on the branches do not fork randomly, but continue straight ahead getting smaller in diameter. Following the *Branchless* object a *Phase3* object is placed. In the replacement rule the function RP(), which creates a tropism towards a point, aids to direct all growth straight outwards. This way the structure grows towards the outside independent of the orientation of the *Internode* beforehand.

There are two rules applicable for *Phase3* objects, which differ in the condition concerning the distance towards the edge. The first rule is executed when the distance is greater than the threshold defined by the parameter PERIPHERY_START. In this case the growth will continue with a *Branchless* and a *Phase3* object. The second rule creates the second branching point, starting the last growth phase, where the *Phase3* objects are replaced by three straight branches, which grow till the edge of the Petri-dish.

Auxiliary Functions

Auxiliary functions were needed in calculations made in the conditions of the replacement rules and for the definition of object attributes.

The function cangrow() tests if the shortest distance from the current node to the edge is greater than a specified threshold. First the shortest distance to the edge is calculated by the function distanceToEdge(). The distance is then compared to the quality dependent threshold set by the parameter SURROUNDING_START and the resulting boolean is returned.

```
protected boolean cangrow(Sphere n){

    float distance = distanceToEdge(n);
    boolean isThereSpace = distance > SURROUNDING_START;
```

```
5
6     return isThereSpace;
7 }
```

The function distanceToEdge() calculates the distance towards the edge using the predefined distance() function and the origin object. The function distance() determines the euclidean distance between two objects extending the GroIMP-own class 'Node'. Since the class Sphere extends the class Node, the distance between a node and the edge can be easily calculated using the object Origin and the constant RADIUS.

```
1 protected float distanceToEdge(Sphere n){
2     float distanceToEdge;
3     [
4         o:Origin::>{
5             distanceToEdge = distance(o, n);
6         }
7     ]
8     return RADIUS - distanceToEdge;
9 }
```

Another auxiliary function calculateBending() calculates the angle between newly emerging branches. While the main branch continues straight ahead, a new branch is rotated according to the angle returned by this function. First a random value is chosen from a normal distribution with a mean of zero and a standard deviation given by the quality dependent parameter BENDING_STD. This values will be used as the angle but is beforehand adapted according to the distance of the node to the center. If the node lies within a range of 0.7 length units near the center the value will be manipulated depending on an auxiliary local coordinate system. This coordinate system is positioned around the center of the structure and is oriented following the line between center and origin and its orthogonal, as depicted in figure 4.1.

The angle is set depending on which quarter of the coordinate system the node lies within. Branches in the upper left and lower right quarter should bend to the right and branches in the upper right and lower left should bend to the left. To achieve bending to the right the value for the angle has to be negative, for a curve to the left the value is positive. This is used to increase the opening angle of the branching from the center, creating a wider and fuller structure and a more prominent symmetry. In a second step the angles from nodes that are further away from center are decreased by a defined factor, making the branches appear smoother and less abrupt.

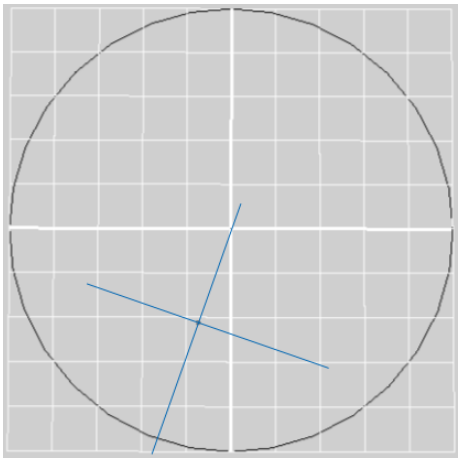

Figure 4.1: Visualization of local coordinate system

```
protected float calculateBending(Node n){

    float m = normal(0, BENDING_STD);

    Point3d ncoords = location(n);

    Point3d center = new Point3d();
    [c:Center ::> center = location(c);]

    float dist = getDistance(center, ncoords);

    Vector3d position = new Vector3d(ncoords.x - center.x, ncoords.y - center.y, ncoords
        .z - center.z);
    Vector3d quarter = rotateVector(new Vector3d(position.x, position.y, position.z), (
        int) calculateAngle(new Vector3d(center.x, center.y, center.z), new Vector3d(0,
        1, 0)));
    Vector3d output = new Vector3d(-quarter.x, -quarter.y, -quarter.z);

    if(dist < 0.7){
    if(output.x * output.y > 0){
        m = Math.abs(m);
        }else{
            m = - Math.abs(m);
        }
    }

    if(dist > 1.5){
        if(m > 1){
            m = m * 0.7;
        }
```

```
28      }
29
30      return m;
31  }
```

The function determineLength() is used to set the length of the *Internode* objects in the growth rules. The length is chosen as a random value from a normal distribution with a mean and standard deviation defined by the parameter set from the chosen quality. Dependent on the parameters the normal function could return negative values, which would result in a backwards growing *Internode*. This would create an unnatural appearance not seen in crystallization structures. To avoid this, negative values are set to zero. A cylinder with a length of zero is not visualized in the scene and results in two nodes on the same position, which increases the amount of branches emerging in that area. Large values for the length of the *Internodes* can cause the structure to exceed the border set by RADIUS. To avoid this the length is compared to the distance to the edge and reduced accordingly.

```
1  protected float determineLength(Node n){
2
3          float length = normal(LENGTH_MEAN, LENGTH_STD);
4
5          if(length < 0){
6                  length = 0;
7          }
8
9          float distance = distanceToEdge(n);
10         if(distance  < length){
11                 length = distance - SURROUNDING_START;
12         }
13
14         return length;
15 }
```

The last auxiliary function die() is used to avoid crowding in the structure. In a query the nodes within a specified distance to the current node are counted. If the count surpasses a specific number, in this case two, the return value of the function is set to true, leading with a 50% chance to a removal of that node.

```
protected boolean die(Node n){
    boolean remove = false;
    if(count((* nn:Node , ((n != nn) && (distance(n, nn) < 0.5)) *)) > 2){
    remove = true;
    }
    return remove;
}
```

Mathematical Functions

Although java, which provides a variety of mathematical functions, is included in the XL language, three additional auxiliary functions for small mathematical operations were implemented.

