

Complex Community and Destructive Craft

Principia Textilica : Project Documentation

Felicitas Höbelt

April 2, 2015

1 Principia Textilica

1.1 Context

In the course "Principia Textilica" we explore intersections of computational algorithms and textile craft, including pattern generation algorithms, weaving techniques, knot theory, hands-on experience with textile machines, history lessons and more. For the final project each student is encouraged to find a subject to implement some of the methods and techniques we see during the course.

1.2 Motivation

I study Computer Science and Media, which means my background is more algorithmic and technical than art-related. Nevertheless, I am familiar with a few textile techniques such as knotting (Macramé), weaving, sewing, felting and embroidery.

For inspiration I look at two topics that have fascinated me for years: Modeling "Living" Things (plants/fungi/insects/animals) and complexity.

Complexity is often mistaken for complicatedness because the result looks complicated. However, most systems that produce complex results are very simple. It consists of a certain number of elements and a few elementary rules which are applied to those elements.

My goal is to use both, complexity and modeling simple "life", to create a textile pattern.

2 Idea

2.1 The Pond/Fish Tank

I will use a swarm model, which means I have a number of individuals which are controlled by a set of rules concerning themselves, their dynamic environment (e.g. the other members of the swarm) and their static environment (e.g. boundaries of the tank). The metaphor I choose is a school of fish moving around in a pond or glass tank.

The desired pattern will be generated by the movements of the individuals. I want to start with the rules and successively adapt them depending on the outcome. I do not want to settle for a specific textile technique at this point, but I know I want to assign a thread to each of the individuals. With this premise in mind, favourable textile techniques are embroidery and knotting.

2.2 Rules

My fish program is in 2D and has two main elements: the tank and the fish.

The tank started out as rectangular, but I find a circle to be more aesthetically pleasing as well as more fitting for an embroidery hoop. The circle embodies the simple inside/outside world that I need.

The basic swarm rules below share some similarities with the "Boids" program by Craig Reynolds¹. He developed an agent-based algorithm to simulate the flocking behaviour of birds.

There is no randomness in my program. The behaviour of the fish is essentially determined by three basic principles:

Company Each fish will follow other fish.

Privacy Each fish needs a minimum of private space.

Boundary Each fish will stay inside the tank.

as well as two optional principles:

Security Each (non-predatory) fish will avoid any predators.

Mortality Each fish can have children. Fish can die of childbirth.

3 Implementation

For implementing my idea I use Processing 2.0², because I am already familiar with Java. Also Processing is one of the tools we use during the course, so all participants know how to look into my code and adapt it if they want to.

This section will focus more on algorithms and data flow rather than implementation details. There are four ways of running the program represented by the combination of optional mechanisms on top of the basic ones:

1. use both Security and Mortality
2. use only Security

¹<http://www.red3d.com/cwr/boids/>

²<https://processing.org/>

3. use only Mortality

4. use none of them

It is worth noting that the code still includes all methods and parameters that I do not use for the textile piece or the parameter variations, but I will restrict the explanations to the version I actually use (3. only Mortality). In this version each fish will have two children when it dies. The resulting pattern is made of straight lines connecting parents with their children. The movement of the fish is influenced by all living fish as well as all dead fish. I start with one fish and stop at 6 or 7 generations.

3.1 Algorithm

Class	responsible for
class Fish	<ul style="list-style-type: none">- parameters (id, color, speed, position, direction...)- updating the state (position, direction)- drawing (body, connections to other fish, traces for the traces-visible-mode)- computing new direction regarding tank borders and neighboring (visible) fish- contribute to log file
class Tank	<ul style="list-style-type: none">- updating and saving images (traces and frequency)- drawing the tank
public methods and variables	<ul style="list-style-type: none">- keeping track of all elements (fish, tank, variables for drawing modes)- initial setup- drawing, includes animation and timing- receiving key presses (for saving or pausing)- log file (.txt)- vector operations (e.g. normalizing, computing the scalar product, turning, computing a reflection vector)

3.2 Exporting the Values

To be able to extract all important positions and values from the program several logging methods are implemented. The program produces a simple text-file, where the parameter values are noted as well as the positions of the fish (original coordinates plus coordinates scaled to fit the textile application), their id and their children. The tank class produces an image where the positions of the fish are noted pixelwise, the image shows their complete movement. I also export an SVG-file with a straight line connecting the end position of each fish with their children. This export can be done with additional text output directly within the SVG-file (the id of each fish above its "grave").

