3D Neural Cellular Automata

Simulating morphogenesis: Shape, color and behavior of three-dimensional structures

Ole Edvin Skjeltorp

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

3D Neural Cellular Automata

Simulating morphogenesis: Shape, color and behavior of three-dimensional structures

Ole Edvin Skjeltorp

© 2022 Ole Edvin Skjeltorp

3D Neural Cellular Automata

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Plants, fungi, humans and all other multicellular organisms go through the same process of growing step by step. Starting as a single cell with a genome containing all the genetic information of the organism, they grow into the shape encoded in their genome with stunning accuracy. Not only do they grow into a shape, but a complex composition of cell types. Organisms also know when to stop growing, and some even have abilities to regrow damaged cells. The study of this process, called developmental biology, can provide insight useful for a range of disciplines, such as medicine and artificial intelligence. Computer science has a long history of benefiting from mimicking models of biology, and modern computing power provides tools to simulate biological models in ways that may benefit both fields. Simulations can provide insight and observations that are hard to catch otherwise.

This thesis contributes to the tools capable of providing such insights, and aims to simulate morphogenesis by growing a single cell into a three-dimensional colored shape. The framework extends recent work simulating 2D morphogenesis using machine learning combined with an abstract computational system called cellular automatas (CA). In addition to the added dimensionality, we further extend the framework and propose a novel solution allowing guidance of the morphogenesis through certain checkpoints during training. We also experiment with a novel approach of training a simple 3D model to exhibit an oscillating motion, with promising results laying the foundation for future work exceeding past simulation of just morphogenesis. A formula for estimating a hyperparameter, the minimum number of updates a CA needs during training, is derived to provide a basis for future work on 3D neural cellular automatas (3D NCA).

The framework is successfully adapted to the higher dimensionality and three-dimensional morphogenesis is simulated with high precision on a range of models covering different geometrical challenges. Both shape and color is correctly grown from a single cell, smaller models are indistinguishable from their targets, while larger models tend to have a few cells misplaced. We observe a significant increase in computational cost with the three-dimensional simulations, indicating that optimisation measures would be critical if using the framework on large scale simulations. In terms of simulating morphogenesis, the framework matches the performance of similar work published while this thesis was written, in this relatively narrow but fast evolving field.

Preface

This research was conducted at Robotics and Intelligent Systems (ROBIN) at the Department of Informatics, University of Oslo, in 2021 to 2022.

I would like to express my gratitude to my supervisor, Mats Erling Høvin, for his never ending enthusiasm and encouragement, and helping me keep focus on the right aspects of the research. I would also like to thank my co-supervisor, Kai Olav Ellefsen, for useful insights and discussions.

Thanks to my fellow students for a great time, and to my friends, family and to Mathilde, for always supporting and putting up with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research motivation

When solving complex problems computationally, the use of biologically inspired computing is a well-established practice, with a quickly growing list of examples surpassing human-level performance in a variety of tasks [2, 10, 29, 37]. Properties found in biological self-organising systems such as parallelism, asynchrony and stochasticity have become increasingly sought after in the field of computation [16], making the incorporation of biological computing principles a natural step. Not only does this benefit the progression of computer science itself, this also paves the way for using computer science to gain insight in biological systems through simulation, or even further down the road, lay the foundation for a new generation of biologically inspired computers.

Recent work on differentiable self-organizing systems [18] has shown impressive capabilities of simulating morphogenesis by combining machine learning techniques with a cellular automata. This simulation is performed in the 2D plane, while in reality this plays out in a three dimensional space. To increase computational capacity and lessen the gap between simulation and reality, we aim to extend the simulation to work in a three dimensional space. This will increase the scope of cases in which the system can be used to simulate, while also adding increased visual flexibility as a result of the added dimensionality. This opens up the opportunity of applying it to physical structures, such as bio-inspired robots. Simulating with increased dimensionality will also provide useful insight of how the computational requirements scale, to give an indication of applicability. In addition to create a model able to stabilise at a desired shape, we propose functionality to utilise more of a CA's dynamic properties as a system well fit to simulate motions and oscillating patterns, by changing the desired behavior from a single shape, to multiple shapes iterating over time.

While the work of this thesis was still ongoing, a paper by Sudhakaran et al.[27] was published with a research goal aligning with ours – extending the dimensionality of [17] to 3D. It was exciting to see this narrow field gain some momentum, and this reassured the quality of our research goal. Their work is based on a different environment, however it is worth noting that

during the process of increasing the dimensionality we share some trivial architectural similarities.

1.2 Research goals

- Develop a neural cellular automata operating in three dimensions, able to grow from a single cell into a given shape, size and color.
- Propose a novel solution allowing guidance of the morphogenesis, by introducing constraints in form of checkpoints the model must pass through before reaching its final form.
- Develop logic enabling a 3D NCA to exhibit a given oscillating motion, rather than growing into a static shape.
- Derive a formula for determining a reasonable range of simulation time steps (updates of the NCA) to use during training of systems of similar architecture.
- Discuss limitations and applicability of the proposed framework.

1.3 Main contributions

Three-dimensional morphogenesis is simulated with high precision on a range of models covering different geometrical challenges. Both shape and color is correctly grown from a single cell, smaller models are indistinguishable from their targets, while larger models tend to have a few cells misplaced. We observe a significant increase in computational cost with the three-dimensional simulations, indicating that optimisation measures would be critical if using the framework on large scale simulations. In terms of simulating morphogenesis, the framework matches the performance of similar work recently published in this relatively narrow but fast evolving field. The framework is further extended and we propose a novel solution allowing guidance of the morphogenesis, achieved by introducing checkpoints to the training process. Additionally we propose a method for training a 3D model to exhibit an oscillating motion, going past simulation of just morphogenesis. A formula for estimating a hyperparameter, time steps used during training, is derived to provide a basis for future work. Applicability of said systems are discussed in light of computational cost, and in comparison to related work.

1.4 Outline

- Chapter 2: Related work. Introduces the background theory that laid the foundation for the related work, followed by recent work.
- Chapter 3: Theory: Categorises and supplements the relevant theory introduced during the previous chapter.

- Chapter 4: Method: Presents the proposed framework, explains design choices and defines terminology.
- Chapter 5: Software And Setup: Lists the hardware, software and frameworks used.
- Chapter 6: Experiments: Presents a selection of experiments of increasing complexity and their corresponding results.
- Chapter 7: Discussion: Sums up and reflects over the results.
- Chapter 8: Conclusion and future work.

Chapter 2

Related work

This chapter will go through some of the related research and publications. The background section goes through the introduction of cellular automata and some work that influenced it in the following years, supplemented by the surrounding theory. The related work section takes a look at recent work more closely related to the subject of this thesis, starting with an in-depth look at the main inspiration, and other research derived from it.

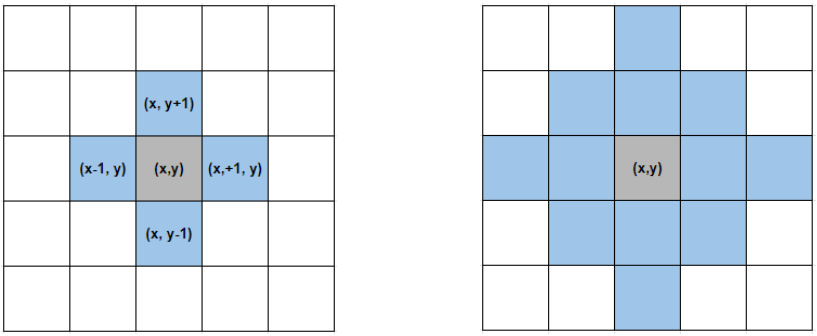
2.1 Background and theory

The first models of CAs were created by Von Neumann with the purpose of providing insight for machine self-replication using a mathematical model [19]. A traditional CA consists of a universal clock, and a number of finite-state machines (FSM), usually referenced to as cells, given their equivalent behavior in biology. Each cell can be in a finite number of states at any given time, and all cells are systematically arranged in a defined structure. In a Von Neumann cellular automata, they are arranged in a two-dimensional Cartesian grid, while other structures can also be used, such as the one-dimensional used for language recognition in [24], or even triangular and hexagonal grids. The state of each cell is a function of the states of the immediately adjacent cells, known as the *Von Neumann Neighborhood*[32], defined as

$$N^{v}_{(x_0,y_0)} = \{(x,y) : |x - x_0| + |y - y_0| \leq r\} \tag{2.1}$$

for any given cell (x_0, y_0) in a two-dimensional Cartesian grid, where r is the range of the neighborhood, see figure 2.1. For each time step all cells are updated based on a static rule table shared by all cells. Commonly, cell states are represented as different colors, although this is just a convenient way of portraying states - each color can represent virtually anything. Specific for a Von Neumann CA, the rule set consist of 29 different states. Von Neumann was successful at creating a theoretical self-replicating machine based on a CA, and demonstrated CAs capability of simulating evolutionary processes such as self-reproduction.[1]

Conway's Game of Life [6] became a landmark in the field of CAs with its remarkable ability to display "life-like" patterns. It was designed



(a) Range $r = 1$, 4 neighboring cells (b) Range $r = 2$, 12 neighboring cells

Figure 2.1: Von Neumann neighborhood(blue) for a cell (grey) located in the center of the canvas. Figure (b) demonstrates how the neighborhood scales with range. The center cell is affected by all blue cells.

to exhibit unpredictable behavior, analogous with the rise, fall and alternations of a society of living organisms. These CAs were originally calculated by hand, but gained more traction as computers evolved and were able to run simulations. Game of Life operates on a two dimensional Cartesian grid, with cells having only two possible states; dead or alive, represented by black and white. While the name implies this is a game, it is a zero-player game, and is to be considered more of a simulation in this context, much like the other CAs described here. It takes any given initial pattern of alive cells as input before iteratively applying the rule set for as long as the simulation is run, where each time step can be considered a generation. As the rules always stays the same, the initial pattern, called the seed, is what defines how the system evolves. Game of life determines a cell's neighborhood using Moore neighborhood[31], defined as

$$N^{M}_{(x_0,y_0)} = \{(x,y) : |x - x_0| \leq r, |y - y_0| \leq \ r\} \tag{2.2}$$

where r is the range of the neighborhood for any given cell (x_0, y_0). Despite its simplicity and few states, Game of Life is touring complete. Looking at the simple rule set gives an intuition of how CAs operate:

1. Survivals. Every counter with two or three neighboring counters survives for the next generation.

2. Deaths. Each counter with four or more neighbors dies (is removed) from overpopulation. Every counter with one neighbor or none dies from isolation.

3. Births. Each empty cell adjacent to exactly three neighbors–no more, no fewer–is a birth cell. A counter is placed on it at the next move. [6]

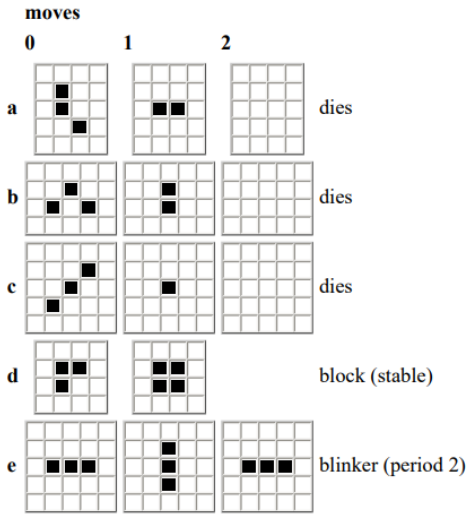


Figure 2.3: Conway's Game of Life: 5 examples displaying evolution over 3 generations, for 5 different starting seeds. [6]

5 examples **a,b,c,d,e** of starting seeds can be seen in Figure 2.3. Time steps **0,1,2** display how they evolve over 3 generations. Seed **a,b,c** dies, seed **d** reaches a stable state, and seed **e** reaches an oscillating state.