The function calculateAngle() determines the angle between two vectors. It is used in the calculations of the local coordinate system in the function calculateBending(). To avoid a division by zero, the denominator is checked and is when necessary set to an approximate value.

```
protected float calculateAngle(Vector3d a, Vector3d b){
    float inner_product = (a.x * b.x + a.y * b.y + a.z * b.z);
    float absolute = Math.sqrt(Math.pow(a.x, 2) + Math.pow(a.y, 2) + Math.pow(a.z, 2)) *
         Math.sqrt(Math.pow(b.x, 2) + Math.pow(b.y, 2) + Math.pow(b.z, 2));

    if(absolute == 0){
        absolute = 0.0000001;
    }

    float angle =  Math.toDegrees(Math.acos(inner_product / absolute));

    return angle;
}
```

The function rotateVector() rotates a vector around a given angle. The rotation is executed similarly to the RU() function as a rotation on a plane.

```
protected Vector3d rotateVector(Vector3d v, float angle){
    if(angle > 0){
    angle = 360 - angle;
    }

```

```
6       double rotatedX = Math.cos(Math.toRadians(angle)) * v.x - Math.sin(Math.toRadians(
            angle)) * v.y;
7       double rotatedY = Math.sin(Math.toRadians(angle)) * v.x + Math.cos(Math.toRadians(
            angle)) * v.y;
8       double rotatedZ = v.z;
9       Vector3d rotated = new Vector3d(rotatedX, rotatedY, rotatedZ);
10
11      return rotated;
12  }
```

The last function in the code is called getDistance() and it is used to calculate the euclidean distance between two objects of the Point3d class. The predefined function distance() can not be used, since Point3d does not extend the GroIMP class 'Node'.

```
1 protected float getDistance(Point3d a, Point3d b){
2     float dist = Math.sqrt(Math.pow(a.x - b.x, 2) + Math.pow(a.y - b.y, 2) + Math.pow(a.
          z - b.z, 2));
3     return dist;
4 }
```

Chapter 5

Conclusion

This chapter is divided into three main parts. In the first section the results from the simulation models are presented and evaluated. In the following section the limitations of the implementation are outlined and lastly alternative approaches are discussed.

5.1 Results

Rule-based languages like XL are widely used and appropriate for the modelling of dendritic structures and have been used in different fields of application in the area of forestry or biology for the simulation of root or tree architecture. Still the task simulating structures of biocrystallisation processes poses a novel field of application. The main difficulties in the implementation were the definition of rules leading to the characteristic pattern. As shown in other studies a comprehensive description of these structures exceeds numerical parameters [13].

The results of the simplification models and the variable model are given in the following paragraphs.

5.1.1 Simplifications

The aim of the simplifications was to reduce the structure of the input images to the basic pattern, where the differences of quality is still visible. The replicated structures with the input images in the background are shown in figure 5.1. The image on the left shows high quality and the image on the right shows the structure from low quality.

One general problem of the creation of simplifications is to determine the level of detail. In this case the division of the structure into three growth phases is lost, as the branches in the outer region are too small and too many to include.

The simplification of the high quality structure is relatively well depicted, showing tension and

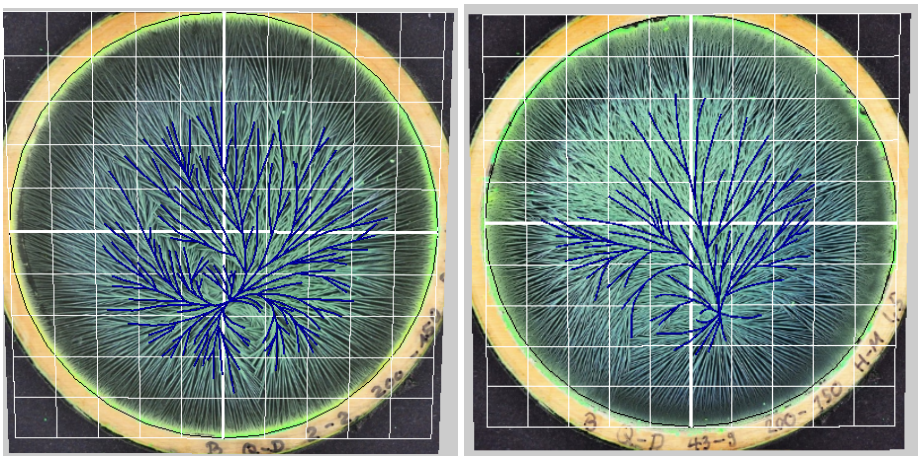

Figure 5.1: Simplification model of high quality milk (left) and low quality milk (right) with texture of the example images in the background

growth directed towards the outer region. Whereas the simplification of the low quality still shows too much initiative in growth and regularity, indicating a higher quality than it should depict. Additionally due to the level of detail the amount of collisions in structures of lower quality is not appropriately portrayed.

5.1.2 Variable Model

The variable model described in the chapter implementation depends on parameters to depict the different qualities. In the following section images of structures created by the variable model are shown.

The model was executed repeatedly to create multiple structures to give an overview of resulting structures. The results from the model run with the high quality parameter set can be seen in the figure 5.2.

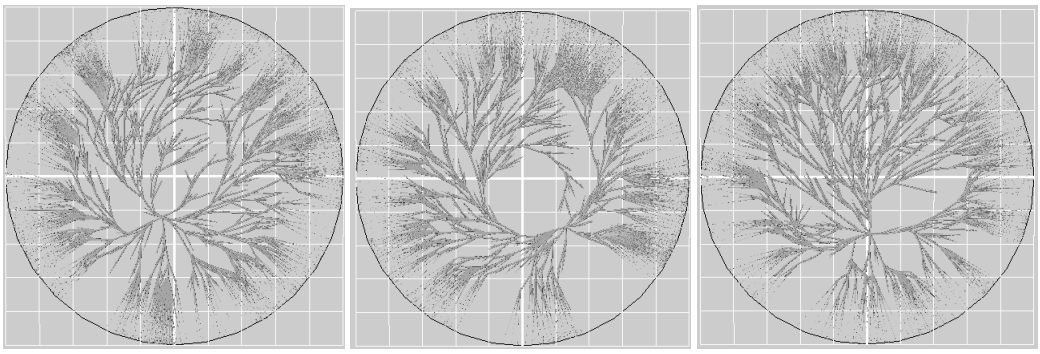

Figure 5.2: Results from the simulation of a high quality crystallization structure

The structures resulting from the variable model executed with the low quality parameter set can be seen in figure 5.3.