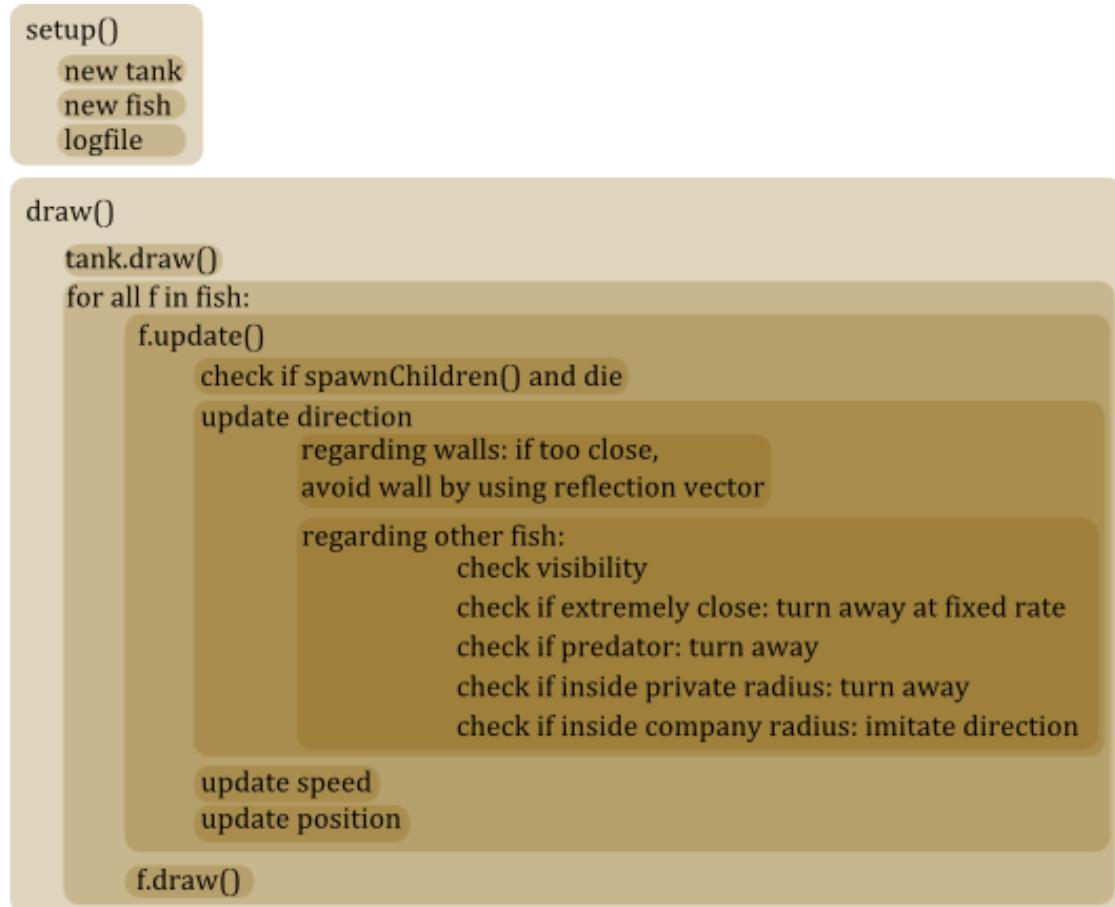


Figure 1: Essential structure of the program

```

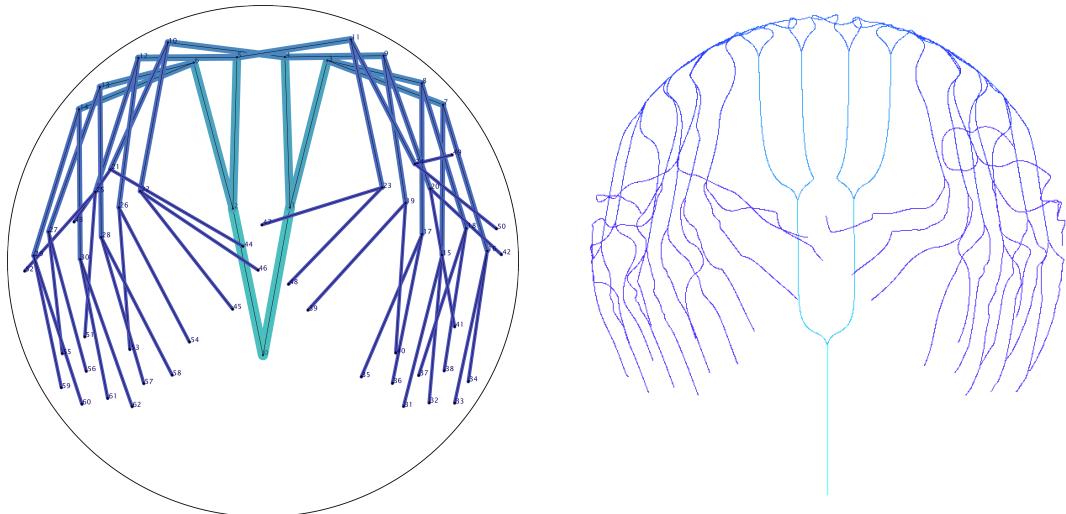
tank cx,cy,radius: 300.0 300.0 300.0
starter fish: 1
start dir: 0.0 -1.0
start pos: 300.0, 300.0
wall/neighbor 0.9/0.1
maxGenerations 6
maxChildren 2

spawnTime 6000ms
spawnAnglesDegree 135.0 -135.0
turnSpeed 0.011/ms
private radius / company radius 30.0 / 70.0

fish:
0 g:0 born 300.0 590.0 : 140 275 c:
0 g:0 died 300.0 410.82532 : 140 192 c: 1 2
1 g:1 died 331.64142 237.63405 : 155 111 c: 3 4
2 g:1 died 264.66016 238.89192 : 124 111 c: 5 6
3 g:2 died 375.58804 65.089584 : 175 30 c: 7 8
...

```

Figure 2: The text-based log file consists of all important parameters as well as the coordinates of each fish, g: denotes the generation while c: denotes the ids of its descendants.



18-3-15-37-43

Figure 3: Connection lines between (dead) fish of 6 generations

Figure 4: Traces of the movement of the fish while living

3.3 Parameters and Variations

There are several more, but some of the most important fish parameters and properties are:

privateRadius If another fish is within this distance (in pixel) the current fish moves away from it.

companyRadius If another fish is within this distance (in pixel) the current fish imitates the other fish's movement (linear interpolation between current direction vector and target direction vector).

interpolationSpeed (related to speed of turns) This value is the number of milliseconds which are needed to reach the target vector with linear interpolation. Since interpolation is mainly used when a fish changes its direction vector the interpolationSpeed is inversely related to the speed of turns.

timetoSpawn This is the life span of one fish (in milliseconds), in the moment of dying it will generate two new fish.

spawnAnglesDegree This array contains the angles (in degrees) for the initial direction of each child (relative to the parent's direction).

When varying one parameter the others are set to a default value to cleanly separate the effects of the change. The "baseline" setting with all parameters set to default

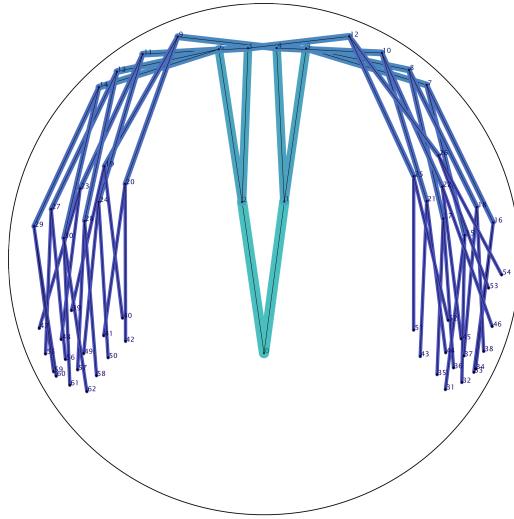
is included in each of the variation series. The natural order of the parameter sizes determines its placement within the series (ordered by increasing values). Table 1 shows the used integer values.

Parameters	Default value	Variation
privateRadius	30	20 30 40 50
companyRadius	70	50 70 90 120
interpolationSpeed	100	50 100 150 200 250 350
timeToSpawn	6000	3000 4000 5000 6000 8000
spawnAnglesDegree (+/-)	135	0 20 45 90 135 180

Table 1: Used parameter values

The following images show the different outcomes for variations of the parameters. On the left you see the connections between related fish, on the right there are their movement traces (actual paths).

3.3.1 Variations of "privateRadius"



18-3-15-33-11

Figure 5: $\text{privateRadius} = 20$

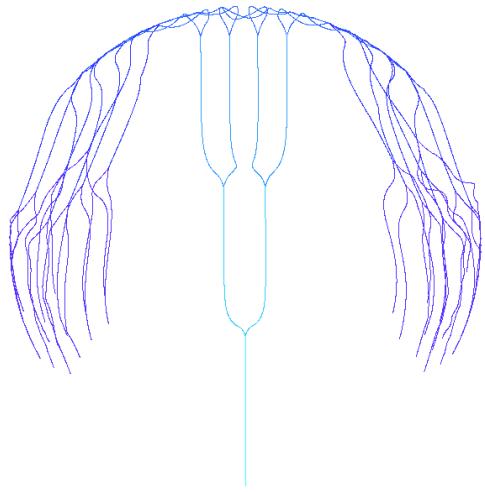
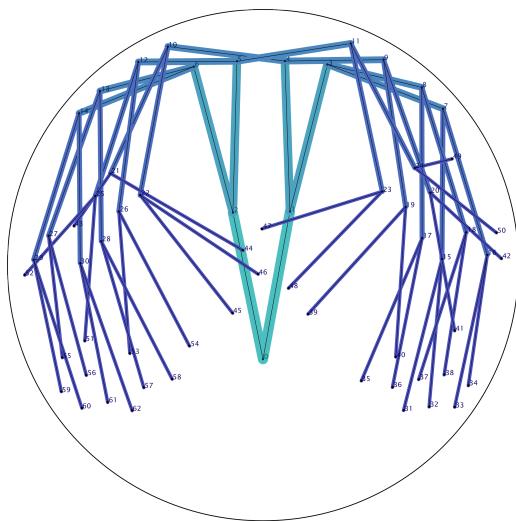


Figure 6: $\text{privateRadius} = 20$



18-3-15-37-43

Figure 7: privateRadius = 30 (default)

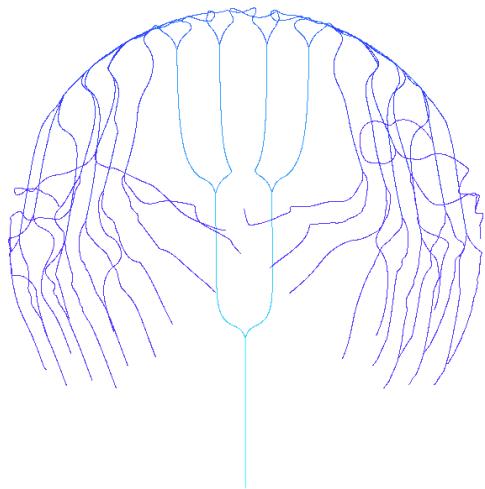
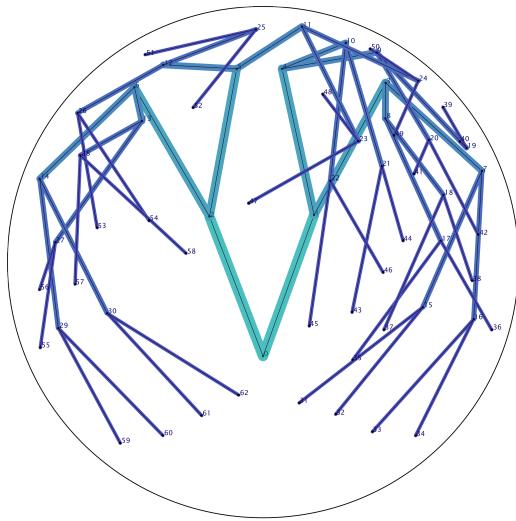


Figure 8: privateRadius = 30 (default)