A CA is applied in the context of 3D free-form shape modelling in [3], where it is used to simulate deformation of clay in a 3D space, using a rule set based on physical conservation laws. The usage of a CA is motivated by a reduced computation time compared to methods based on strict physical laws. A framework using short computation time is achieved, demonstrating realistic and intuitive behavior of the virtual clay.

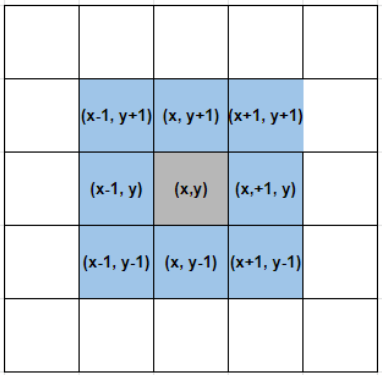


Figure 2.2: Moore neighborhood (blue) for a cell (grey), with r=1. The neighborhood consist of 8 cells. A radius of 2 would include the surrounding white cells.

The CAs described above are conventional CAs where one designs a set of rules, and simulates the behavior the chosen rule set generates. This process gets turned on its head in [34], where the concept of neural cellular automatas (NCA) is introduced. Instead of searching for a behavior given a rule set, a rule set is searched for given a desired behavior. This is done by training a neural network of Σ-Π units with short-range connections. Different degrees of success is achieved, depending on constraints placed on the learning, but a neural networks ability to learn simple rules and model the underlying rules of a CA is demonstrated. The field of machine learning has exploded since then, as one of the most rapidly growing technological fields [13], creating a whole new set of tools and computing power available, making this approach viable on a scale which previously

was not possible.

This reversed process of searching for a rule set has seen success with a variety of biologically inspired computing methods [5, 12, 20, 30]. The concept of using biologically inspired computing for finding such rule sets is a logical step, as the very rule sets one is trying to mimic was developed in a similar matter by nature as a result of evolution.

2.2 Related work

Mordvintsev et al. [17] proposes a framework for a differentiable model of morphogenesis, essentially simulating the growth from a single cell to a given shape, based on a neural cellular automata. Their research is motivated by understanding the underlying dynamics of biological life, and getting insight to what lies behind its plasticity and robustness, which would enrich both the field of developmental biology and computer science.

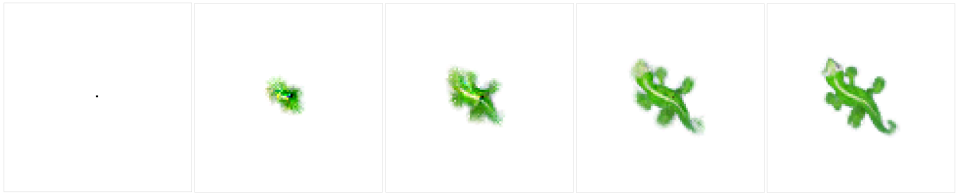


Figure 2.4: Growing Neural Cellular Automata: morphogenesis of a salamander. A single seed (far left) side is grown into a complete salamander (far right) using an NCA. The NCA is updates 20 times between each frame. This simulation takes place after the neural network model has been trained. [17]

A salamander can be seen growing from a single seed (pixel) to its complete shape in Figure 2.4. In each frame from left to right, the NCA has developed for 20 steps. The NCA operates on a 2D Cartesian grid, with a pixel as the equivalent of a cell, and the end shape is a picture composed of pixels. Instead of representing each cell state as an FSM, their states are represented as a vector of continuous values. This is the foundation for having a differentiable update rule, as it allows for a gradient to be computed of the loss function with respect to the weights of the neural network. Each cell-vector is of length 16 and holds 4 values representing RGBα used to visualize the cell's color, and the remaining slots are referred to as hidden channels, which are described as chemical signalling stored internally in a cell in the biological analogy. Certain thresholds on the alpha channel determines which cells are considered living and dead. An update step of the model consist of for all cells, perceiving the states of neighboring cells to create a perception vector, which holds information about the neighborhood, as well as the cell itself. The perception vector is propagated through a small neural network, consisting of a dense layer of size 128, followed by ReLu, and smaller dense layer of size 16 as the final layer. The

output updates the cells state vector incrementally. Additionally, cells are updated stochastically, which differs from typical CAs. The motivation for this is to resemble the dynamics of a biological self-organising system.

Alongside learning to grow into a given shape, functionality for persisting and regenerating is proposed. Persisting means stability for any given time step. Growing from a seed state to an end state is a challenge, but staying stable, holding the shape for any given time is another challenge, and needs to be taken into account during training. If an NCA is only trained to simulate morphogenesis over a set amount of time steps, it becomes unstable if simulated longer than trained for. Figure 2.5 shows what the models evolve to after being simulated past their training time. The left model has not been trained adequately, and is still unstable as it has only been trained to persist for 100 training steps. The model on the right side has been trained for 4000 training steps, which appears to be enough for the model to stabilise at its goal-shape at any given time step.
Learning to persist is done by altering how seeds are used during training.

Figure 2.5: Growing Neural Cellular Automata: CA behavior at training steps 100, 500, 1000, 4000, respectively. After just 100 steps, the target is not recognisable and the NCA "explodes", whereas after 4000 training steps the target shape persist. [17]

Instead of always growing from the same one-pixel seed, a pool of seeds is used. The contents of this pool is updated during training by adding previous high scoring output-states, replacing some of the seeds. This way the networks learns an attractor for the goal-shape from different start-positions, adjusting the dynamics slightly compared to a CA trained without pooling. On early stages the pool will consist of less accurate shapes, and over time the pool will slowly be refined to consist of patterns close to the goal-shape, and finer adjustments are made.

Regenerating means that the NCA learns the ability to recover from damage. If a limb of the salamander in Figure 2.4 was removed, a new one should grow out. Given the nature of how CAs work, there is already a certain capability to regenerate, but this feature is enhanced by further modifying the pooling-function by damaging the models in the pool. This broadens the landscape of models the NCA learns an attractor for the goal-shape. Figure 2.6 displays a fully grown salamander on the left side, which in the next frame gets half of the model removed. In the following frames, the salamander grows into its complete shape.

Based on [17], a new type of model capable of learning a space of

Figure 2.6: Growing Neural Cellular Automata: Patterns exposed to damage during training exhibit astounding regenerative capabilities. [17]

programs in the form of CA is introduced by Ruiz et al [22]. A manifold of NCAs able to generate their respective images are encoded in a model. A cell's environment information is combined with an encoding in an Auto-Encoder architecture performing dynamic convolution. In the encoder-decoder architecture, the NCA becomes the last part of the decoder, as it is reproducing an image in the end. Generalization capabilities is demonstrated, and even though the design goals are different and can not be used as a baseline, the mean square error achieved is comparable to the one in [17]. Their model is demonstrated using both emojis and images from from the CIFAR-10 dataset. In the biological analogy, they place the model's function right before morphogenesis, where genes are mapped into specific proteins that drive cellular differentiation.

A probabilistic generative 3D model named Generative Cellular Automata is proposed by Zhang et al. [36], generating diverse and high fidelity shapes in a voxel space. This differs from an NCA by formulating the shape generation process as sampling from the transition kernel of a Markov chain, and the transition kernel employs the local update rules of a CA. Their model gains increased performance by exploiting the connectivity and sparsity of 3D shapes, however, the generated models have no colors or other attributes and only the outer shell layer of a shape is generated, which means their models perform great visually, but lack information about inner structure.

Another extension of [17] applies an NCA to generate 3D structures in a Minecraft environment [27]. As mentioned earlier, this was published while the work of this thesis was ongoing, and shares the research goal of extending dimensionality to 3D. This simulation environment allows the use of building blocks with a variation of properties, such as different visual textures, and how some blocks interact with each other and the environment. For instance, some blocks react and transform into another block if they are neighbors, and some blocks push and pull other blocks if they are adjacent to specific blocks. This allows the NCA to grow what is referred to as functional machines, which is essentially a simulation of how the building blocks in the grown structure reacts to each other and the environment. Structures composed of up to 3000 building blocks are successfully grown, and stability and regrowth capabilities similar to [17] are demonstrated. An NCA's ability of generating 3D structures is demonstrated. However, the behavior their functional machines are exhibiting is merely a result of the mechanics of the EvoCraft environment, and not dynamics of the NCA itself, as the NCA only grows the initial

starting configuration for the environment.

Chapter 3

Theory

This chapter will categorise and supplement the relevant theory introduced during the previous chapter.

3.1 Automata theory

Automata theory is a branch of theoretical computer science where abstract machines and automata are put to the task of problem solving. An automata represents a simple machine, often working in a manifold to collectively perform a logic of computation. Each of them function as self propelled computing device, processing an input which leads to an output based ion a predetermined sequence of operations. They come in a range of variants and are defined by the "machine" they are modelling. Common distinctions are discrete automata, analog automata and continuous automata.

The first cellular automatas described in the Background section, the Von Neumann CA and Conway's Game of life, falls under the category of discrete automatas. The CAs described in the related work section however, falls under the category of continuous automata, which is the focus of this thesis. The key difference is the continuous cell state representation, which makes a shift in the systems these CA's are fit for describing. While a discrete automata succeeds at consistency, reliability and logic, the continuous automatas see success in modelling biological, physical and chemical systems. Extremely simple rules in a continuous cellular automata can generate behavior of considerable complexity [33].

3.1.1 Morphogenesis

Morphogenesis describes the process where cells, tissue or an organism develops its shape.

> The ability to progress from simple to more complex, organized, and spatially differentiated forms, or morphogenesis, is, perhaps, one of the most fundamental properties of biological systems from individual cells to large multicellular organisms, to whole populations. [8]

It is studied not only in biology, but in multiple disciplines such as physics, chemistry and mathematics. In computer science, we can befit from theories derived in these disciplines to run simulations, while also benefiting from the tools available in computer science. Accurately modeling all the factors in play during morphogenesis requires an immense insight, which calls for alternative methods.

3.2 Differentiable programming

The proposed method for simulating morphogenesis falls under the category of self-organizing systems [18], and more specifically the use of differentiable programming. Differentiable programming encapsulates neural networks and deep learning, but also any program utilising automatic differentiation, which allows for automatically computing derivatives of functions. This is the key functionality enabling simulating morphogenesis without manually describing the dynamics of the system. Instead machine learning is used to describe these dynamics, found by adjusting the parameters of a model through gradient descent, by calculating derivatives of a loss function.

3.3 3D Simulation

This section briefly introduces a reoccurring concept in this thesis – scaling a simulation from a two-dimensional, to a three-dimensional environment. Table 3.1 is included to shed some light on how the total number of cell or voxel spaces scale for a three-dimensional space. If all axes are equal, the amount scales exponentially from 2D to 3D. The example sizes are chosen based on the ones used in the experiments, and this table will be revisited when discussing how this affects performance of the framework proposed in Chapter 4 and 7. Note how the total amount of voxels scale in the 4th column.

Height	**Width**	**Depth**	**Total voxels(h*w*d)**
5	5	5	125
10	10	10	1000
20	20	20	8000
32	24	12	9 216
32	32	32	32 768
40	40	40	64 000
64	64	64	262 144
128	128	128	2 097 152

Table 3.1: Amount of voxels corresponding to dimension size for a three-dimensional space.

Chapter 4

Method

This chapter presents the proposed framework for simulating three-dimensional morphogenesis. The reasoning behind design choices and their effects are derived, as well as presenting the theory of adding movement to the simulation. A scope of terminology used throughout the experiments chapter is defined.

4.1 3D Neural cellular automata

To achieve the research goal of simulating morphogenesis in three dimensions, an extension of the NCA architecture proposed by Mordvintsev et al. [17] is proposed. The key elements of the architecture persists, such as cell state representation, stochastic cell update, living cell masking, stability and regeneration measures. Architectural tweaks are proposed to create a 3D neural cellular automata (3D NCA), and furthermore the functionality of the framework is increased with a novel approach of adding movement to the simulated models.