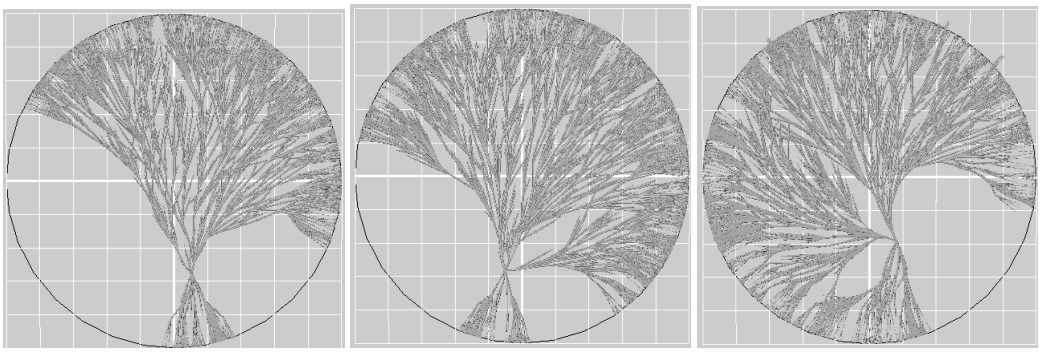

Figure 5.3: Results from the simulation of a low quality crystallization structure

5.2 Evaluation

The structures do show some characteristics of the example images and the use of parameters creates an obvious difference between the portrayed qualities.

5.2.1 Visual Evaluation

The structures representing high quality show growth over the whole area with a orientation towards the edges, tension in the branches and a clear division into three growth phases. A deficiency of the simulation is the stronger irregularity compared to the example images. Within the structures some areas remain empty, which is especially noticeable near the edge, whereas other regions are too crowded resulting in collisions. Additionally there are much less small branches in the structure compared to the example images.

The simulation of low quality showed other difficulties. Using the parameters the changed position and orientation of the center result in an imbalance of the structure as seen in the example images. The growth towards the edge is also less uniform. The amount of collisions are increased compared to the structure simulating high quality. An unrealistic appearance is caused by the large gaps, as the structure fails to fill out the whole area.

One basic problem of the model is that not every execution creates a reasonable result. With the variance and randomness in the growth rules the structure can develop in unnatural ways or even stop growing at early stages leaving an uncharacteristic pattern.

To determine in detail as to what extent the structures depict the essence of milk biocrystallization structures is beyond me to recognize.

5.2.2 Sensitivity Analysis

The parameters determining the represented quality of the structure form a crucial aspect of the simulation model. The parameters can be changed in the first part of the code to allow individual adaptation of the structures. In the following the influence of these parameters on the appearance is described based on structures created from the simulation model. To best display the influence of changes in one parameter, the remaining parameters are kept constant. They are set to the parameter settings described in table 4.1 for high or low quality.

START_POSITION_DISTANCE

The parameter START_POSITION_DISTANCE determines how far from the origin the center is positioned. Example images of structures with different starting positions are given in the following figure 5.4 and figure 5.5. In the images in figure 5.4 the parameters are set to the values defined in table 4.1 for high quality. In figure 5.5 the parameters are set to the values of low Quality.

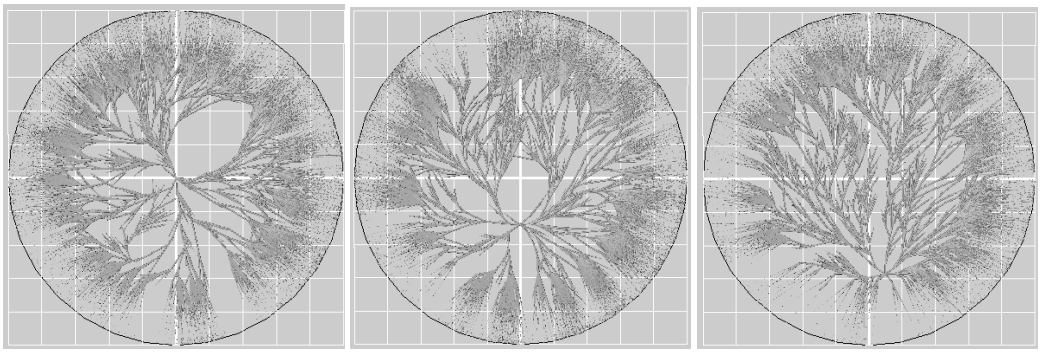

Figure 5.4: Structures with a START_POSITION_DISTANCE of zero (left), 1.4 (middle) and 2.9 (right)

For structures with parameters specified as high quality the influence of the parameter START_POSITION_DISTANCE is not as prominent. In the first image, where the center is positioned at the origin, the symmetry is not very pronounced and empty areas are more common. A start position closer to the edge on the other hand reduces the amount of empty space in the image and the branches appear slightly more curved. The symmetry is most clearly visible in the structure, where the center is placed at a distance of 1.4 length units away from the origin.

For structures, where the remaining parameters are set to represent low quality, the position of the center has a stronger influence. In the images in figure 5.5 one can see that structures, where the center is positioned closer to the origin, show a more equal distribution of the branches around the center compared to the structure, whose center is positioned closer to the edge.