18-3-15-34-9

Figure 9: privateRadius = 40

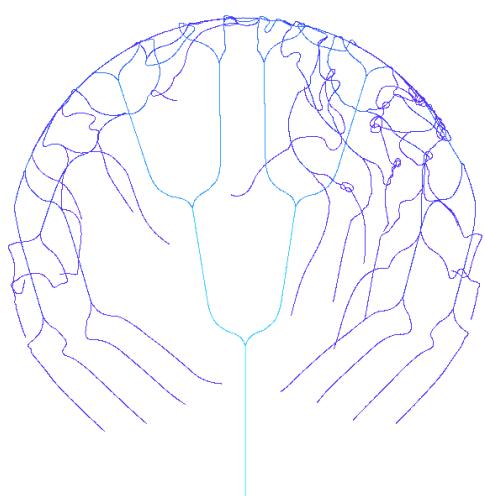
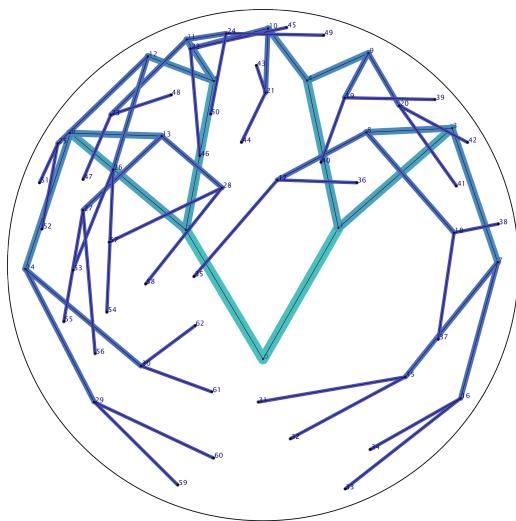


Figure 10: privateRadius = 40



18-3-15-35-1

Figure 11: privateRadius = 50

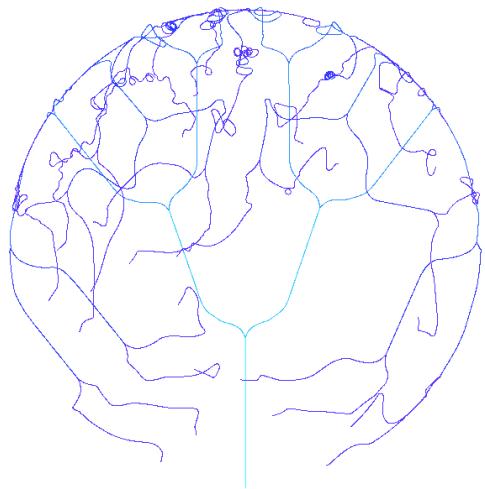
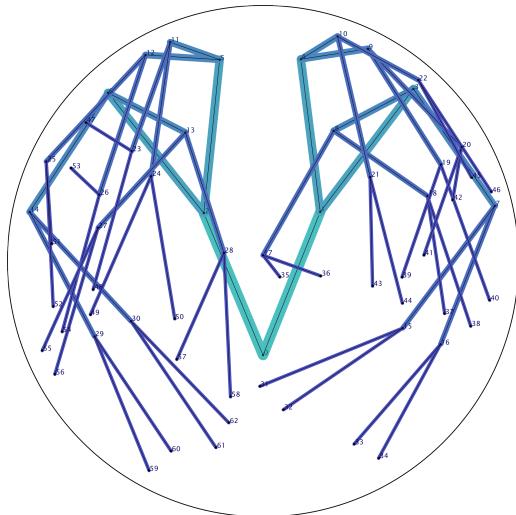


Figure 12: privateRadius = 50

3.3.2 Variations of "companyRadius"



18-3-15-30-2

Figure 13: companyRadius = 50

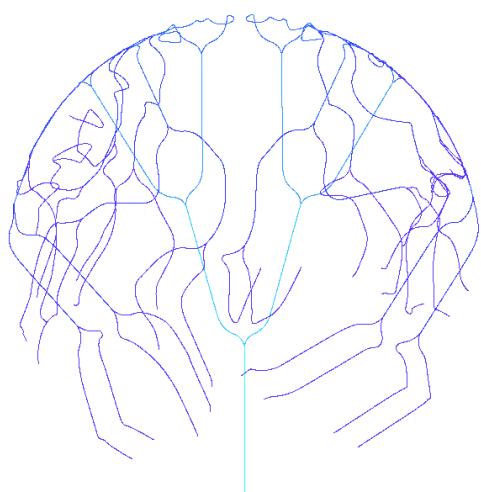
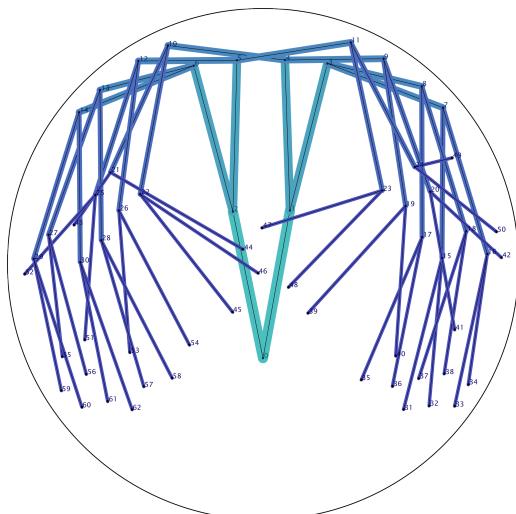


Figure 14: companyRadius = 50



18-3-15-37-43

Figure 15: companyRadius = 70 (default)

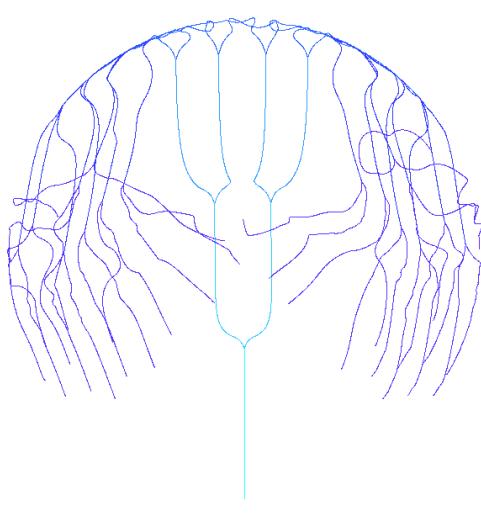
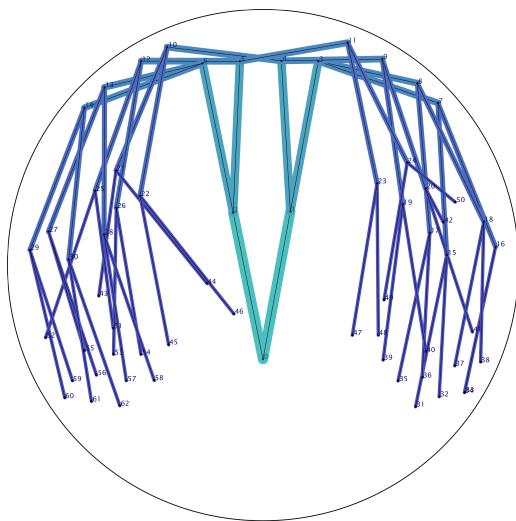


Figure 16: companyRadius = 70 (default)



18-3-15-30-53

Figure 17: companyRadius = 90

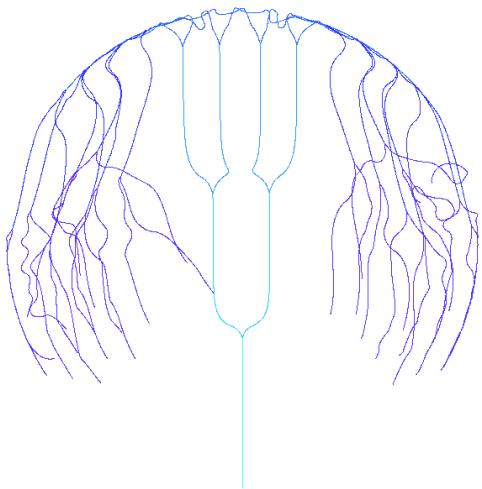
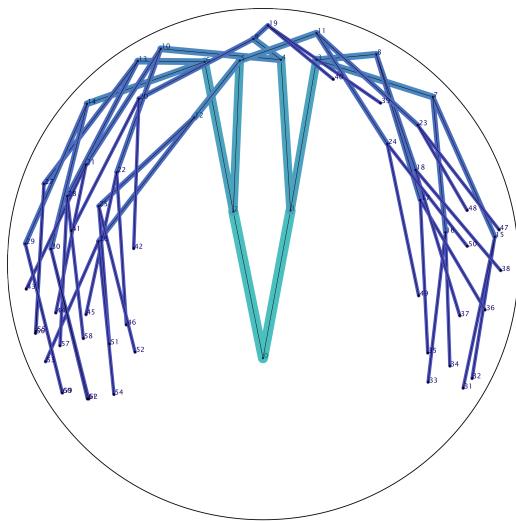