The rule table of a traditional CA is stored in the weights of a small, dense neural network, and each cell's state is dependent on neighboring cells in a three-dimensional Moore-neighborhood. The cells are iteratively updated in a stochastic manner. The NCA is trained to grow from a single cell to a given shape, and each cell is assigned a specific RGB color. The target shape and cell color itself is somewhat arbitrary and only used as examples for demonstrating the process. Figure 4.1 shows the architecture of a time step update as the NCA evolves. Perception of the cell's environment takes place in the left block, and the new states are generated in the right block. The far left side shows the cells in time step t. On the far right side, the cells has gone through an update, ending in time step t+1.

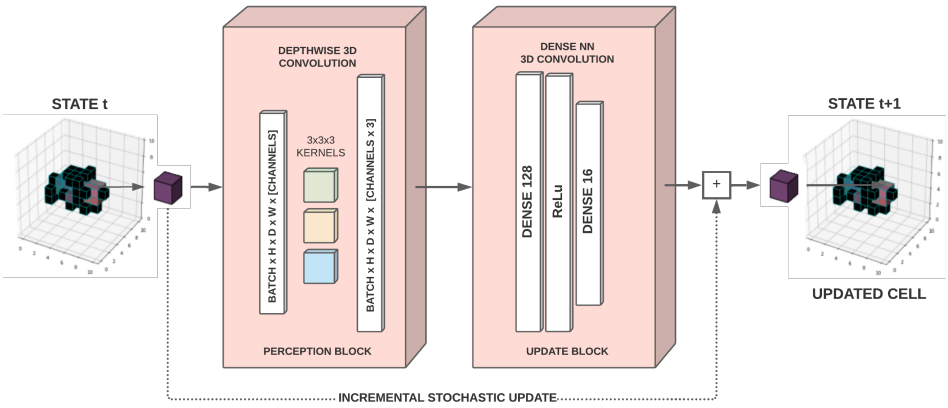


Figure 4.1: 3D NCA Architecture: Left side: A set of cells in state t on a canvas in time step t. A cell is forwarded to the first block, which perceives the cell's environment through depthwise 3d convolution. The perceived states of the surrounding cells are forwarded to the next block together with the state of the cell itself. This data is propagated through a small neural network (second block), which generates an incremental update to the cell' state. The cell is now on the far right side, and has reached state t+1. This process happens batch-wise and simultaneously for all cell on the canvas.

4.2 Design choices

4.2.1 Canvas and cell representation

A CA consist of cells organised on a *canvas*, see Figure4.3. These cells can be represented in different ways. The simplest way of representing a cell is a boolean assigned to a specific location on a given canvas, as true/false or 0/1. Each spot on the canvas is a cell, and the cell's state is described by its assigned value. To demonstrate, a one-dimensional CA of size 4 with cells having only two possible states could be represented as a vector [0, 0, 1, 0]. The cells in cell-spot 1, 2 and 4 have the state 0, while the cell in cell-spot 3 has the state 1. Alive/dead is the biological analogy for a cell's state being 0/1. This representation can be extended to any number of dimensions, with the canvas being multidimensional binary matrices. *Cell representation* does not refer to how a cell's state is internally represented, but how it is externally represented, typically visually, if the simulation has a visual projection. This is depending on the problem the CA is applied to. Visual representation allows for an intuitive way of observing a CAs behavior, however when the dimensionality surpasses 3D,

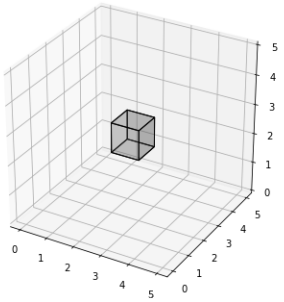


Figure 4.2: A voxel in a 5x5x5 canvas (height, width, depth, usually refereed to as x,y,z).

humans struggle with visual representations. CAs operating as computer simulations on a 2D grid is often represented as pixels [5, 6, 12, 17, 20]. For each time-step, an image is produced, where each cell corresponds to one pixel in the image, where typically alive cells are rendered visible and dead cells are rendered invisible.

$$N_{cells} = C_{height} * C_{width}. \tag{4.1}$$

Where N_{cells} is the total number of cells operating on the canvas, C_{height} and C_{width} are the dimensions of the canvas.

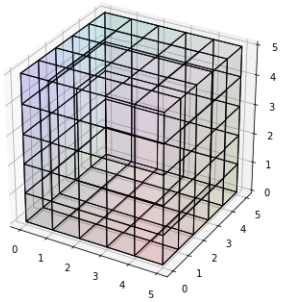


Figure 4.3: A 5x5x5 canvas consisting of 125 voxels.

For simulating 3D morphogenesis, voxels were chosen as the cell representation, see Figure 4.2. Similar to pixels, voxels represent a location, but on a three-dimensional grid. The voxel itself has no internal reference of its own placement, it is defined by a space or a slot on a 3D grid. Representing a figure as voxels with a 3D matrix was motivated by the large set of powerful computations easily available for matrices, and its direct uncomplicated "one-to-one" way of representing a structure being directly transformable to a cell in a CA. A key step in a CA is perceiving the neighborhood of a cell, which happens for all cells at every time step. Locating and accessing neighbors when representing the canvas as a matrix is highly efficiently done with simple indexing in programming. The model could also be represented as an array of 3D-coordinates, which would be more memory efficient for large and sparse models, but the ease of use of a matrix representation was valued higher. Other popular ways of modeling a 3D structure, such as polygon meshes or volumetric meshes[4], which describes a 3D surface based on lists of vertices, edges and faces, would offer a useful computer graphics rendering toolbox, but considering the process of coordinating the correspondence between cells and points in a mesh, it would not be worth for the level of simulations aimed for in this thesis, as proof of concept is valued over visual performance.

4.2.2 Cell state representation

Cell state representation refers to how a cell's state is internally represented. It can, like the canvas, be extended to virtually any complexity. Like mentioned in the previous section, the simplest way of representing a cell is as a binary – the cell is either dead or alive. This referred to how the cell is perceived externally, while internally, the cell may have a more complex representation. To clarify, a cell considered to be in one of two states by the CA, may internally be represented more complex than with a binary, for instance with any natural number, meaning multiple internal states would correspond to each of the two external states. How a cell state is represented relates to the type of CA. Traditionally, CAs contain cells acting

as FSMs, meaning each cell can be in exactly one of a finite number of states at each time. In NCAs, the cell state is represented as a continuous value, breaking with the concept of acting as a FSM, but allowing a differentiable update rule.

The chosen cell state representation follows [17], where the cell state is represented as a vector of real values of length 16, see Figure 4.4. The first three entries contain the red, green and blue channel values, describing the cell's color. The next channel is an alpha channel, which in computer graphics is used to determine a pixel's visibility. This channel is used to determine whether a cell is considered dead or alive, see next section. The following channels 5 to 16 are considered hidden channels. Mordvintsev et al. describes these as "They can be interpreted as concentrations of some chemicals, electric potentials or some other signaling mechanism that are used by cells to orchestrate the growth" [17]. The neural network learns during training to produce state vectors encoded in such a way that desired behavior is achieved, using these channels to identify a cell beyond just its color. The exact number of hidden channels necessary would be hard to tell, but generally smaller, simple figures should be able to make due with fewer channels than larger more complicated figures could. The length of the cell state vector becomes one of the hyperparameters of the NCA. The length 16 is kept static during the experiments of this thesis, due to the large amount of other parameters in play.

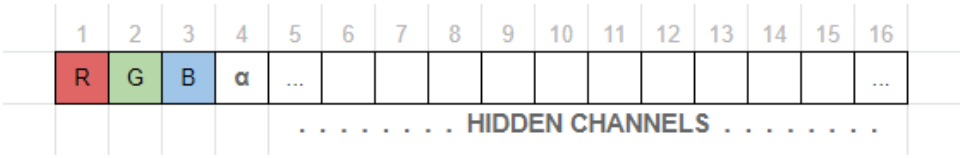


Figure 4.4: Cell state vector. Each cell contains a state vector of 16 real values. The first three channels are used to represent the cell's color in RGB format. RGB is represented as three seperate channels with values ranging between 0-255. Here, they are scaled to a float value between 0 and 1. The 4th channel, α is used to determine if a cell is dead or alive, based on a threshold. The remaining channels contain hidden values generated by the neural network, and contribute to describing the cell's state. This cell representation was derived by [17], and is also seen in [27], but with a slight change as the RGB channels is instead represented by a one-hot vector denoting a cell type.

4.2.3 Colors

All voxels in the target shape are assigned an RGB value. Both the shape and the colors of a grown structure is evaluated during training of the model. The color is described in RGB format as 3 float values, each channel having a set spot on the cell state vector. Colors may not bare an immediate equivalent in the biological analogy, but is rather used to demonstrate the ability of learning not only where a cell should grow, but what type of cell it should be. The ability to grow a cell in a specific spot with a specific

value proves that the system is able to grow cells with any other learnable attribute as well, such as different types of cells. The colors used in the target shapes are in most cases arbitrary colors, created by distributing a color range across the 3D space in such a way that each cell has a unique color. This is done to test the models performance with a challenging color distribution, instead of mass assigning the same color value to large parts of a model. The process of manually assigning colors to a cell is tedious and does not add any value in the context of this thesis.

4.2.4 Target shapes

Just like colors, the actual shape of the structure we are trying to grow does not particularly matter. For instance, simulating the growth of a cat vs. a dog would provide no useful insight of the models performance, as their shapes are relatively similar and given the nature of NCAs, the challenge is not set by the type of shape, but rather the complexity of the shape. Complexity is affected by size. Models containing a high amount of voxels are harder to successfully grow than smaller models. Complexity is also affected by a model's shape in terms of its geometry – a square cube is easier to grow than eg. an octopus. Some target shapes are manually created by adjusting the values of a 3D-matrix manually. This works fine for small figures such as Figure 4.6a. Symmetric figures like Figure 4.6b can be created efficiently with an algorithmic approach to adjusting matrix values. For complicated replica models like 4.6d, 4.6e and 4.6f, the process is as follows:

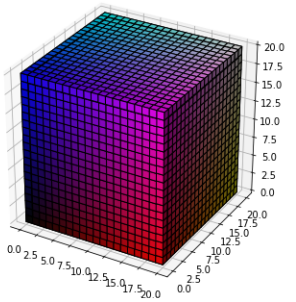
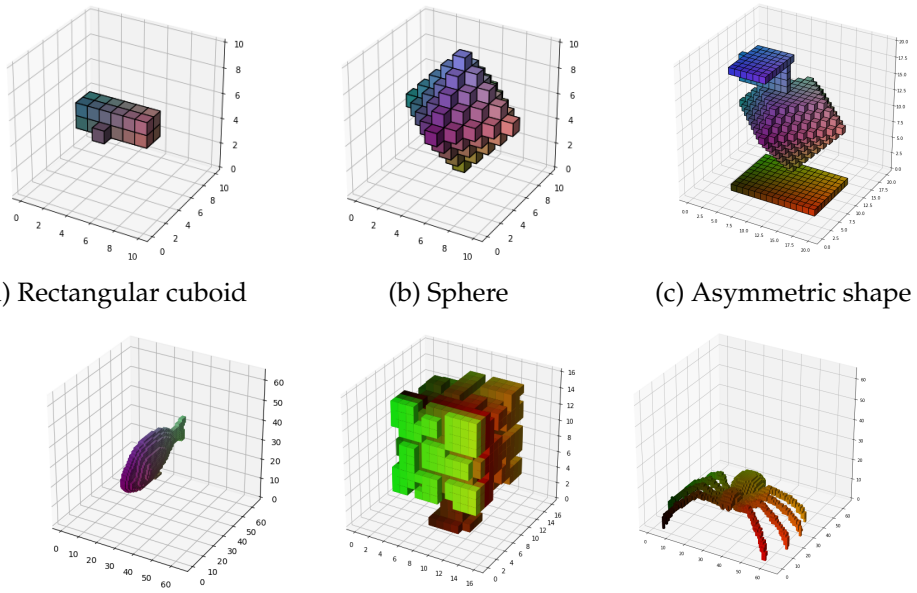


Figure 4.5: Color distribution example in a 20x20x20 voxel space

- Using a wide range of online resources, high detail 3D models of various formats can be downloaded, typically as .obj, .stl, Blender or Cinema4D.
- Convert model into .binvox file format. This voxelises the model, and the new models resolution is based on the chosen canvas size. A small canvas will force the model to be represented by few voxels. Complex figures such as the spider in Figure 4.6f can naturally not be represented by e.g. a canvas size of 5x5x5 as the model is simply too complex to be modelled.
- Read the .binvox file using a publicly available python module called binvox_rw, to represent the voxel data as a dense 3-dimensional numpy matrix.
- Iterate through the 3D matrix and fill all internal "holes" in the matrix. There should be no (dead cells) inside a closed loop of alive cells. Some 3D models only describes the outer shell of the object, which

can leave empty spaces. We want a solid model for simulating morphogenesis.