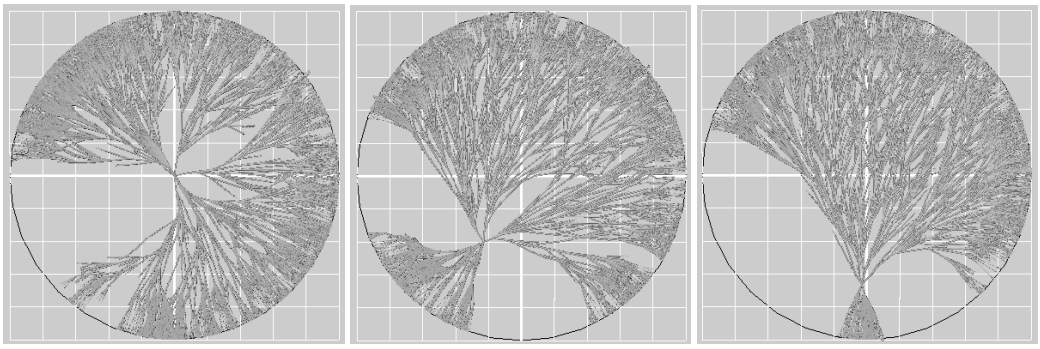

Figure 5.5: Structures with a START_POSITION_DISTANCE of zero (left), 2.3 (middle) and 4.6 (right

INITIAL_BRANCHING_ANGLE

The INITIAL_BRANCHING_ANGLE determines the orientation of the branches emerging from the center. The values used for the simulations shown above in the section "Results" represent the extreme cases. Since the images shown above have differing values for all parameters, additional structures are given in the following figure 5.6. The structure in the image on the left can be compared to the structures given in figure5.2 displaying high quality. The difference is, that in this case the INITIAL_BRANCHING_ANGLE is set to zero, resulting in one side oriented towards the edge. The image on the right has parameters can be compared to the structures of low quality.

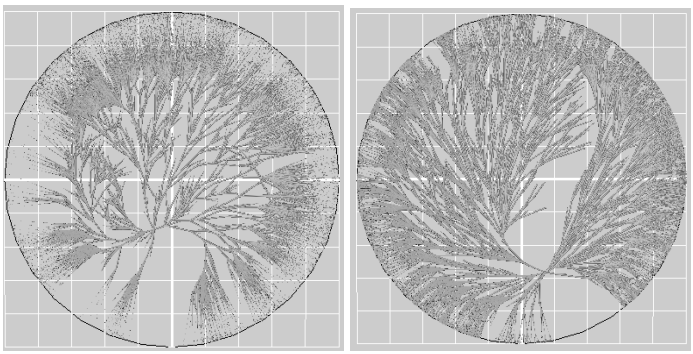

Figure 5.6: Example structures for parameter INITIAL_BRANCHING_ANGLE

In the high quality structures, as seen in the images in figure 5.2, the branches are positioned more beneficial, as both sides from the center have an equal amount of space. In low quality structures, as seen in the images in figure 5.3, one side is oriented towards the edge creating a disadvantage for the growth. The INITIAL_BRANCHING_ANGLE has a great influence on the overall appearance of the structure, as it influences the symmetry by determining the ratio of the area occupied by the two sides. Especially in combination with a high value for the parameter START_POSITION_DISTANCE this parameter has a strong negative influence on the growth.

INITIAL_BRANCHING_STD

The parameter INITIAL_BRANCHING_STD determines the standard deviation used for the angles of branches emerging from the center. Examples of structures with different values can be seen in the images in figure 5.7. The other parameters are set as described for high quality.

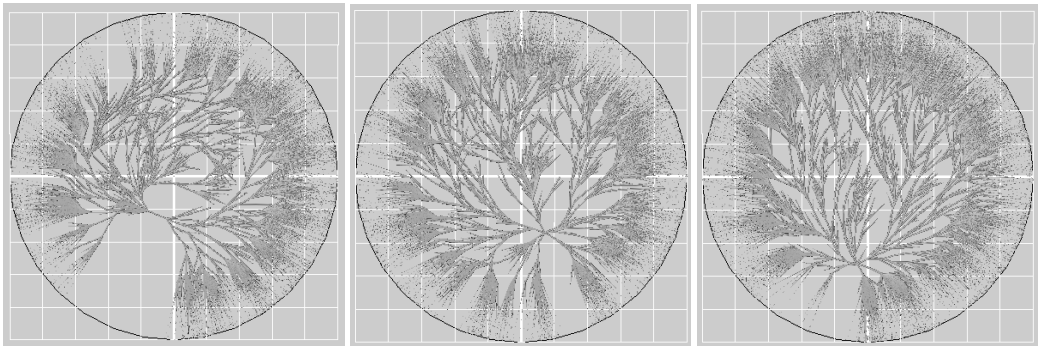

Figure 5.7: Structures with an INITIAL_BRANCHING_STD of zero (left), 45 (middle) and 90 (right)

A lower value increases the visible distinction into two halves, whereas larger values create fuller structures. A difference between the structures is visible, as the image on the left shows a partially empty area consisting of only one internode directly near the center. Still the influence on the overall pattern is not very prominent, as the parameter is only used for the branches emerging from the center. The branching angle in the majority of the structure is determined by the parameter BRANCHING_ANGLE. The influence on the pattern is independent from the other parameters, as its application is limited on this one location.

COLLISIONFREE_ANGLE

The parameter COLLISIONFREE_ANGLE determines how wide the area in front of a node is, where for potential collisions are tested. In the following figure 5.8 images of structures with different values can be seen.

With a value of zero the computation time increased excessively preventing the structure to fully spread, since too many branches had to be simulated. Larger values decrease the amount of collisions, however this prevents the growth of the branches at all. Since the collision avoidance mechanisms tests up to one length unit in front of the current node, multiple small gaps form. In a sparse structure with little competition, the branches grow less curved compared to structures with more branches. The parameter influences the structure quite strongly, as it determines how many branches grow and how crowded the structures appears, which has a major influence on the overall appearance.