Figure 18: companyRadius = 90



18-3-15-31-57

Figure 19: companyRadius = 120

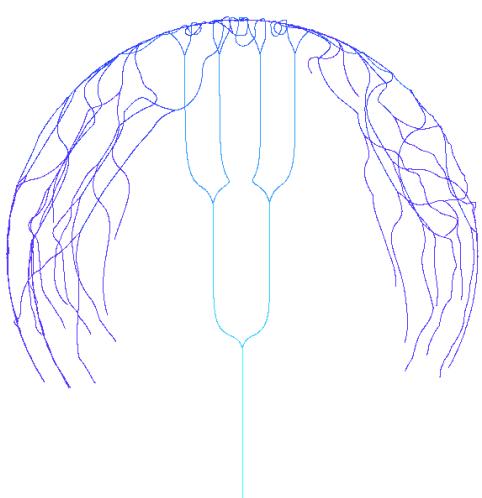
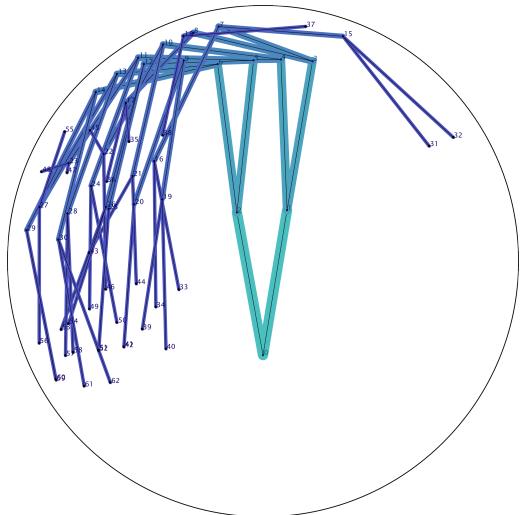


Figure 20: companyRadius = 120

3.3.3 Variations of "interpolationSpeed"



18-3-15-24-36

Figure 21: interpolationSpeed = 50

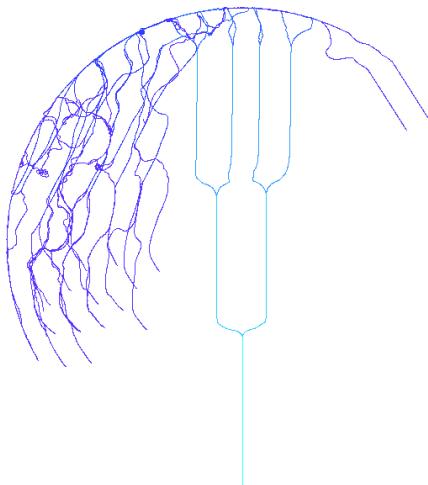
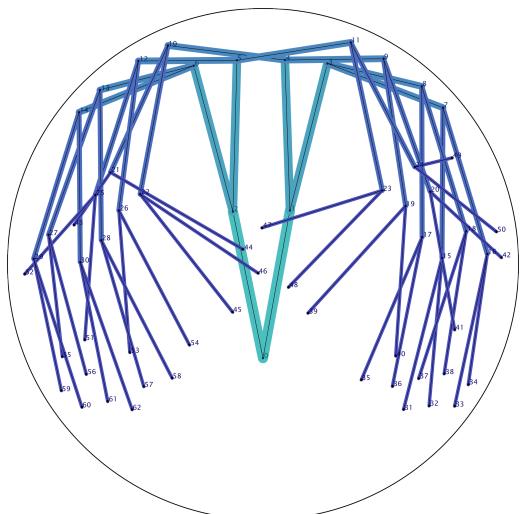


Figure 22: interpolationSpeed = 50



18-3-15-37-43

Figure 23: interpolationSpeed = 100 (de-
fault)

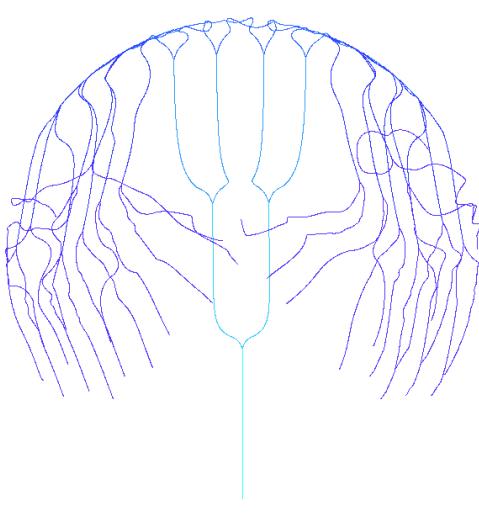
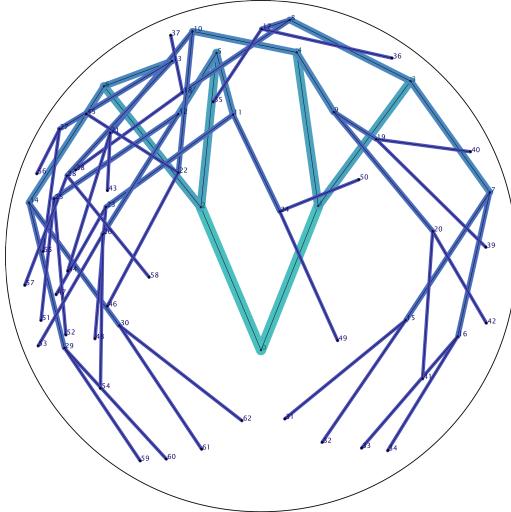


Figure 24: interpolationSpeed = 100 (de-
fault)



18-3-15-25-27

Figure 25: interpolationSpeed = 150

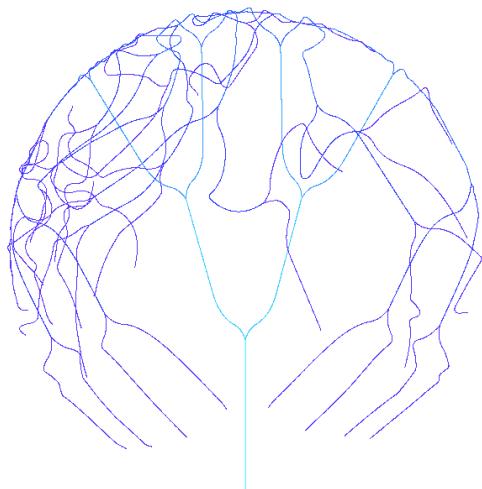
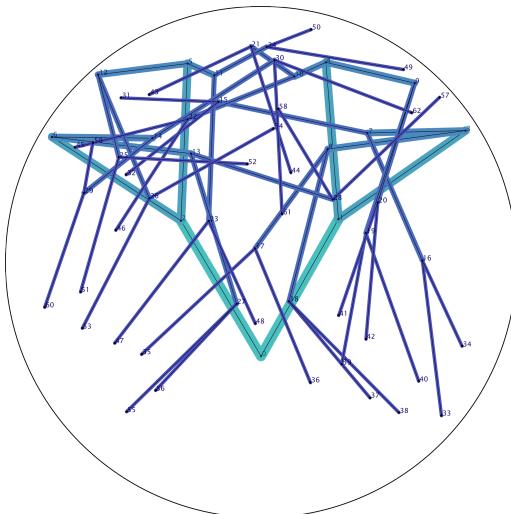


Figure 26: interpolationSpeed = 150



18-3-15-26-31

Figure 27: interpolationSpeed = 200

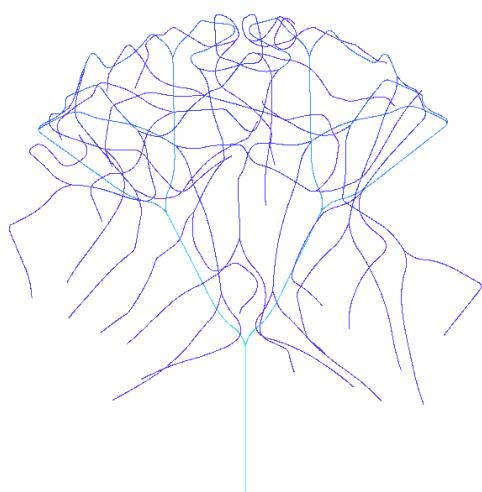
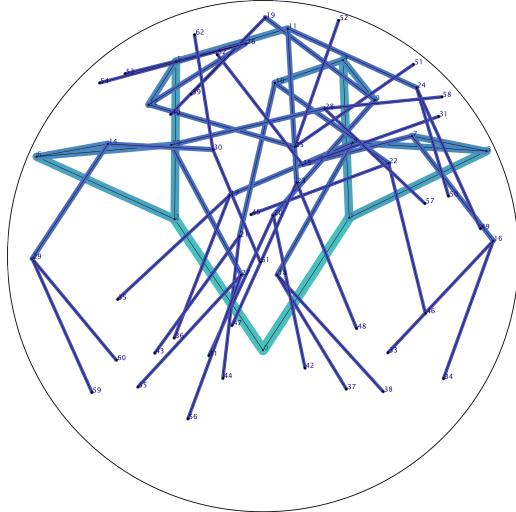