- Extend the matrix to hold RGBα values for each voxel entry.
- Assign RGBα values.



(a) Rectangular cuboid (b) Sphere (c) Asymmetric shape

(d) Fish (e) Tree (f) Spider

Figure 4.6: Target shapes. Figures (a), (b), (c) are created to test the framework with increased complexity, scaling both in canvas and model size as well as geometry. Figures (d), (e), (f) are created following the process described above, and are chosen to each represent complexity in a different way.

4.3 Neural network architecture (Update network)

The structure of the neural network generating the update rule is kept static through training of the different models. Adjusting hyperparameters such as composition of hidden layers and the sizes of each layer would be beneficial for increased performance, as differently sized target shapes do not necessarily share an optimal set of neural network hyperparameters. However, there are more components in play than just the NN, and the structure of the NN itself falls outside the scope of this thesis. Our approach is therefore to keep the same architecture during the various experiments, unless training completely stagnates, in which scenario expanding the size of the network would be a reasonable measure. Specifically, the number of time steps the CA is iterated for each training step is instead calculated and adjusted for each target shape, see Experiment 1. Figure 4.1 shows the network's layers in the second block. Specifically for the update

network, the major change we implement to the architecture proposed by Mordvintsev et al. is the type of layer layer used. As a result of the added dimensionality, we use spatial convolution over volumes (instead of images), referred to as dense 3D convolution in Figure 4.1. The first layer has 128 output filters in the convolution, a kernel size of 1 and stride of (1, 1, 1), followed by Rectified Linear Unit (ReLU) activation function, a widely used activation function for deeper neural networks [7]. Since the next layer is also the last, it has an equal amount of filters as the cell state vector has channels (16), without an activation function as a result of the incremental nature of the update rule. ReLu acts as a ramp function and would prevent subtraction during update of the cell state vector. We note that the same choice of using 3D convolutional layers were made by Sudhakaran et al.[27], which is verifies the design choice, although the options are limited in this scenario and the chosen solution is trivial when extending dimensionality.

4.4 Perception

Perception refers to the process of gathering data about a cell's neighborhood, so the data can be fed through the neural network to update the cell's state. The type of neighborhood and its range is a key factor of the perception step. The Moore-neighborhood displayed in Figure 2.2 can be extended to 3 dimensions with the following definition:

$$N^{3D-M}_{(x_0,y_0,z_0)} = \{(x,y,z) : |x - x_0| \leq r, |y - y_0| \leq r, |z - z_0| \leq r\} \quad (4.2)$$

where r is the range of the neighborhood for any given cell (x_0, y_0, z_0). Figure 4.7 displays a 3D Moore-neighborhood for a cell in position $(1, 1, 1)$, with a range of 1, which is the neighborhood used in this architecture. This means that each cell's state is a function of the 26 neighboring cells.

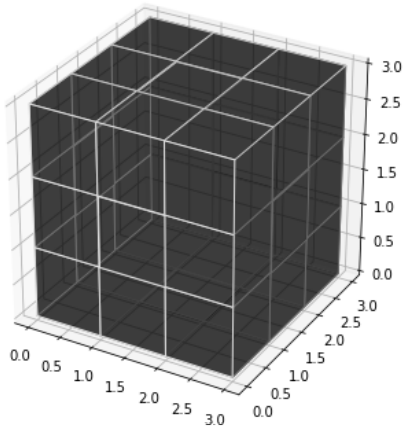


Figure 4.7: 3D Moore-neighborhood with a radius of 1, consisting of 26 cells

To perceive the neighborhood in 2D, Mordvintsev et. al [17] uses convolution to essentially measure the differences of what is to the left, right, over and under each cell. This is done using a 3x3 kernel of classical Sobel filters [25] to estimate partial derivatives. The two filters,

$$G_x \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \text{ and } G_y \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

extends to three dimensions:

$$[h'_x(x,y) = h'(x)h(y)],$$
$$[h'_y(x,y) = h(x)h'(y)]$$

$$\Updownarrow$$

$$[h'_x(x,y,z) = h'(x)h(y)h(z)],$$
$$[h'_y(x,y,z) = h(x)h'(y)h(z)],$$
$$[h'_z(x,y,z) = h(x)h(y)h'(z)]$$

and on an early stage of our model, convolution using 3D Sobel filters were implemented. During this stage, several parts of the architecture were yet to be fine tuned, and the model did not seem to learn during training. At this stage, the Sobel filters were replaced by a different operation, and as a result of other changes being made over the same time span, it is unclear to which degree the Sobel filters were sufficient for perceiving the neighborhood, although in theory they should work excellent if correctly implemented.

The static Sobel filters were replaced by a learnable kernel using depthwise 3D convolution, see Figure 4.8. While 3D convolutional networks see broad use in a wide range of 3D computer vision tasks [15, 23, 26, 28], their memory usage and computational cost becomes a bottleneck, and the performance comes with a trade-off of latency[35]. When considering scaling an NCA to a high level of operating cells, the computational cost of performing 3D convolution is likely to at some point outweigh its benefits. Although performance is not the focus of the proposed architecture, lessening the computational cost without reducing performance well justified.

The decision of not using static Sobel filters was further verified by the 3D convolution used at the perception step in [27]. "A 3D depthwise convolution splits a single standard 3D convolution into two separate steps, which would drastically reduce the number of parameters in 3D convolutions with more than one order of magnitude [35]". The backend framework used to model the NCA (Tensorflow, see Section 5) does not yet offer built in functions for depthwise 3D convolution, meaning the implemented solution may not be optimized equally as e.g. 2D depthwise convolution, which is built in. The implementation used is based on a part of MobileNets, proposed by Howard et. al [11], which uses depthwise separable convolutions to build light weight deep neural network.

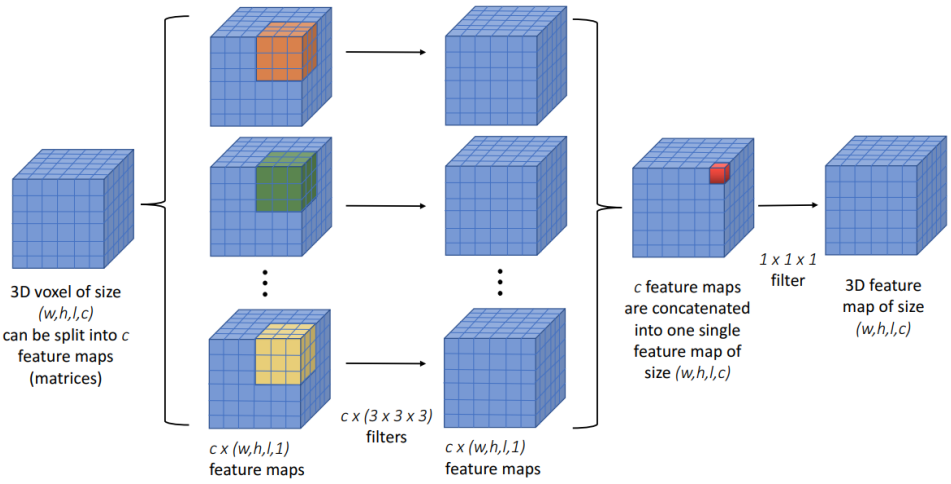


Figure 4.8: Depthwise 3D Convolution. Source: [35]

Figure 4.1 shows the perception network in the first box. A kernel size of 3x3x3 is used, with a stride of 1, and a depth multiplier of 3. This produces an output of size $[batch\_size, dim_x, dim_y, dim_z, channels * 3]$, where dim_{xyz} is the size of each axis, which denotes each cells *Perception vector* - a concatenation of each cells state and its neighborhood. This output is then propagated through the NN generating the new states for each cell.

4.5 NCA training iteration step-by-step

The following section goes through training the model step-by-step. This process also includes the steps of how simulation of a pre-trained model would take place, since this iteration takes place within the training process.

1. Initialisation

    - A target shape is selected and a seed matrix of shape $[size_x, size_y, size_z, channels_{cell}]$ is initialised. The xyz-dimensions in the first three axes matches the dimensions of the chosen target shape. $Channels_{cell}$ is the number of channels in each cell, including the RGBα channels. All entries are set to zeros, except the seed voxel, which is set as alive by assigning an α value of 1, and its hidden channels are set to 1. An appropriate location for the seed is set manually, usually located in the center of the canvas. Some models can require a different location for the starting seed based on its shape, e.g. a starting seed on the "floor" with z = 0.
    - A sample pool is filled with replicas of the same seed.
    - An *iteration range for time-steps* used during evolving the CA is defined, see Section 6.1

2. Stability and regeneration measures. This is functionality proposed by [17], and also seen in [27]. It can be enabled during training if we want the model to be able to remain stable for any give time after it reaches its target shape. The regeneration measures refers to measures improving a models capability of regrowing cells that are removed during simulation by external factors (damage). We refer to the stability measures as pooling (the sample pool mentioned in the initialisation step above), and regeneration as damage training.

   A batch is sampled from the pool. Highest loss sample gets replaced by the original seed, preventing catastrophic forgetting, as over time the pool will end up consisting of the end-states of previous training steps, and we want to ensure the functionality of growing from a seed state is not forgotten. Lowest loss samples gets damaged by deleting random sections of voxels. A set amount of randomly selected locations gets a sphere-shaped selection of voxels' channels set to 0. This adds damage to previously grown models, and the model now learns to grow to the target shape a from a broader scope of states.

3. The following steps are repeated n times, where n is sampled from the *iteration range for time-steps*.

   (a) The batch of shape $[batch, size_x, size_y, size_z, channels_{cell}]$ is forwarded to the perception network described in Section 4.4. This outputs a matrix of shape $[batch, size_x, size_y, size_z, channels_{cell} * 3]$, which denotes each cells *perception vector*.

   (b) The matrix of perception vectors is fed through the neural network described in Section 4.3.

   (c) 50% of the cells are incrementally updated, while the rest have their update discarded.

   (d) Cells with with an α value lower than the threshold for being alive are cleared by having their hidden channels reset to 0.

   (e) The batch now consists of the updated cells. Each entry in the batch has now reached time step $n + 1$

4. A loss is calculated by comparing the matrix generated by the NCA to the target shape. Gradients are computed and the weights of the network are updated.