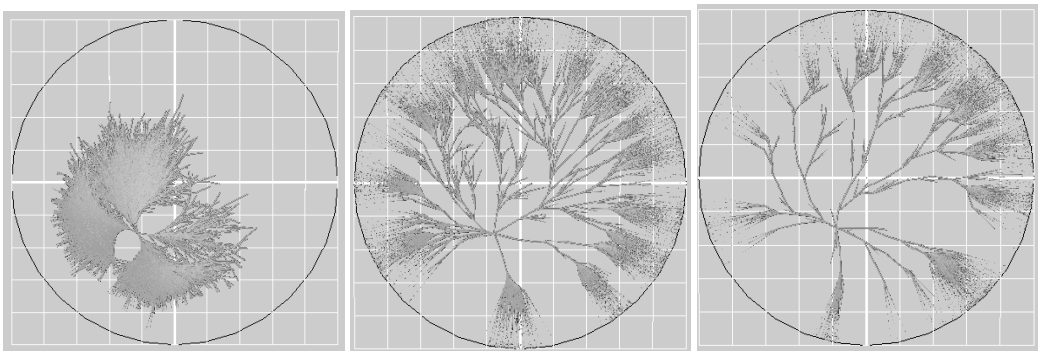

Figure 5.8: Structures with a COLLISIONFREE_ANGLE of zero (left), 15 (middle) and 30 (right)

BENDING_STD

The value of the parameter BENDING_STD is used to calculate a random variable of a normal distribution to set as the angle between the main branch and a newly emerging branch. Multiple structures differing in the standard deviation, can be seen in the following figure 5.9.

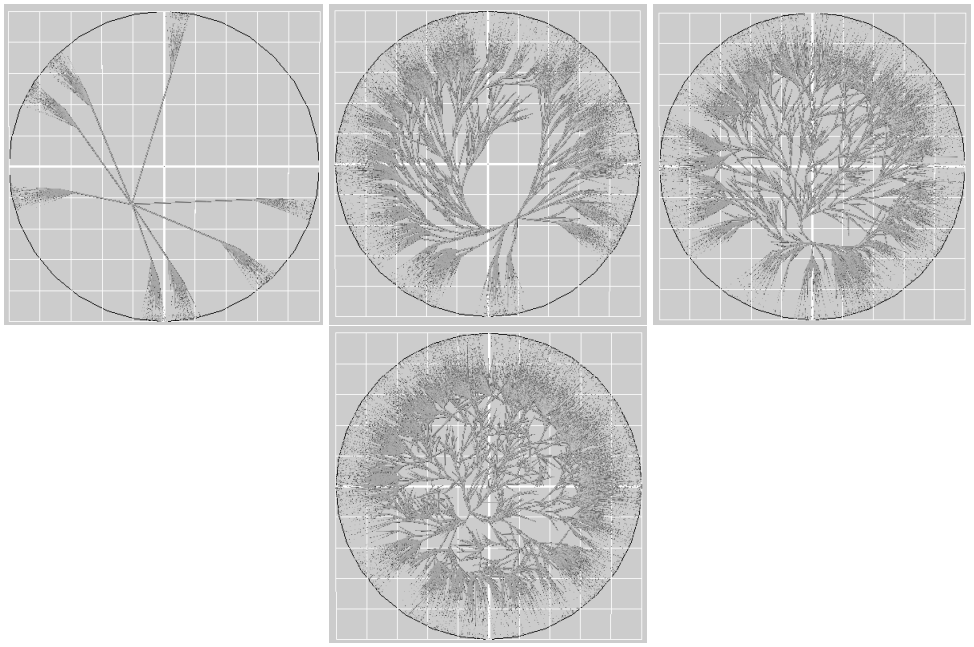

Figure 5.9: Structures with a BENDING_STD of zero (top left), 10 (top middle), 20 (top right) and 30 (bottom)

With a bending angle of zero, new emerging branches are constrained by the collision avoidance mechanism and only the main branches from the center evolve. A low value results in empty areas in the structure, which reduces with larger values. Still the lower values have the advantage,

that the overall pattern looks rather uniform and regular, as the variance is low throughout the whole structure. A higher value creates very sharp turns in the branches, which results in a chaotic appearance. As seen in the images, the extreme values create an overall unnatural structure, but within a slimmer range many values are appropriate. The parameter seems to be independent from other parameters, as structures created with the parameter settings of low quality show the same main aspects. The bending value on itself has a strong impact on the overall appearance, as it influences the amount of branches and the uniformity of the pattern.

LENGTH_MEAN

The parameter LENGTH_MEAN determines the mean of the normal distribution used to select the length of the internodes. Example images of structures with differing values are given in the following figure 5.10. The influence of the parameter on the strucutre is closely connected to the parameter LENGTH_STD.

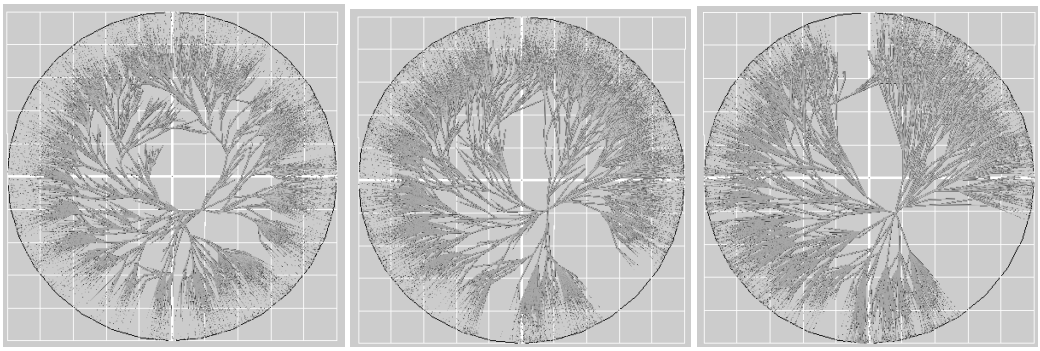

Figure 5.10: Structures with a LENGTH_MEAN of 0.3 (left), 0.6 (middle) and 1 (right)

Small values involve the danger that by using the standard deviation, negative values can be chosen as internode length. A negative value will be set to zero in the simulation to avoid a backwards growing branch, but this still leads to an increase of nodes at a single point making the structure appear more crowded. A larger value increases the amount of gaps in the structure, as no branches can emerge over the length of an internode. Large values also make the structure appear more stiff and less dynamic, as seen in the image in the right. At this point the collision avoidance mechanism only test up to a distance of one length unit in front of the node, so values larger than one would increase the amount of collisions. Compared to other parameters the parameter LENGTH_MEAN has a rather subtle influence on the appearance, but it is still one of the main aspects determining the structure.

LENGTH_STD

The value used for the parameter LENGTH_STD sets the standard deviation of the normal distribution determining the length of the internodes. Structures with different standard deviations can

be seen in the following figure 5.11, the mean is set to 0.3.