Figure 28: interpolationSpeed = 200



18-3-15-27-26

Figure 29: interpolationSpeed = 250

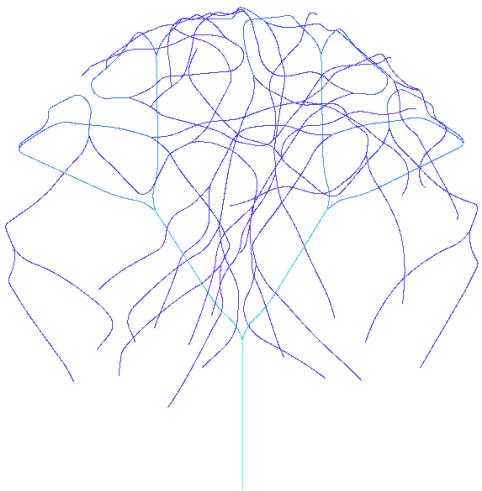
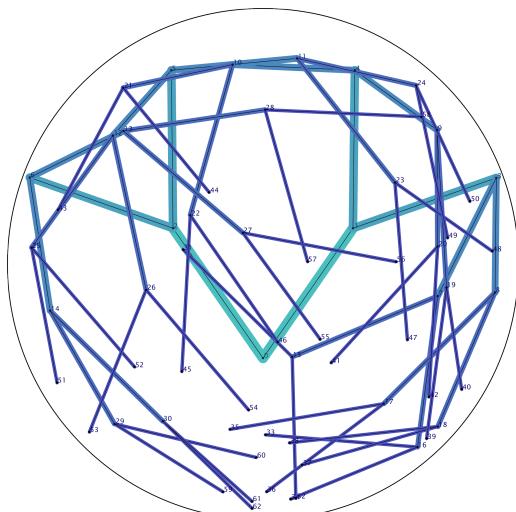


Figure 30: interpolationSpeed = 250



18-3-15-28-27

Figure 31: interpolationSpeed = 350

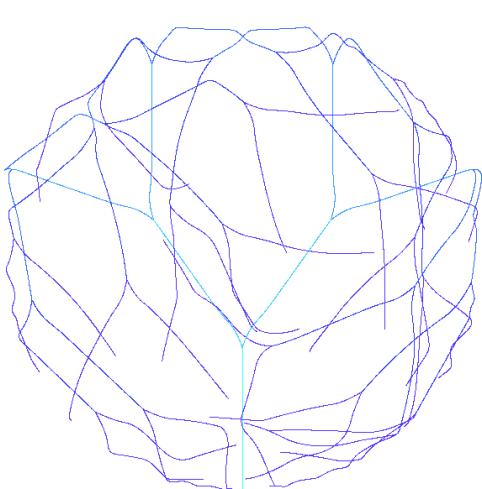
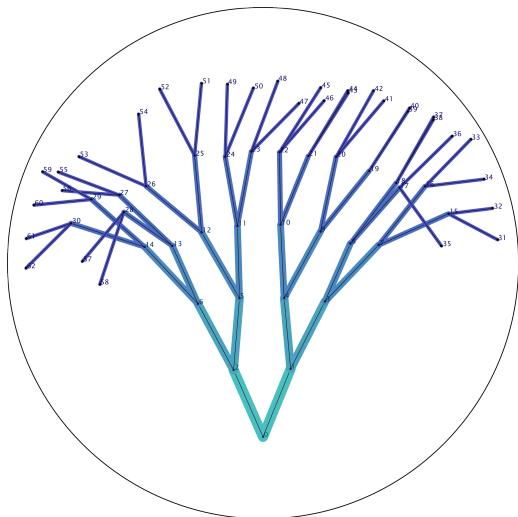


Figure 32: interpolationSpeed = 350

3.3.4 Variations of "timetoSpawn"



18-3-15-21-0

Figure 33: timetoSpawn = 3000

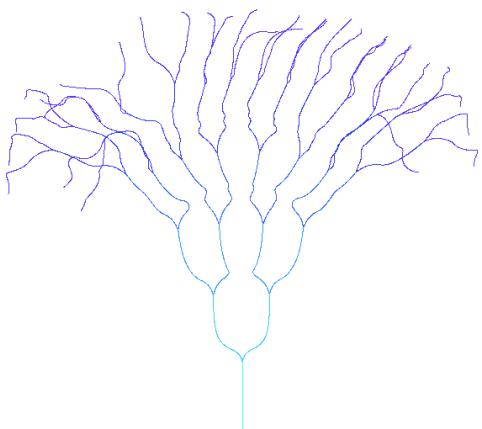
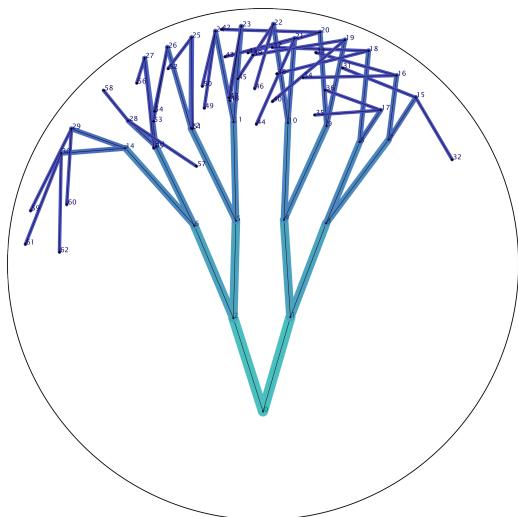


Figure 34: timetoSpawn = 3000



18-3-15-21-37

Figure 35: timetoSpawn = 4000

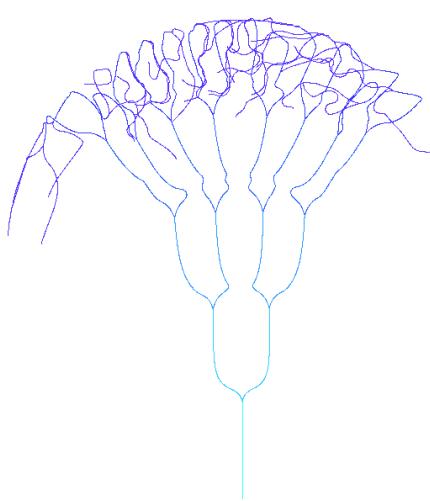
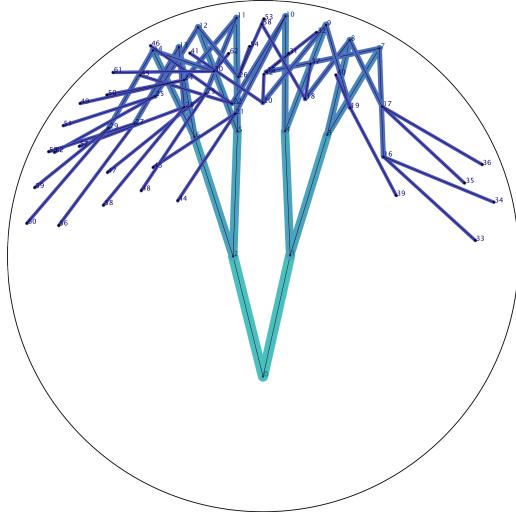