5. The batch is committed to the sample pool.

6. Steps 2 to 5 are repeated for a set amount of training steps.

4.6 Update

Whereas perception refers to the process of gathering information from the surroundings of a cell, update refers to processing this information to generate a new state, and the operations applied during this stage. The

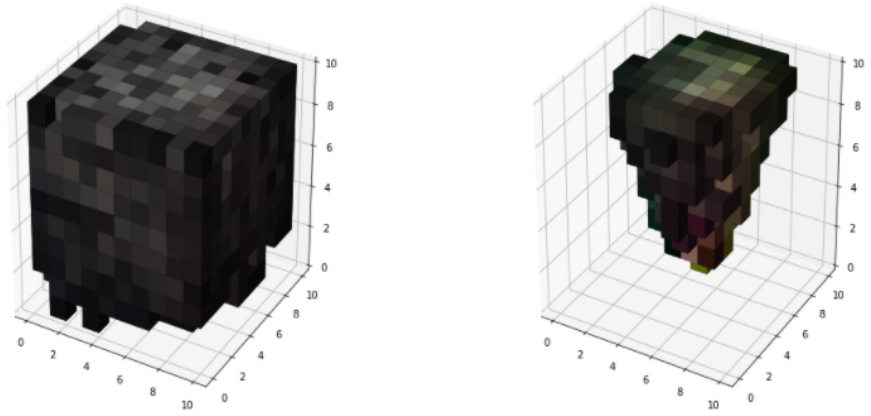
logic behind updating cell states follows the the established method seen in [17, 21, 27], and will therefore only be briefly described. The output of the neural net generating the new cell states goes through a few simple operations to ensure the desired CA functionality. A selection, randomly sampled from a uniform distribution, lets half of the cells be updated while the rest have their update discarded as a form of dropout to prevent overfitting. This also satisfies the aspect of not having a global clock in biological morphogenesis. The incremental update method is heavily inspired by residual neural networks [9].

4.6.1 Loss

Loss is calculated using L2-loss, defined as:

$$L2 = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2 \tag{4.3}$$

The squared differences between the target shape and the predicted shape are summed, and the optimizer, Adam [14], which is an extension to stochastic gradient descent, aims to minimise this loss by updating the weights and biases through backpropagation. The Adam optimizer is computationally efficient with a low memory requirement, and scales well to high amounts of data and parameters. The hidden channels are naturally omitted when calculating loss, as they do not exist in the target shape - nor do we have any interest in their values. The NCA has finished growing when $time\,step\,t = total\,time\,steps$. A trained model will reach and stabilise at the target shape at this step. The loss should be low, and if further trained, stable. For an untrained model, the grown shape will at the first training batch be completely random and just be a result of the initialised weights in the network. When different shapes emerge during training, the ones with more similar shape and colors to the target shape will get a higher score (lower loss), and the weight and biases of the network will be adjusted to generate shapes closer to these. Loss is calculated as a mean of a batch with the dimensions $[batch\_size, size_x, size_y, size_z, 4_{RGB\alpha}]$, where a batch either consists of identical single-cell seeds, or if pooling is enabled, the batch consists of a combination of a seed, and a selection of states sampled from models grown during previous training steps. The diversity within a batch is caused by the stochastic cell updates, causing the models within the same batch to grow differently.



(a) Untrained: The grown shape and color is completely random. Voxels have grown to fill almost the entire canvas, and no colors have emerged.

(b) The model has been trained a few steps further, and the rule table has been adjusted slightly, making a move towards the target shape.

Figure 4.9: A model in the early stages of training. The target shape can be seen in Figure 4.6b. This illustration is provided as an example of how the NCA evolves based on a random rule set in the early stages, which gets refined and adjusted towards the desired rule set during the training steps.

When adding functionality for guided morphogenesis and oscillating behavior of a shape, the way loss is calculated changes, see Section 4.8.

4.7 Time steps

Throughout this thesis, *time steps* refers to the number of times the NCA is updated. For each time step, all perception vectors are calculated, and all cells are updated based on the output of the NN. Figure 4.1 shows the process going from time step t to time step $t + 1$. When initialising training, time steps is given as a range. Each training step sets the amount of time steps by sampling from this range. The model evolves as time steps increases. In time step 1 the canvas has only the seed, and in the last time step the model should be fully grown. A trained model must be run in simulation for roughly the same amount of time steps as it was during training, unless measures for making the model stable given any time step was taken during training. If trained with a large interval t, the model becomes more stable, as it is guided back to its target shape for longer runs where the model might become unstable. The downside of this is training time and computational cost, making training a model with an excessive amount of time steps hard. Section 6.1 elaborates this further.

4.8 Guided morphogenesis and oscillating behavior

So far the aim has been to simulate morphogenesis - the process of growing from a seed to a stable shape, which focuses on the growth process between

the start and end shape. Oscillating behavior marks a shift in the objective, and introduces *behavior* as a goal. Oscillation is a central concept in traditional CA's but is not yet seen implemented in NCAs. Rather than having one set target shape, it aims to reach multiple target shapes in a defined order, and repeating this indefinitely. This is a novel proposal not seen in other work in this field, and can be divided into two sub-goals:

1. Guided morphogenesis: Reaching more than one target shape in a set amount of time steps (checkpoints).

2. Oscillation: Repeated cycling through all target shapes for an infinite amount of time steps (oscillating).

Sub-goal 1) offers the opportunity of adding what can be considered as checkpoints during morphogenesis, forcing the model to go through certain stages. With this we introduce the term *guided morphogenesis*, which refers to the process of interfering with the growth stages between seed and target, by introducing constraints in form of checkpoints the model must reach before reaching its target shape. To clarify with an example, if simulating growth of a salamander, we could add sub-goals in between the seed and the target shape. These could be inspired by e.g. scientific observations of how a salamander grows. Let us say its body and head emerges first, followed by its legs, and at last its tail. To force the simulated morphogenesis to grow in a similar manner, we can add the stages of body and head as a sub-goal, followed by body, head and legs as another sub-goal. The seed would still be the same singular cell, and the target shape would still be the complete salamander with all limbs, see Figure 4.10 If using a NCA to simulate growth of physical objects, this functionality can be used to set constraints such as growing bottom layers and support elements first.

Figure 4.10: A (2D) model with 2 checkpoints between seed(left) and target shape(right).

Sub-goal 2) aims to utilize more of a CA's dynamic properties by broadening the scope from just a set amount of time steps, to running the simulation indefinitely. This allows us to create a system exhibiting a movement beyond just one final target shape. For this part, we take a step away from complex shapes and work with small, simple shapes as a proof of concept. To create a stable repeating pattern – an oscillating motion varying in time, a pillar is used, with the aim of having its height rise and shrink.

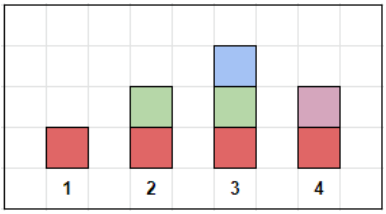


Figure 4.11: Example of the steps in a model of oscillating behavior. The 4th state has a different color of the mid cell, to indicate whether the model is moving downwards or upwards. The next step after step 4 is 1, creating a loop.

The example figure 4.11 oscillates like a sine wave. It is displayed in 2D just for visual purposes, but can be viewed as the variation of the z-axis in a three-dimensional model. To achieve an oscillating motion for any range of time steps, the sub-targets must be arranged in a way allowing a looping pattern.

4.8.1 Loss over multiple target shapes

Checkpoints and motion is implemented by altering how loss of a model is calculated. Previously, loss has been calculated based on the final state of the NCA for each training step. Instead of discarding all previous states, these intermediate states are now stored and compared to the respective sub-goals, and the loss is a mean of their combined losses. This forces the NCA to pass through all the given target shapes, and extends the dimensions of the input to the loss function to $[goal\_shapes, batch\_size, size_x, size_y, size_z, 4_{RGB\alpha}]$, and as a result a lot more complex to train than a single target shape.

At which time step the different sub-goals should take place is a design choice set by the user and is largely dependent on the range of time steps the NCA is simulated between each loss calculation. For creating an oscillating behavior, we use an even distribution where the chosen time step for a sub-goal number s is $\frac{total\ time\ steps * s}{total\ sub-goals}$. As the NCA evolves, the state is saved every time it passes through one of the time steps corresponding to a sub-goal. When paired with the varying time step range, and the pooling functionality previously discussed, the process becomes a bit more complicated. Since the total number of time steps the NCA iterates changes (within a given range) for each training step, the sub-goals will not land on the same time steps for every train step. This means it will get contradicting results in terms of the exact time steps the model should be in the sub-goals. Furthermore, the pooling functionality stores previously high scoring models, in which are sampled and used as starting point (seed) for further training, but for an oscillating model, the saved state could be any of the sub-states, if the time step range is large enough. To handle this an evaluation function is added. The function compares the starting state of a model sampled from the pool, with all possible sub-models. The sub-model with the lowest loss determines where in the

sequence of sub-models the current state is. We can then adjust the starting point in the sequence of sub-models, so that each subsequent model has its loss calculated to its respective model, see Figure 4.12 for an example of 5 sub-states. When training a non-oscillating multi-goal model, the pooling function should be omitted, due to the fact that it will not be able to revisit the previous states, and so it should be trained from a starting seed each iteration. This comes at the cost of not benefiting from the pool's ability to refine the dynamics of well trained networks, and will result in less stable models. To compensate for this, the time step range could be increased.

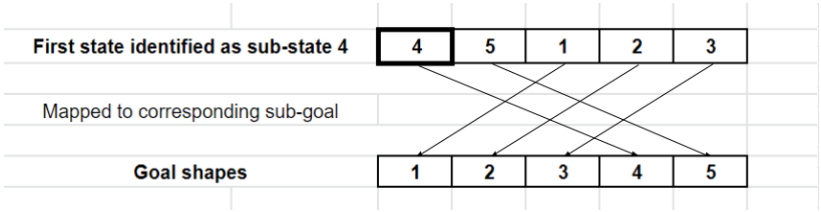


Figure 4.12: The identified order of states are mapped to their respective sub-states. The first state defines the location in the sequence where the loss is calculated. In this example, the identified state of the model was state number 4. This lead to the next state being compared to target shape 5, the next to target shape 1, and so on.

Chapter 5

Software setup

This chapter lists the hardware, software and various libraries used in the thesis.

5.0.1 Software

The codebase for this thesis is written in Python. Based on experience, this was a natural choice when aiming for proof of concept for academic purposes. The deep learning API Keras is used with Tensorflow 2.8.0 as backend.

Python packages

3D models are internally represented by Numpy matrices, and visualised in 3D using Matplotlib. This allows for plotting a voxel representation of a matrix, with assignment if of RGB float32 values to each voxel. The plot also offers shading and opacity for better visual quality. Moviepy is used to generate video examples of the simulations, and Pickle is used for storing loss history over separate training runs.

3D models are converted from their respective file formats to .binvox using an online "voxelizer"[1], and further converted to a binary representation in Numpy matrices using a python module called binvox_rw.

5.0.2 Hardware

The experiments were conducted using a Tesla P100-PCIE-16GB GPU.

[1] https://drububu.com/miscellaneous/voxelizer

Chapter 6

Experiments

This chapter will explore the functionality and performance of the proposed 3D NCA. First a universal method for determining a time step range will be derived. The 3D NCA will be tested step-by-step, where each experiment builds upon the previous one, adding complexity and testing different aspects. We start with verifying its ability to grow just a shape, then add colors, size, asymmetry and complexity to the target models, and end with exploring movement with guided morphogenesis and oscillation in the final experiment.

The very first and main experiment was to identify which architectural changes would be necessary to modify the already established neural cellular automata architecture in [17] into a higher dimensional one. While doing so, a central goal was to only modify the architecture where it was necessary, and keeping other aspects unchanged. This was motivated by several factors, as they are not competing architectures, but rather one being an extension of the other, a similar logic, structure and architecture would lay the foundation for easily comparing how the increased dimensionality itself effected an NCA's ability to grow into a shape. It would also be useful when comparing how the increased dimensionality affected the computational cost. Vast different architectures would add a large amount of extra variables, making these factors more complicated to measure. Furthermore, Mordvintsev et. al [17] achieves a certain standard of success for their model, making it a reasonable benchmarking goal in a narrow, but fast evolving field of study.