Larger values might make the structure appear less uniform, but in general the influence of this parameter on the structure is not very prominent.

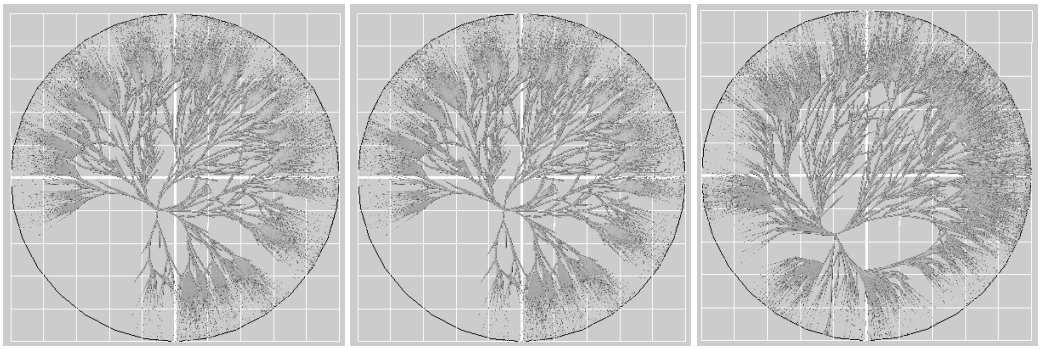

Figure 5.11: Structures with a LENGTH_STD of zero (left), 0.2 (middle) and 0.4 (right)

SURROUNDING_START

The parameter SURROUNDING_START determines how far from the edge the second phase starts. In the following figure 5.12 structures with different values are depicted. The parameter is closely related to the parameter PERIPHERY_START, as they both describe a border within the structure. The parameter PERIPHERY_START is set to 0.25 in all three images.

A larger value decreases the area available for the inner region, which causes in this case a vary imbalanced form. This could be caused by the current setting for the diameter, which emphasizes the contrast between the crowded part of the inner structure and second phase. The images in the middle and on the right show a prominent crowded area between the inner region and the second phase, which is not as clearly visible in the left image. If not for extreme cases as seen in the right image, the influence of the parameter on the structure is rather low.

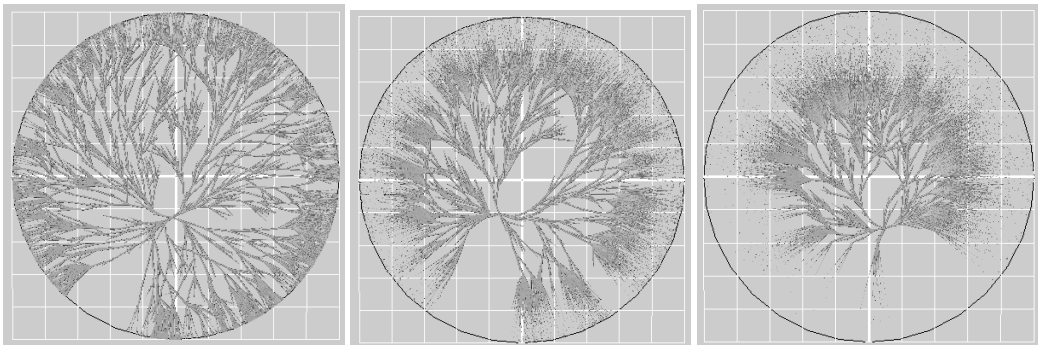

Figure 5.12: Structures with a SURROUNDING_START of one (left), two (middle) and three (right)

PERIPHERY_START

The parameter periphery start determines how far from the edge the third phase starts. It should be set dependent on the parameter SURROUNDING_START, as to not transcend the second phase. Here the parameter SURROUNDING_START is set to 2. Example images of different values for this parameter can be seen in the following figure 5.13.

The influence is marginal as the diameter of the internodes is too small. From this distance and scale no relevant difference is visible in the three images. In structures with parameter set to values responding to the low quality setting, there is also no significant difference visible. The main difference in that case is that the low value for the parameter SURROUNDING_START reduces the area of the second phase. But due to the overall high variance in these images the difference between the values set for this parameter is not very prominent.

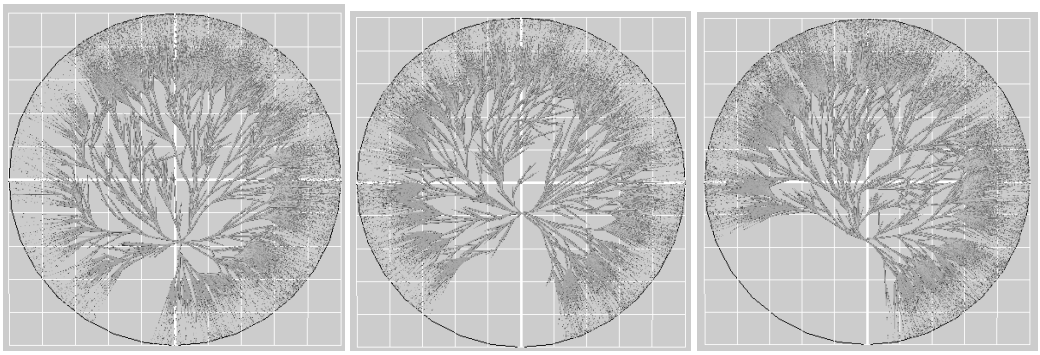

Figure 5.13: Structures with a PERIPHERY_START of 0.5 (left), 0.75 (middle) and one (right)

5.3 Limitations

The development and implementation of a simulation model always holds difficulties. Some problems that could not be solved sufficiently due to a lack of time or knowledge are explained in the following section.

Collisions

One of the main problems in the implementation was to avoid collisions. It is known that collision avoidance is a complex problem for rule-based languages. The following approaches were implemented in order to prevent collisions.

The first idea was to use the distance function in order to avoid any two objects to be closer to one another than a specified threshold. The main difficulty with this was that an ancestor node or nodes growing in different direction, like the ones directly around the center, would need to be excluded from the rule, since their are naturally close without any danger of colliding. Otherwise

they inhibit the growth of the structure even though they could not cause a collision. Since multiple different relations would have to be excluded, it was too complicated to formulate as a rule.