Figure 36: timetoSpawn = 4000



18-3-15-22-17

Figure 37: timetoSpawn = 5000

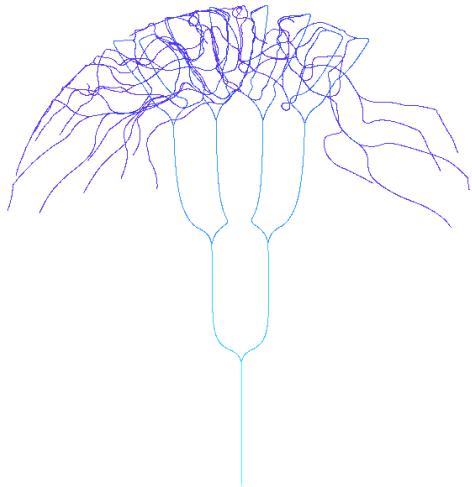
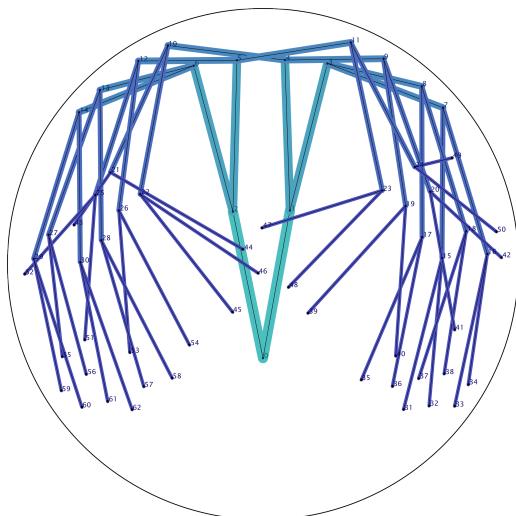


Figure 38: timetoSpawn = 5000



18-3-15-37-43

Figure 39: timetoSpawn = 6000 (default)

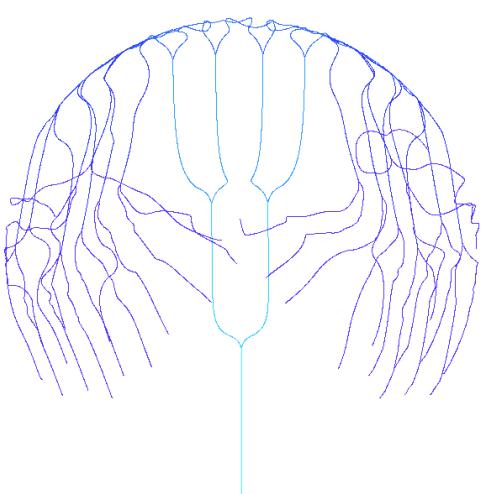
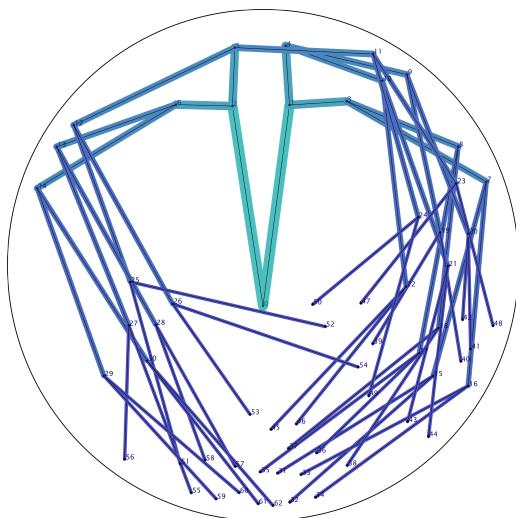


Figure 40: timetoSpawn = 6000 (default)



18-3-15-23-3

Figure 41: timetoSpawn = 8000

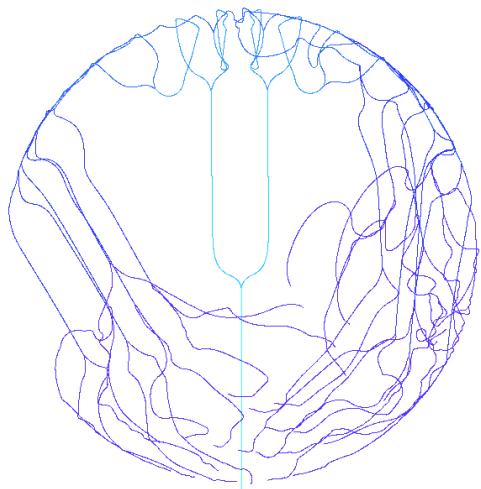


Figure 42: timetoSpawn = 8000

3.3.5 Variations of "spawnAnglesDegree"

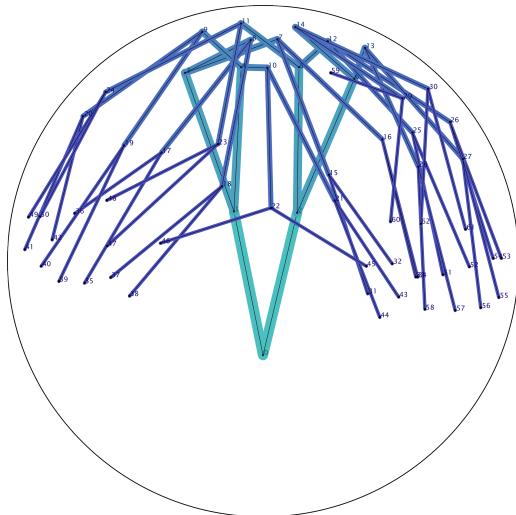


Figure 43: spawnAnglesDegree = +/-0

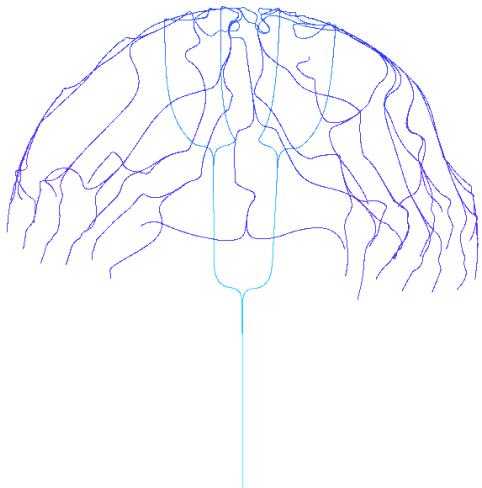


Figure 44: spawnAnglesDegree = +/-0

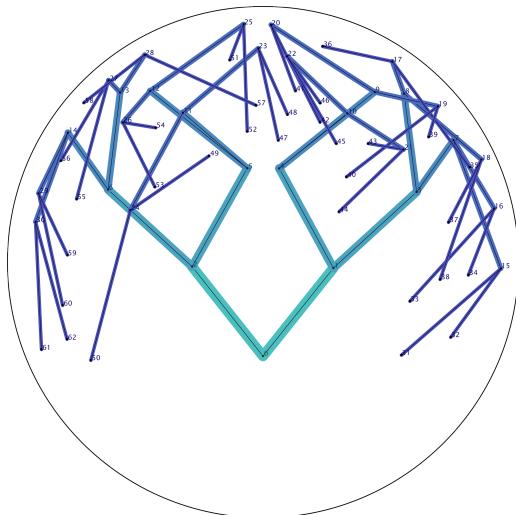


Figure 45: spawnAnglesDegree = +/-20

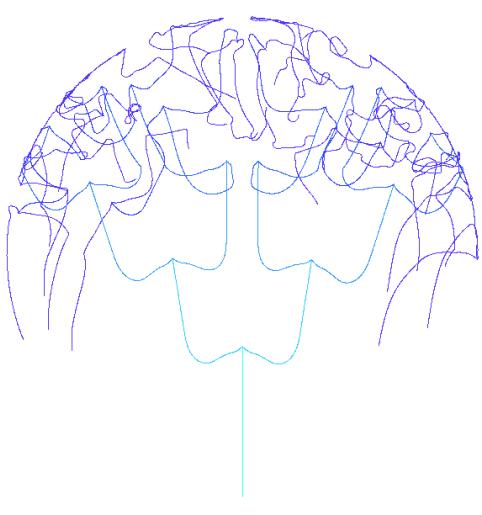
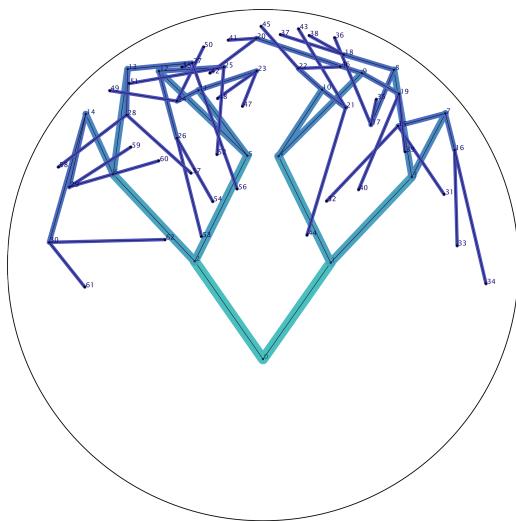