6.1 Experiment 1: Time step range

Experiment 1 aims to derive a universal method for determining a suitable time step range

To our knowledge, similar work in this field has yet to provide any guidelines or insight on how to choose a time step range when training an NCA, or the reasoning behind their used ranges. This experiment aims to find a method for determining an appropriate time step range to use during training of a model. This is motivated both by providing a method

for future work in this field, as well as benefiting from a consistent way of choosing time step range for the following experiments. In short, time steps is how many times the CA updates the cell states. Since the NCA benefits from having a variation in the amount of time steps used during training, the time step range refers to the min/max time steps used, see section 4.7 for further definitions and context. Time step range is defined as two integers in a range $[n, m]$. When referring to the time step range, we refer to the size of the values n and m, and not the spacing between the two. Although both are of interest, we focus on the former unless specified otherwise. Assumptions made for this experiment:

- We consider an ideal time step range to be the lowest possible range that does not negatively impact the loss, but speeds up training due to no unnecessary update steps. Note that this assumption forces the morphogenesis to happen in as few time steps as possible.
- Canvas and model size is equal. While a model can be way smaller than the canvas it is placed within, the two are assumed to have the same height, width and depth. More specifically, the biggest dimension is assumed to be the same, should the canvas, or more commonly, the model, not be of equal size in all dimensions.
- The NCA uses pooling during training. This means it is trained to regenerate, and affects how training occurs by using previous end-states as seed, instead of always starting from a single seed-state. This also removes the incentive to have a high spacing between n and m in the step range for stability purposes.
- Seed is placed in center of all axes unless specified otherwise.

The main factors we consider affecting time step range is model size, and complexity of the model. The next sections will take a closer look at the correlation between these.

Model size

Model size and canvas size are used interchangeably in this chapter, and refers to the height, width and depth. The following examples will use a two-dimensional canvas instead of a three-dimensional one, for easier visual demonstration. All the concepts are extendable to three dimensions. First we take a look at how fast a model can grow. The cells can maximum grow one cell per neighbor-spot, per time step (this applies to three-dimensions as well). Figure 6.1 shows on a 2D pane the minimum amount of time steps (blue numbers) the NCA would need to reach each cell on a 10x10 canvas. As a result of the even numbered canvas dimensions, the seed (green cell) can not be placed exactly in the center, but is located in [5,5]. This causes the shortest path to all cells in the top row [1:10, 10] and right-most column [10, 1:10] to be 5, which is one more than in the opposite directions. Note that this, and the following examples, uses a Moore neighborhood. Let us consider a target shape marked by the black

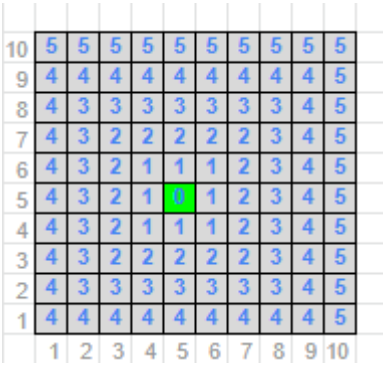
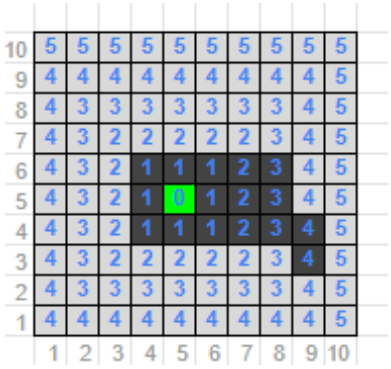


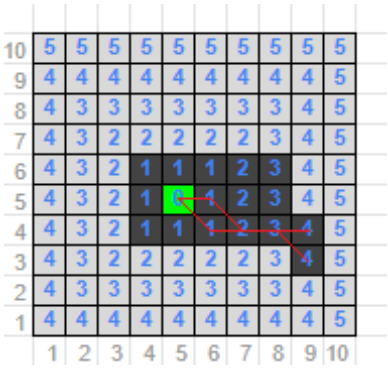
Figure 6.1: Time steps (blue) needed to reach tiles from seed (green cell) on a 10x10 canvas.

cells in Figure 6.2a. The shortest path to the cell [9,3] is, as marked by the red line in Figure 6.2b, and as shown by the blue number, 4. We can visually verify that this cell (and its neighbor [9,4]) is the part of the model that is furthest from the starting seed. This means the model would need at least 4 time steps to reach this cell. We call this the critical path. By the time the model has reached this cell, all other cells in the target shape can be reached. We use this as a starting point for defining a time step range.

The critical path is relevant for simulations where the targets does not occupy the entire canvas. This applies to all targets with a more complex geometry than the canvas they are grown on, whereas for e.g. a 2D CA growing a picture where the target graphics covers all pixels, the critical path will always be set by the canvas size.



(a) A target shape marked by black cells



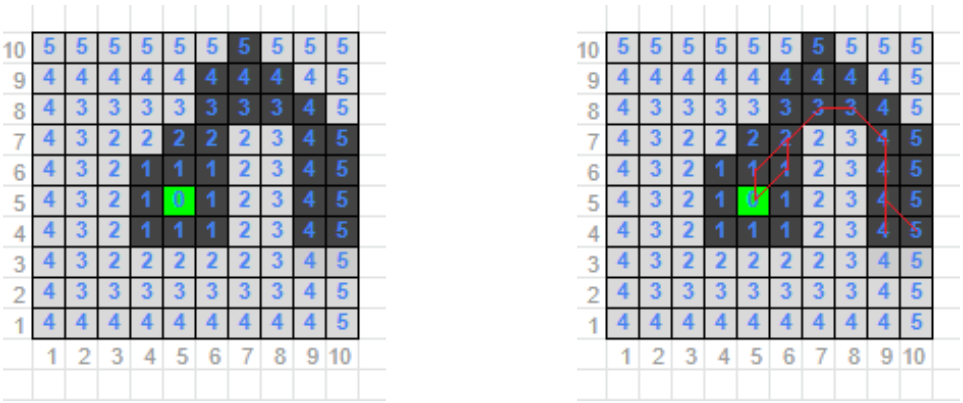
(b) The critical path marked by red lines has a length of 4. The lines split where there is multiple routes with equally short path.

Figure 6.2: A target shape and the critical path

The next thing we need to take into consideration is the stochastic update of cells. At each time step, only 50% (set as a design choice) of the cells actually get their states updated. The other 50% get their update

discarded. This affects the number of time steps needed to reach each cell. If we call the percentage of cells that gets updated for *cell fire rate*, we can approximate that with a cell fire rate of 0.5, the time needed is at least doubled. We add a safety margin of 10% to compensate for the chances of an over-representation of discarded updates in the critical path. The critical path now has a length of $\frac{critical\ path\ length}{cell\ fire\ rate - 0.1} = 10$.

Model complexity



(a) A target shape marked by black cells

(b) The critical path marked by red lines. The lines split where there is multiple routes with equally short path.

Figure 6.3: A more complex target shape and the critical path

The target shape in Figure 6.2 has a direct path from the seed to the furthest edge, meaning the length of the critical path will be the blue number in each cell. Depending on the target shape, this is not always the case. Figure 6.3a displays such a model. From the starting seed, the furthest edge is found in cells [9:10, 4], but the target shape has a gap of "air" in the direct line between the seed and this cell. Figure 6.3b displays the critical path. The path goes through the shortest possible route, while staying "inside" the target shape. This causes the critical path to be longer than the numbers denoted in blue. Instead of 5, Figure 6.3b has a critical path of 8, as this is the number of cells the critical path passes through. We refer to this as a result of *model complexity*: The target shape is formed in such a way that the critical path to furthest edge is forced to take a detour, and is not equal to, but greater than the blue numbers.

It is worth noting that an NCA has the ability to both grow new and remove old cells. This means that it does not necessarily grow the way the critical path has been marked in Figure 6.3b. The NCA could learn to first fill the gap in cells [7:8, 4:5], making it reach the furthest edge in the bottom right corner, and then the cells in the gap would get removed again, step by step, with the end result being the target shape. Depending on the shape and degree of model complexity, this would still add extra

time, and we estimate that overall, this roughly matches the time used with the critical-path approach. Precisely determining the critical path and its length is more complex to solve for a three dimensional shape. We leave this as a future research goal, and settle for an approach based on visual inspection of the model. A variable *complexity factor* is introduced in the critical path length equation, and the critical path length is replaced by the direct distance from seed to furthest edge:

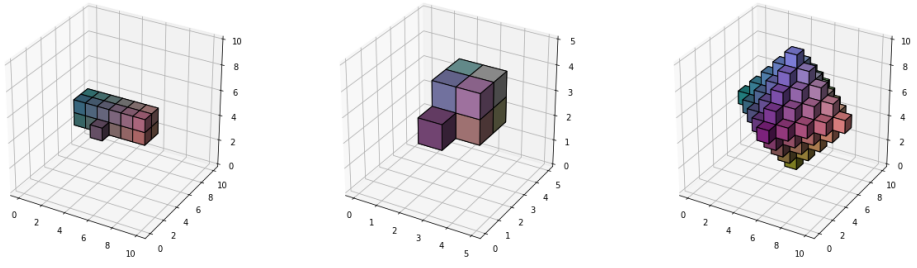
$$Minimum\ time\ steps = \frac{seed\_to\_edge_{distance} * complexityfactor}{cell\ fire\ rate - 0.1} \tag{6.1}$$

Complexity factor is manually set based on visual inspection of the target shape. While having a manual factor is not ideal, we lessen the scope of the manual factor with this approach, as it leaves just a part of the equation as a manual factor, opposed to the entire estimation which so far has seem to be the case in other work.

- Complexity factor = 1+β
  - where β is a real number, and describes the degree of detour relative to the shortest distance between seed location and furthest edge.
  - β = 0 means the shape takes no detours, and length of critical path is equal to the shortest distance from seed to furthest edge.
  - β = 0.2 means the detour adds a slight curve to the critical path, adding 20% to the distance.
  - β = 1 means the detour doubles the length of the critical path.
  - β = 2 means the detour triples the length

Figure 6.4 displays a collection of models with an estimated β of 0, as none of the models have a shape forcing a detour in the critical path. The models in Figure 6.5 however, each have a suggested β value in their respective caption. Note that all models are assumed to have a seed located in their center. This differs in Figure 6.5c, as this model has no voxels in the center, and therefore the seed must be placed on either side of the hole. This means the critical path has a slight curve to it, as it has to pass around the center hole. The suggested β value for Figure 6.5b is caused by the slight curing of the spider's legs.