Another aspect of this problem is that the distance function compares the start position of an object, thus for cylinders collisions near the end position will not be identified. Therefore the threshold of the distance function would have to be increased to entail the end position of the internode, which on the other hand increases the amount of nodes, which are not of interest.

Even if it were possible to formulate a query excluding special cases, the growth would be stunted on all nodes involved, thus leaving larger empty areas in the structure.

These are some of the problems which have to be taken into account making the distance function not easily usable or effective in collision avoidance.

A mechanism, which is currently implemented as a part of the collision avoidance functionality, was presented by Ole Kniemeyer in his dissertation [16]. He uses a cone as an angle of vision and only calculates the distance for nodes within this field of view. This way no unrelated nodes, like ancestor nodes or nodes on the opposite site of the growth direction, influence the mechanism. A remaining problem is that only the start position of cylinders are compared.

Another solution to prevent collisions previously developed is given in the GroIMP example *smart line*. It uses the class *AvoidIntersection*, which simulates something like an antenna using rays emitted in a half circle from the current node. For these rays collisions are determined and evaluated depending on how close to the node the collisions occur. The growth direction of the node is then chosen as the direction of the least collisions in the rays. The class can be adapted using parameters like the number of rays, the opening angle, prediction length or strength of direction change. The main problem of this approach is that even with adapted parameter the simulation of the amount of rays for each node in the structure is increasing computation time to an inefficient level.

Parametrisation

On one hand the implementation posed problems concerning the definition of specific rules, like collision avoidance or how to achieve a natural amount of curvature in the branches. On the other hand it was very difficult to put into parameters what makes the pattern so distinct.

Since the evaluation of the crystallization images is done visually, the idea emerged to create groups of sample structures with differing parameters to determine the most realistic ones visually. This would automatically create a set of parameters, which could be used to statistically compare the structures. In order to create a large amount of sample structures, a sensitivity analysis using GroIMP in headless mode would have been used. This is not possible, since the GroIMP headless mode does not create a visual output, but only returns alphanumeric output.

Initially the aim was to use the GroIMP headless mode to run a sensitivity analysis, which

unfortunately is not possible as the headless mode does not create a visual output, but only return alphanumeric values.

Originally the simplification models were created with the aim of using them as a basis to run queries collecting parameter values for the variable model. Although it is clear that single feature parameters, which would be achieved by the queries, are not the most useful in the description of crystallization structures. Additionally the amount of simplifications and the level of detail do not suffice for data collection.

Extract

Originally the scope included a comparison between extracts additional to the difference in quality resulting in four categories. For this, structures created from carrot of high and low quality would have been used. Apart from the lack of time, these additional structures present the difficulty of finding characteristic differences between structures grown from carrot extract and milk.

5.4 Alternative Approaches

The rule-based modelling platform GroIMP provides many different possibilities and features to implement simulation models. Since an implementation of biocrystallization structures has not been tested in a rule-based language yet, it was not directly foreseeable which approaches do hold the best result. Due to lack of time, knowledge or other reasons not all ideas could be implemented and tested. Hence a short overview of alternative approaches is given in the following.

Statistical Basis

A different approach is, to create a model based on statistical data measured on the crystallisation structures. In a previous study different parameters, for example the length and diameter of internodes, were measured and analysed in regard to the ROI [11]. This data could have been used as reference to set the attributes of the objects. This approach was not implemented since later studies show, that these single feature measurements are inferior in the characterization of the structure compared to criteria describing the patterns as a whole. This has not bee implemented since the model should depict characteristic morphology, which could not be achieved using only statistical values.

Branching Order

A main aspect of dendritic growth is the distinction of the branches in orders. This could have been emphasized in the implementation by dividing the growth into phases/steps. In a stepwise growth the additional branches are placed onto the existing branches. This could create a distinct and uniform pattern of larger and smaller branches. It might increase the regularity of the structure, when the main branches of first order would be salient and uniform.

Collisions

Two possible methods to handle collisions, which were not implemented and tested, are given in the following.

The first approach uses additional small auxiliary spheres positioned on the outside of the internodes. These small scale objects are then used to calculate the distance to other small objects on a different internode. That way the distance between the whole objects of the internodes can be computed instead of comparing the starpositions of the internodes. The auxiliary objects do not need to be visible, as to not influence the overall appearance of the structure. Still the method yields problems, for example to choose the distribution of the auxiliary objects and the increase in calculation time. Also it might be difficult to determine the appropriate distance between objects while considering the growth direction of the internodes.

A second method to avoid collisions unique for the replacement rules is to divide the growth of the structure into two parts. In the first step each nodes grows with no restriction, thus allowing collisions. In the second step all collisions are removed. This method takes advantage of the applicability of replacement rules. The apparent difficulty in removing collisions is to identify them. For this, the collision volume of two cylinders in three dimensional space has to be calculated, for which, to my knowledge, no functions are implemented in GroIMP.

Reduced Dendritic Structure

Another idea would have been to not use nodes and internodes but instead only place internodes in a general crystallization shape. The internodes would not continue directly after one another but be placed close to each other. This would come closer to the form of the branches in the images, which actually resemble single needles more than branches. It poses the risk of not enhancing the overall structure but on the opposite creating a more chaotic pattern.

Concentration Map

One last approach was to use a kind of invisible density map in the background, which would simulate the $CuCl_2$ concentration on the plate. Growth of the structure would decrease the concentration in that area and the structure would grow in the direction with higher concentration, thus avoiding collisions. It might be a good approach to create a distinction of larger and smaller branches, if the growth was executed in multiple steps. In each iteration the diameter of the internodes and the threshold for growth could be adapted, creating an order in the structure. But implementing the behaviour of the $CuCl_2.2H_2O$ solution goes beyond the scope of a morphological model. Especially since the chemical reactions of the crystallisation process are not clear. Additionally the depiction of different qualities, where again the chemical basics are not known, forms another problem. Also it might be too computational expensive since the map in the background would have to have a high level of detail and it would have to be updated in every step.