Figure 46: spawnAnglesDegree = +/-20



18-3-15-15-25

Figure 47: spawnAnglesDegree = +/-45

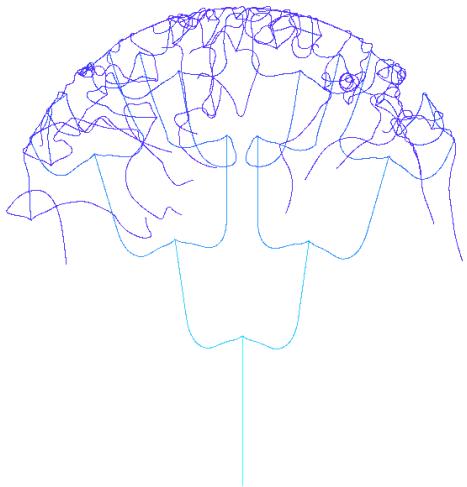
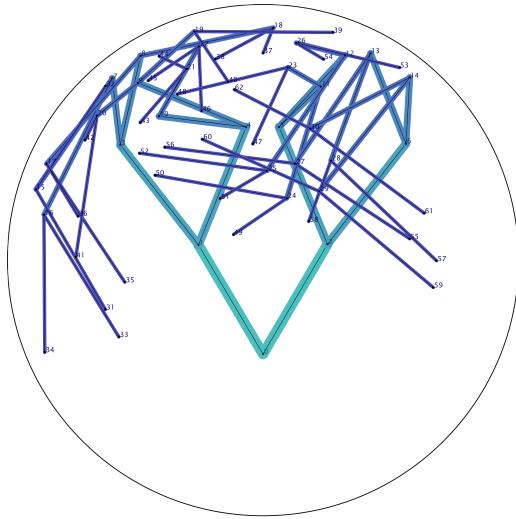


Figure 48: spawnAnglesDegree = +/-45



18-3-15-17-32

Figure 49: spawnAnglesDegree = +/-90

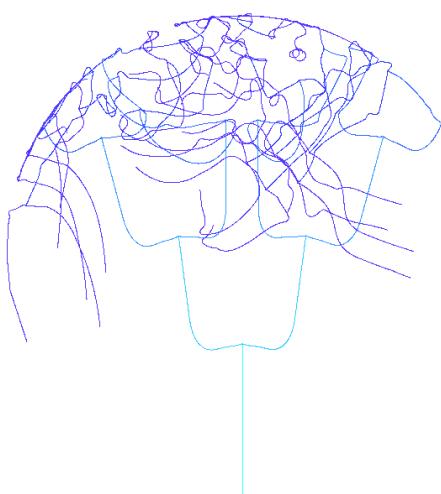
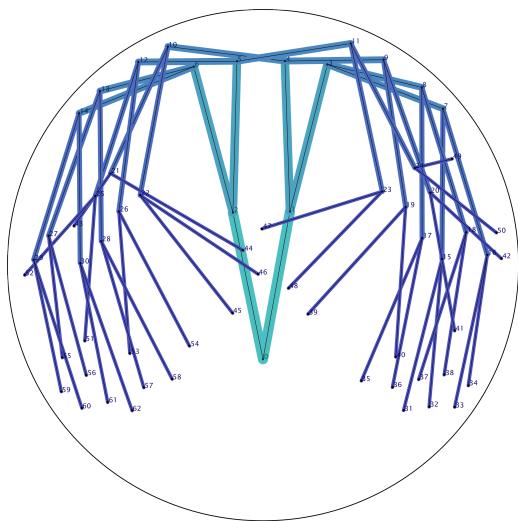


Figure 50: spawnAnglesDegree = +/-90



18-3-15-37-43

Figure 51: spawnAnglesDegree = +/-135
(default)

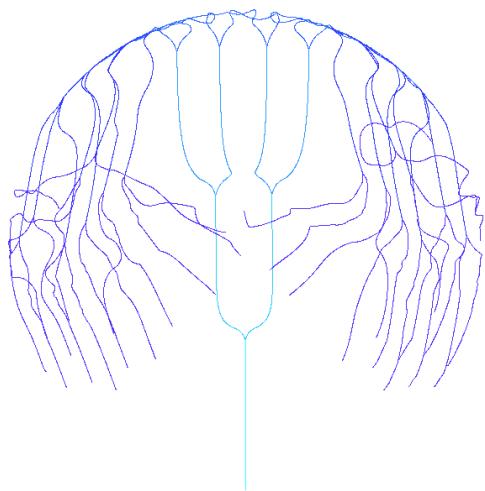
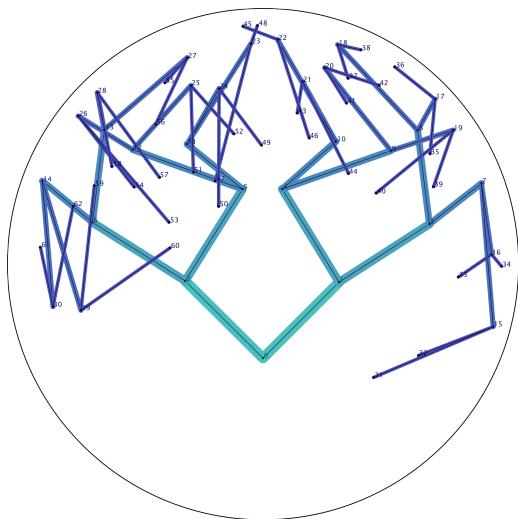


Figure 52: spawnAnglesDegree = +/-135
(default)



18-3-15-19-23

Figure 53: spawnAnglesDegree = +/-180

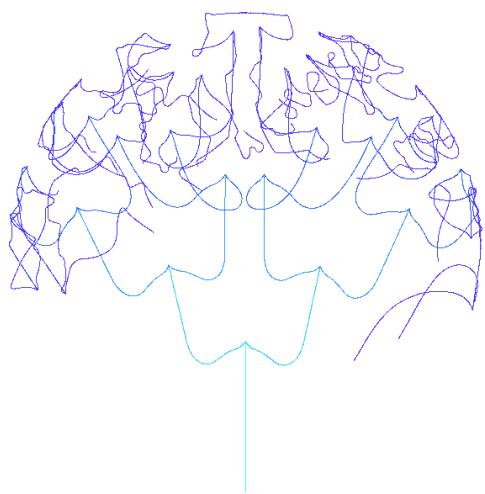


Figure 54: spawnAnglesDegree = +/-180

4 Textile Implementation

4.1 Thoughts on Goals

Throughout the whole development of the algorithms and parameter variations I played with several ideas on how to go from digital to textile. The basic idea was to use threads as entities or individuals. However, I was very indecisive about the details. Early ideas included just embroidering the movement of the fish or use the movements to (pseudo-) weave. I also came up with a Macramé-related knotting scheme, that resembled an RPG: each thread could win or lose in a fight against other threads.

None of those concepts are interesting enough for me. All of them have a building or producing element in common, so I tried to turn that element upside down. Working with textiles is usually constructing something. However, there will be steps you have to reverse, because you made a mistake or the material was flawed. You have to untie the knot, unweave your cloth or just untangle the mess. Yet these unwanted destructive processes happen in favor of the final product.

I want to exaggerate the principle of destruction for my textile piece. Still adhering to the threads as individuals it means deconstructing the elements itself: one thread contains other, thinner threads and fibers. The parent-child-relation between the fish in my tank fits in seamlessly with the construction-by-deconstruction-concept. The parent fish contains all the material needed to form the children, which then again contain everything for their children.

I will use a thread and split it up into subthreads, which match the connection between parent and children.

4.2 Textile Piece I

4.2.1 Method and Tools

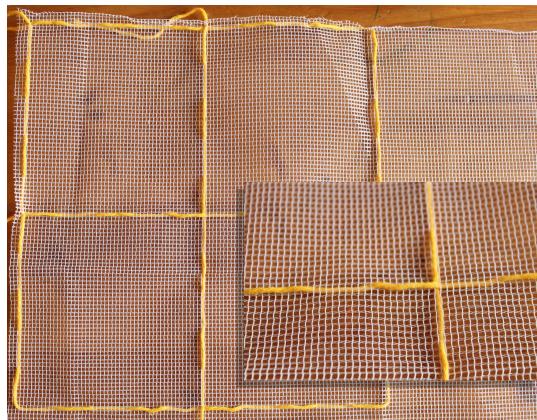


Figure 55: Base material as canvas

For my first practical trial run I use even-weave with approximately 2mm per unit. To limit the size I scale down the computed positions to fit a 100 by 100 grid (resulting in appr. 20cm by 20cm for the finished piece). The yellow thread is used as a counting help.