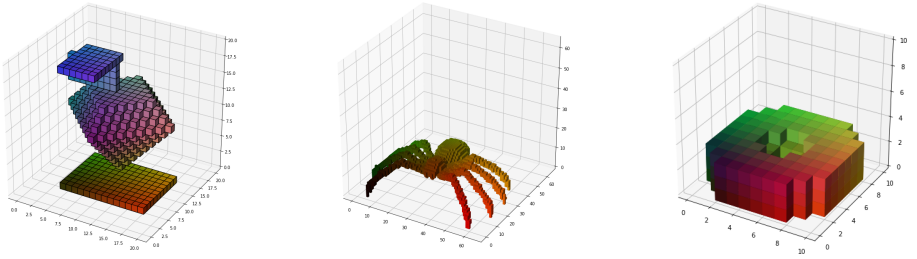
The final factor we take into consideration when determining minimum time steps, is the time needed to adjust colors of each cell. We observe that when a new cell first emerges, it tends to have a different color, and over the following time steps, the color gradually adjust towards the goal-color. Since a cell's state is depending on the states of all cells in its neighborhood, it makes sense that the cell is not "fully grown" and the colors are not yet correct, until all the cells in the neighborhood has emerged, and has finished updating all their channels as a result of each other. With this we introduce a constant *stabilisation time*, which is added to the minimum time steps equation to ensure all cells has time to stabilise. We default



(a) (b) (c)

(d) (e)

Figure 6.4: Shapes with a suggested $\beta = 0$



(a) Suggested $\beta = 0.5$ (b) Suggested $\beta = 0.15$ (c) Suggested $\beta = 0.1$

Figure 6.5: Shapes with a suggested $\beta > 0$

this constant to a value of 5, chosen by visual observation of how many time steps the RGB color of a cell seems to adjust after a neighborhood is established. This should be scaled up if the neighborhood has a long range, as changes in cell states further away would make an impact. Since both the color and the hidden channels are represented the same way as a real number, this should be adequate time for the hidden channels to stabilise as well.

$$Min\,time\,steps = \frac{seed\_to\_edge_{distance} * complexity factor}{cell\ fire\ rate - 0.1} + stabilisation\,time \tag{6.2}$$

While it is possible to train with the same time step for all training steps, using a range – the spacing between n and m, helps the model gain an attractor for the target shape from a broader scope of states, as it introduces variation in training alongside the stochastic dropout, and

it works together with the pooling function to increase stability of the target shape. We suggest scaling the upper range based on adding 20% to minimum time steps, as this would scale the spacing between n and m based on the size of minimum time steps. This is beneficial for larger models as the stochastic cell updates introduce a higher variance in time steps needed to reach furthest edge, and also benefits small models that update few times by adding few extra steps. When using the pooling function for stability training, the necessity of a range decreases, but it still adds value and can help increase stability further. If training without pooling, a larger range is recommended if stability is a goal. This essentially comes down to a trade-off between stability and training time, as a high upper value on the range favours stability but drastically increases the computational cost, and a low upper value decreases both.

$$Time\,step\,range\ = [minimum\_time\_steps, minimum\_time\_steps * 1.2] \quad (6.3)$$

Model	Canvas size x,y,z	Est. Seed to Edge Distance[1]	Est. Complexity Factor	Minimum Time Steps[2]	Time Step Range[3]
2 Cuboid	5x5x5	1	1	8	[8, 10]
3 Rectangular cuboid	10x10x10	3	1	13	[13,16]
4 Sphere	10x10x10	4	1	13	[15,18]
5.1 Asym. shape	20x20x20	10	1.5	43	[43,52]
5.2 Asym. colors	10x10x10	5	1	18	[18,22]
6.1 Spider	32x24x12	16	1.15	63	[63,76]
6.2 Tree	16x16x16	15[4]	1	45	[43,52]
6.3 Fish	32x32x32	16	1	45	[45,54]

[1] Direct distance from seed to furthest edge of the target
[2] Equation 6.2
[3] Equation 6.3
[4] Seed located at bottom of the model.

Table 6.1: Time step ranges

6.2 Experiment 2: Shape

Experiment 2 aims to verify the NCAs ability to reproduce a shape

We start testing the NCA with a simple case of growing a small cuboid with a height, width and depth of 2, see Figure 6.6. For this experiment we exclude colors, and only care about whether the correct shape is grown. This is done to isolate the ability to reproduce a shape, to verify the core aspect of the architecture working in a three-dimensional space. We leave the RGB channels as a part of the channels, but do not care about their values when calculating loss or visualising the model, meaning they become part of the hidden channels. The alpha channel keeps its function and dictates whether a cell is considered alive or dead. Table 6.2 lists the hyperparamters used during training.

Let us take a look at how the model evolves over time in Figure 6.8. Only the 4 first models of each batch is displayed, to decrease visual clutter, but still display the diversity within a batch. If following just one model as it iterates through the time steps, it would be hard to tell if this represents the mean or if it is an outlier. The voxels are portrayed in blue just for visual purposes.

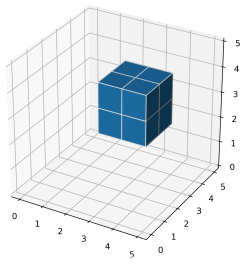


Figure 6.6: Target for Experiment 2: Cuboid. The voxels have no assigned colors, and are blue for demonstration only.

See Figure 6.7 for how loss decreases during the training steps. Loss is plotted as log10, so lower value on the y-axis means a better loss. We observe that the loss flattens out after about 200 training steps. When observing the batches in Figure 6.8, we see that after 5 training steps (b) the model has already some resemblance to the target shape. During 10 (c) and 15 (d) train steps the model grows far beyond the target shape. This is expected behavior at such an early stage in training, as the weights of the rule-generating network is randomly initialised and has no bias towards the target shape.

Already in train step 20, we observe that all examples have stabilised at the target shape. The loss graph however, shows that the loss keeps decreasing for another 180 training steps. During this time gap, most models reach the intended shape, but not all the models in every batch. Loss is calculated as a mean of all models in a batch, so during this time gap, fewer and fever models in each batch end up with a different shape than intended. The model took just over 3 minutes to train, with rendering of graphical elements slowing training significantly down. If disabling rendering of graphical elements (loss plots and 4 models rendered every 5 training steps), the train time cuts down to 52 seconds for 500 steps.

As listed in Table 6.2, pooling for stability and damage for regrowth is disabled during this experiment. Note that hyperparameters such as hidden channels, batch size and NN architecture are intentionally not scaled down for this experiment, to gain insight how computational cost scales with larger models. This experiment successfully demonstrates the NCA's ability to grow into a very simple given shape, and verifies the lowest level functionality of a three dimensional NCA.

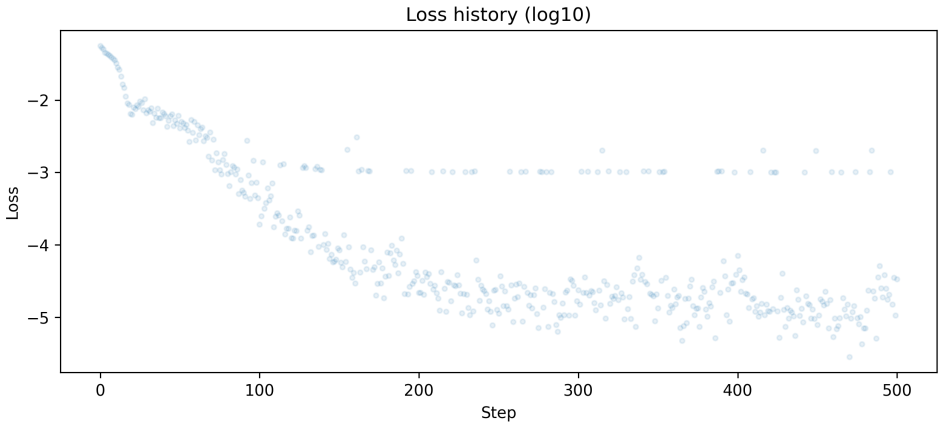
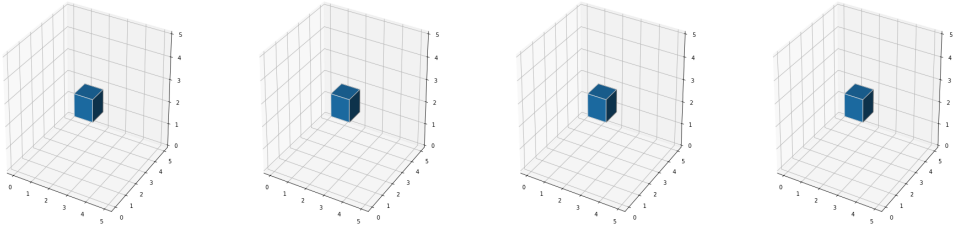


Figure 6.7: Loss, Experiment 2. After 200 training steps the model does not improve further.

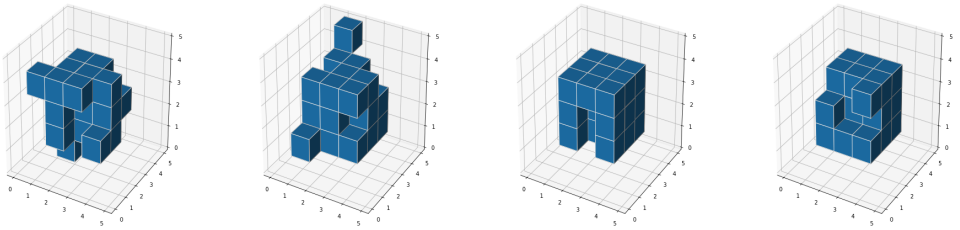
Experiment #	Train Steps	Time step range	Height width depth	Voxels	Channels	Batch Size	Pool (size)	Damage (nr./batch)	Lr.
2	500	[8, 10]	5x5x5	4	16	8	-	-	2e-3
3	4000	[13,16]	10x10x10	25	16	8	-	-	2e-3[1]
4.1	12000	[15,18]	10x10x10	129	16	8	-	-	2e-3[1]
4.2	12000	[25,30]	10x10x10	129	16	8	-	-	2e-3[1]
5.1	7000	[43,52]	20x20x20	968	16	8	-	-	2e-3[2]
5.2	5000	[18,22]	10x10x10	515	16	8	-	-	2e-3
6.1	10000	[63,76]	32x24x12	810	16	8	1024	3	2e-3[2]
6.2	14000	[43,52]	16x16x16	1151	16	8	512	3	2e-3[3]
6.3	7000	[45,54]	32x32x32	620	16	8	512	3	2e-3[4]

[1] Learning rate changes at step 2000 by a factor of 0.1 (PiecewiseConstantDecay)
[2] Learning rate changes at step 5000 by a factor of 0.1 (PiecewiseConstantDecay)
[3] Learning rate changes at step 9000 by a factor of 0.1 (PiecewiseConstantDecay)
[4] Learning rate changes at step 4000 by a factor of 0.1 (PiecewiseConstantDecay)

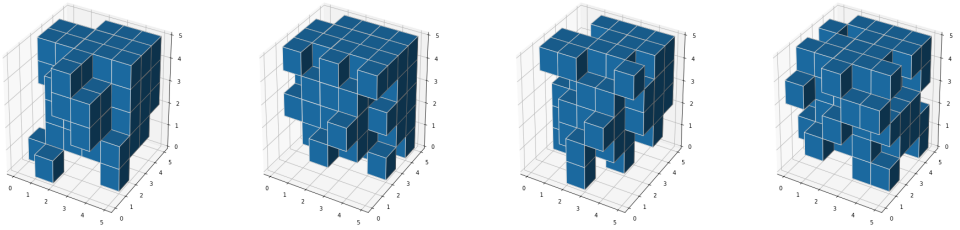
Table 6.2: Hyperparameters



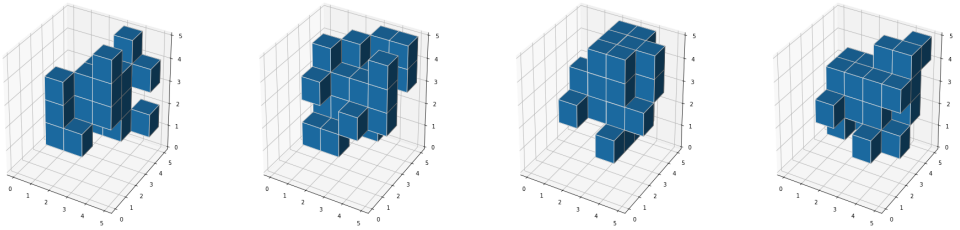
(a) Train step 0: A batch of seeds



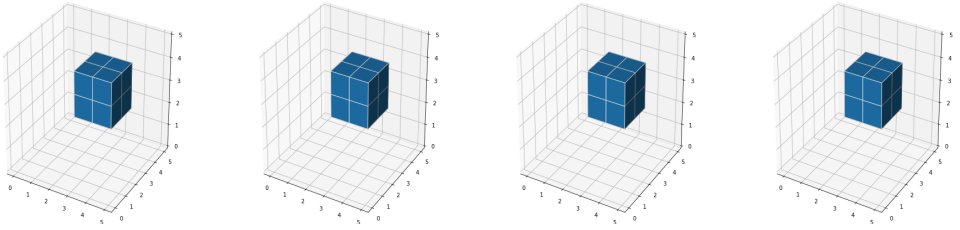
(b) Train step 5



(c) Train step 10



(d) Train step 15



(e) Train step 20 - identical to the target shape.