Bibliography

[1] J. Kahl, G.-J. van der Burgt, D. Kusche, S. Bügel, N. Busscher, E. Hallmann, U. Kretzschmar, A. Ploeger, E. Rembialkowska, and M. Huber, "Organic Food Claims in Europe," *Food Technology*, vol. 64, 03 2010.

[2] N. Busscher, J. Kahl, P. Doesburg, G. Mergardt, and A. Ploeger, "Evaporation influences on the crystallization of an aqueous dihydrate cupric chloride solution with additives," *Journal of Colloid and Interface Science*, vol. 344, no. 2, pp. 556–562, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021979709016075

[3] M. Szulc, J. Kahl, N. Busscher, G. Mergardt, P. Doesburg, and A. Ploeger, "Discrimination between organically and conventionally grown winter wheat farm pair samples using the copper chloride crystallisation method in combination with computerised image analysis," *Computers and Electronics in Agriculture*, vol. 74, no. 2, pp. 218–222, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169910001456

[4] J. Kahl, N. Busscher, P. Doesburg, G. Mergard, H. Machteld, and A. Ploeger, "First tests of standardized biocrystallization on milk and milk products," *European Food Research and Technology*, vol. 229, pp. 175–178, 05 2009.

[5] D. E. Donohue and G. A. Ascoli, "A comparative computer simulation of dendritic morphology," *PLoS Computational Biology*, vol. 4, no. 5, p. e1000089, June 2008.

[6] T. Wang, J. Jin, X. Zheng, and I. Ohnaka, "A novel simulation method for the prediction of dendritic grain structures in solidification," *International Journal of Cast Metals Research*, pp. 231–236, 01 2003.

[7] R. Kobayashi, "Modeling and numerical simulations of dendritic crystal growth," *Physica D: Nonlinear Phenomena*, vol. 63, pp. 410–423, 03 1993.

[8] Z. Gao, C. Zhu, M. Qi, C. Wang, Y. Wang, and B. Zhao, "Multi-phase field model simulation based on mpi+openmp parallel: Evolution of seaweed and dendritic structure in directional solidification," *AIP Advances*, vol. 12, 03 2022.

[9] J. Kahl, N. Busscher, W. Hoffmann, G. Mergardt, I. Clawin-Raedecker, and A. Ploeger, "A novel approach for differentiation of milk fractions and polyvinylpyrrolidone with different molecular weight by patterns derived from cupric chloride crystallization with additives," *Anal. Methods*, vol. 6, pp. 3173–3176, 2014. [Online]. Available: http://dx.doi.org/10.1039/C3AY41568F

[10] N. Busscher, J. Kahl, and A. Ploeger, "From needles to pattern in food quality determination," *Journal of the Science of Food and Agriculture*, vol. 94, 10 2014.

[11] P. Doesburg and A. F. Nierop, "Development of a structure analysis algorithm on structures from $CuCl_2.2H_2O$ crystallization with agricultural products," *Computers and Electronics in Agriculture*, vol. 90, pp. 63–67, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169912002608

[12] M. Huber, J.-O. Andersen, J. Kahl, N. Busscher, P. Doesburg, G. Mergardt, S. Kretschmer, A. Zalecka, A. Meelursarn, A. Ploeger, D. Nierop, and L. Vijver, "Standardization and validation of the visual evaluation of biocrystallizations," *Biological Agriculture & Horticulture*, vol. 27, pp. 25–40, 01 2010.

[13] P. Doesburg, J. Fritz, M. Athmann, R. Bornhütter, N. Busscher, U. Geier, G. Mergardt, and C. Scherr, "Kinesthetic engagement in gestalt evaluation outscores analytical 'atomic feature' evaluation in perceiving aging in crystallization images of agricultural products," *PLOS ONE*, vol. 16, pp. 1–17, 03 2021. [Online]. Available: https://doi.org/10.1371/journal.pone.0248124

[14] P. Doesburg, M. Huber, J.-O. Andersen, M. Athmann, G. Bie, J. Fritz, U. Geier, J. Hoekman, J. Kahl, G. Mergardt, and N. Busscher, "Standardization and performance of a visual gestalt evaluation of biocrystallization patterns reflecting ripening and decomposition processes in food samples," *Biological Agriculture and Horticulture*, vol. 31, 12 2014.

[15] O. Kniemeyer, R. Hemmerling, and W. Kurth. GroIMP. [Online]. Available: http://www.grogra.de/

[16] O. Kniemeyer, "Design and implementation of a graph grammar based language for functional-structural plant modelling," Ph.D. dissertation, University of Technology at Cottbus, 02 2008.

[17] R. Hemmerling, O. Kniemeyer, D. Lanwert, W. Kurth, and G. Buck-Sorlin, "The rule-based language XL and the modelling environment GroIMP illustrated with simulated tree competition," *Functional Plant Biology 35 (2008) 9-10*, vol. 35, 12 2008.

[18] G. Buck-Sorlin, O. Kniemeyer, and W. Kurth, "A grammar-based model of barley including virtual breeding, genetic control and a hormonal metabolic network," *Functional-Structural Plant Modelling in Crop Production*, 01 2007.

[19] W. Kurth, "Specification of morphological models with L-systems and relational growth grammars," *Image – Journal of Interdisciplinary Image Science*, vol. 5, 01 2007.

[20] P. Prusinkiewicz and A. Lindenmayer, "The algorithmic beauty of plants." Springer, 1990.

[21] O. Kniemeyer, G. H. Buck-Sorlin, and W. Kurth, "Rule-based modelling with the XL/GroIMP software," *The Logic of Artificial Life*, 2004.

[22] O. Kniemeyer, G. Buck-Sorlin, and W. Kurth, "Representation of Genotype and Phenotype in a Coherent Framework Based on Extended L-Systems," *Lecture Notes in Artificial Intelligence*, vol. 2801, pp. 625–634, 09 2003.

[23] J. Fritz, M. Athmann, J.-O. Andersen, P. Doesburg, U. Geier, and G. Mergardt, "Advanced panel training on visual gestalt evaluation of biocrystallization images: ranking wheat samples from different extract decomposition stages and different production systems," *Biological Agriculture & Horticulture*, vol. 35, pp. 1–12, 07 2018.