Figure 56: Applied material

Because my generations of fish are based on the number 2, I also need thread based on number 2. So I use cotton wool threads each already containing 8 (= 2^3) sub-threads when feazed.

To be able to manage the threads and associate them with one fish id I simply attach some paper to the string groups. The positions are directly read from the text in the log file.

```
tank cx,cy,radius: 300.0 300.0 300.0
starter fish: 1
start dir: 1.0 1.0
start pos: 300.0, 300.0
wall/neighbor 0.9/0.1
maxGenerations 7
maxChildren 2

spawnTime 5000ms
spawnAnglesDegree 135.0 -135.0
turnSpeed 0.0051/ms
private radius / company radius 40.0 / 75.0

fish:
0 g:0 born 300.0 300.0 : 140 140 c:
0 g:0 died 409.58185 409.58185 : 191 191 c: 1 2
1 g:1 died 434.7048 547.7104 : 203 256 c: 3 4
2 g:1 died 547.1275 435.0458 : 255 203 c: 5 6
3 g:2 died 299.06696 479.5542 : 140 224 c: 7 8
4 g:2 died 290.70502 579.7467 : 136 271 c: 9 10
5 g:2 died 578.5094 290.45987 : 270 136 c: 11 12
6 g:2 died 474.91296 301.06262 : 222 140 c: 13 14
...
...
```

Figure 57: First part of the used log file

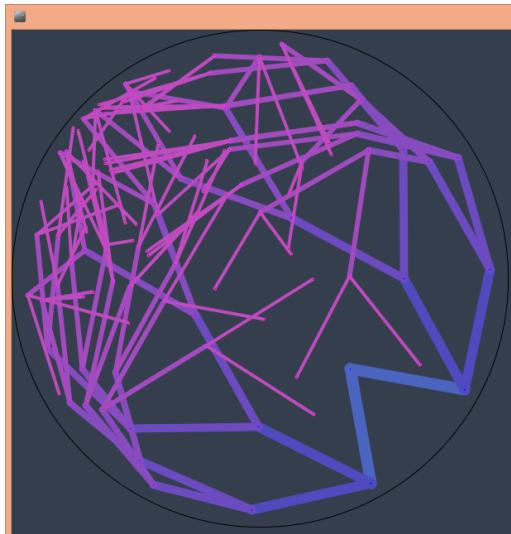


Figure 58: Connection lines between (dead) fish of 7 generations

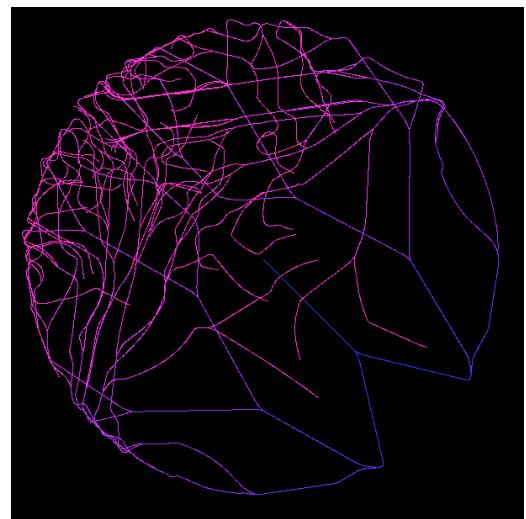


Figure 59: Traces of the movement of the fish while living

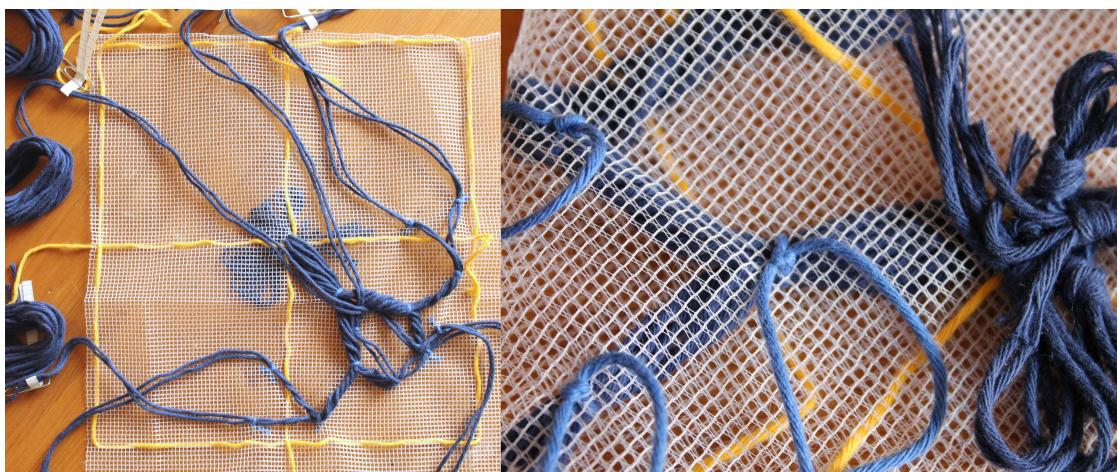


Figure 60: Total view of the experiment and details on the back. For 7 generations of fish, each having 2 children I need 2^7 threads (128). Using the 2^3 cotton strings, I need 16 strings in total - which is quite thick. Fitting it through one "grid point" compromises the accuracy of the grid physically, so I will dilute the positions for the first maybe also for the second generation of fish. The starting point for the fish is 50/50. After dividing the thick 16-strings-cord into 4 subcords with 4 strings each, the starting points are 51/50, 49/50, 50/49, 50/51.

4.2.2 Result

The left of Fig.60 shows the state of the first experiment when it is apparent I am not getting anywhere near to what I wanted. One reason is a simple mistake in determining the thread count: For 7 generations I actually would only have needed 2^6 instead of 2^7 single threads in the end, because I forgot to start counting at 2^0 . This would have reduced the thickness of the starting thread by half and also changed the appearance significantly.

A more serious problem for me is that the visual outcome is somehow off. Maybe I chose the evenweave's units too large or the wrong color, but most importantly the intermediate result does not have the properties I expected it to have: it is neither 2D nor 3D, neither are the threads stiff and straight enough for the connections of the fish nor do they show the actual movement of a fish during its life cycle. This first piece is an unsatisfying accumulation of imprecise craftsmanship, so I have to find other possibilities to work with the pattern in the material world.

4.3 Textile Piece II

4.3.1 Method and Tools

Unlike techniques such as stitching or weaving where the interaction between up and down movements is crucial, the destruction concept I want to explore is independent from that. That means I need a more robust canvas. I use a non-textile base, namely a semi-see-through PVC sheet. Although it is possible to cut holes in the PVC for guiding the threads, as said before, this would artificially add something to the concept that does not belong to it. Instead, a hot glue gun is used to fixate the splitting points and keep the strings neatly in place. Pieces of a series can be stacked and the viewer can easily compare the different layouts of the strings.

For the threads I use the same type of cotton wool as with experiment number one (in white instead of blue), but only 6 generations, meaning $2^5 = 32$ sub-threads which equals 4 of the normal threads. The wool is hand-dyed to achieve a color gradient from dark blue to green to yellow to white, which aids the visual understanding of distinct generations. The dyeing process makes it more difficult to feaze the threads.

4.3.2 Result



Figure 61: Finished piece number 1



Figure 62: Detail

There are four templates for the finished pieces, including the default setting, which are described under Fig. 11, Fig. 31 and Fig. 33.

I am satisfied with the result, as it fulfills the goals I set before. There are definitely alternatives for the fixation method, but at this point the glue dots, which are quite visible despite being transparent, act as subtle placeholder for the fish bodies.

Additionally, while working with the threads, some aspects emerged that I did not actively take into account in the planning phase. The first is the overlapping of the threads, naturally falling into place if the correct order of the fish births and deaths is maintained. The second is the color imperfection as a result of the (amateur) hand-dyeing. The dye did not touch any fibers below the surface of the initially whole thread, as a consequence the feazed threads partly stayed white. Both aspects fit beautifully into the whole concept, the latter one especially adds another element to the destructive dimension.

5 Conclusion

We see crafting usually as a process of creation, building, fixing and making. But most crafting techniques have destructive aspects in between: we cut fabric into pieces before sewing and we remove excess threads when knitting.

I started with an artificial, minimal life form, that moves, gives birth and dies and interacts with the other life forms in a complex way, based on a set of simple rules. I explored variations of the rules and observed changes in the complex behaviour of the life forms, developing both expected and surprising features as outcome. Then I turned its heritage into a destructive textile piece, where I consciously nullified the effort of making a thread from several fibres, by feazing and dissecting it.