Figure 6.8: Experiment 2: Sample of batches in the last time step, for different (training) steps a) to e). Each row contains an individual batch of seeds being updated.

6.3 Experiment 3: Shape and colors

Experiment 3 aims to verify the NCA's ability to reproduce a colored shape, while also increasing complexity of the shape.

The next step of testing the NCA extends upon experiment 2 by adding colors to each cell. Loss is now calculated based on the 4 channels RGBα. Additionally we increase the size of the canvas to 10x10x10, and extend the target shape from a cuboid to a rectangular cuboid consisting of three times as many cells. A single cell is attached on the outside of the rectangular cuboid, intended as a check of the NCA's ability to grow small details deviating from the otherwise straight lines in the model, specially testing if the perception module is able to detect it, and if the update module is able to generate rules for growing it. The colors are arbitrarily assigned to each cell in the target-shape based on the cell's placement relative to a color spectrum.

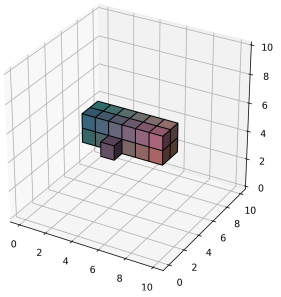


Figure 6.9: Target for Experiment 3: Colored rectangular cuboid

Hyperparameters can be found in Table 6.2. An appropriate amount of train steps (4000) is chosen based on monitoring of loss during training. See Figure 6.10 for loss history. Note that we lower the learning rate by one order of magnitude after 2000 steps, which can be observed by a fall in loss in the loss history plot. Lowering the learning rate after a while helps the NCA refine the finer details in weighting after narrowing down the search space. In an additional training run, the learning rate was lowered further after 3000 training steps, but with no effect on the loss. We observe in the loss history plot that the model gains little to none improvement in loss after 3000 training steps.

Figure 6.11 displays a forward iteration of the NCA, after the 4000 training steps. The NCA was iterated for 30 time steps, while only trained with a range of [13,16]. Each figure has its corresponding time step marked at the top. Time steps 0 through 9 is displayed, followed by the last time step 29. The final figure shows the target shape for comparison. In frame 0 to 9 we can observe the growth of cells per time step. Frames 9 to 28 is not displayed to reduce visual clutter, but the model stays stable with the same shape and colors through these time steps. We observe that the NCA is able to reproduce both the shape and colors with impressive accuracy. The fully grown model can not be visually distinguished from the target shape, neither by shape or colors. Once it reaches the correct shape in time step 9, it does not deviate in the examined time span. If ran for much longer times however, we expect the model to become unstable relatively fast, as it is trained with a short range of time steps and no pooling. The "outlier" cell placed outside the rectangular cuboid is correctly grown and further verifies both the perception and update modules ability to reproduce details on a per-cell level.

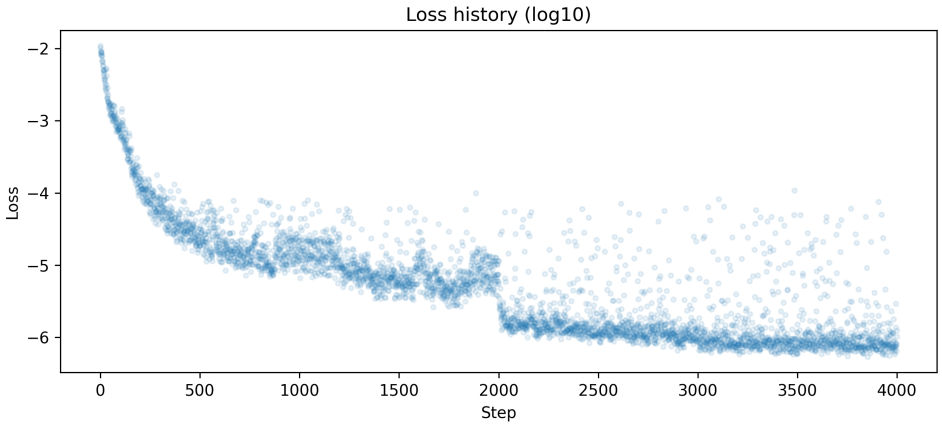


Figure 6.10: Loss, experiment 3.

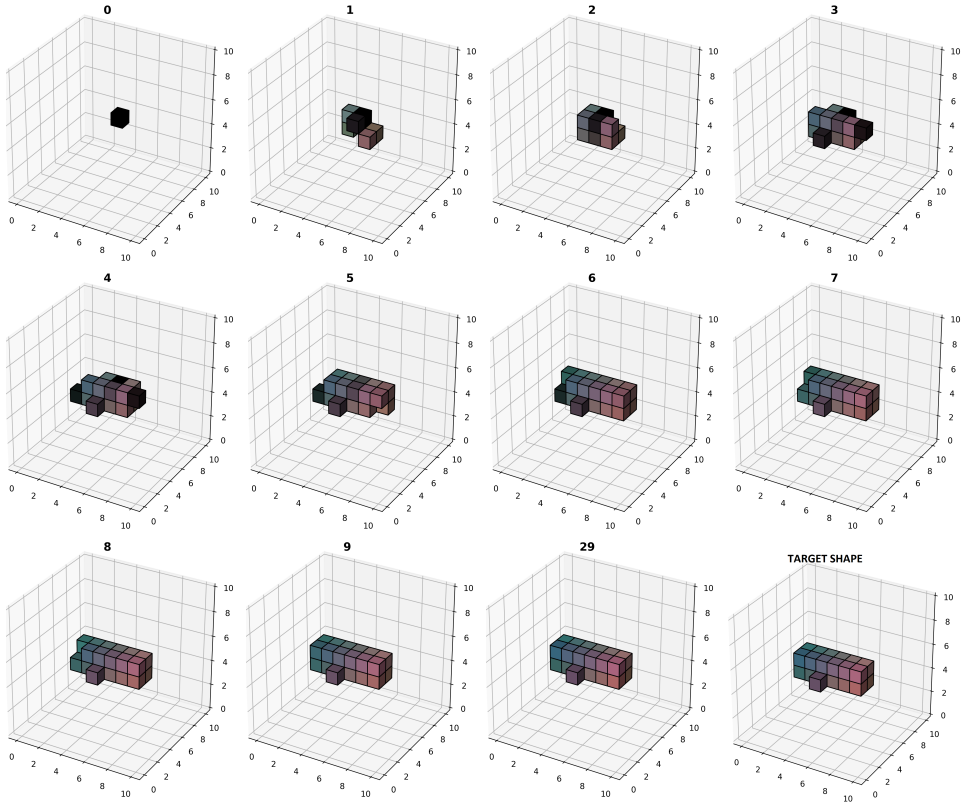


Figure 6.11: Experiment 3: Growth of rectangular cuboid. Development per time step, marked at the top of each frame. Last figure is the target shape for comparison. The model is indistinguishable from the target shape both in frame 9 and 29.

6.4 Experiment 4: The effects of time step range

Experiment 4 introduces increased model complexity, and examines the effect of adjusting time and train steps

Compared to the previous experiment, we now take a look at how the NCA performs when given a target shape with a higher complexity. To have a basis of comparison to Experiment 3, we keep the canvas size of 10x10x10 , but choose a target shape (Figure 6.12) consisting of more cells, and significantly more edges, challenging the perception module further as each neighborhood is more diverse in terms of placement of living and dead cells. As for complexity in colors, both experiments operates with a unique color assigned to each cell, and we consider this to be at maximum complexity, see Experiment 5 for further examination of colors. We train two versions of the same model, slightly differently tuned. Experiment 4.1 and 4.2 train a model with the exact same hyperparameters except time step range, which is almost doubled in Experiment 4.2. We do this to gain insight of how model performance is affected by tuning this parameter towards a higher computational cost. Longer and more extensive training does not necessarily mean an increase in model performance, and the result here will be used to evaluate Equation 6.2 derived in Experiment 1. This examination could be done extensively by comparing a broader spectre of hyperparameters and their effect, but this falls outside the scope of focus for this thesis and we leave this to future work, and instead look for an indication whether increasing this parameter has a positive or negative impact on the model.

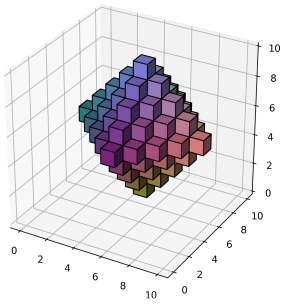


Figure 6.12: Target for Experiment 4: Colored sphere

6.4.1 Experiment 4.1

Note from the hyperparameters in Table 6.2 how the amount of voxels in the target shape scale from 25 in the previous experiment to 129 in this. The calculated time step range however does not scale equivalently, as the length of the critical path was 3 in the previous experiment, but 4 in this one, yielding only a few extra steps. The model is trained with a time step range of [15, 18]. Similar to Experiment 3, we train the NCA with a decrease in the learning rate after 2000 steps. Looking at the first 4000 steps, we see a similar trend in the two loss history plots 6.10 and 6.13. Note that the losses are calculated from different targets and their values are not directly comparable. Both benefit from the adjusted learning rate, however Experiment 3 reaches a point where the loss flattens out earlier.

An additional training run indicated that the model trained for 12000 steps can benefit from having the learning rate lowered at a later point than

step 2000. They both have a distribution of batches with higher loss spread out even in the latter train steps. We believe these to be a result of minor fluctuations in the cells' channel values causing the odd cell to become alive momentarily and then affecting the loss in a batch, or minor fluctuations in the RGB values which have not completely stabilised during the given time step range.

Figure 6.15 shows a simulation of the trained NCA for 60 time steps to examine the behavior when simulated for a longer period than seen in training. Let us first consider time steps 1 to 17. Like the previous experiments, the model successfully grows from the start seed to the target within these steps, reaching the target at approximately time step 12. During the stable period from around frame 12 to 25, the model is not visually distinguishable from the target. After this, the model exhibits the expected behavior of instability, as it has not seen these time steps during training. A few cells start to emerge, and the model starts growing exponentially into chaos. This result confirmed the NCA's ability to grow models of more complex geometry, and gave insight to how the model behaves when simulated for more time steps than trained for.

6.4.2 Experiment 4.2

As stated, we run the exact same model, and with a time step range of [25, 30]. We revisit equation 6.2 and adjust the *stabilisation time* constant from 5 to 15. The exact increase does not particularly matter for this experiment as we now focus on the effect of increasing these parameters relative to each other rather than finding their optimal value. However, the stabilisation time constant was initially set based on observations in very small models, and may be subject to change based on the outcome. In experiment 4.1 the training time was 21 minutes, while this model took 34 minutes to train. The fever train steps per second is caused by the increased number of updates of the CA for each train step. Let us compare the losses from each training in Figure 6.13 and 6.14. Since the two losses are calculated from the same target, their values are directly comparable. Note that the y-axis representing loss value is slightly different scaled in the two plots. The first model reached a (log(10)) loss of ~-4.7. The second model reached a better loss of -5.2. Both models were ran with the same learning rate which decreased after 2000 steps, and all initialisation of NN weights was seeded for reproducibility. If we compare the loss when both models are in train step 4000, the second model still outperforms the first one with a loss of ~-4.9 vs ~-4.5. In terms of loss over a batch the model in Experiment 4.2 outperforms the first one at any time step, at the cost of approximately 40% increased train time per train step. We can also observe fom the plot that the loss also has a lower variance which can be equally important. However it is worth noting that this performance measure is purely based on loss, and might not justify the increased computational cost, depending on the purpose of the simulation. Both models are in their fully grown states indistinguishable from the target, meaning if visual performance is the criteria, the higher train step range of Experiment 4.2 does not provide

any benefits over the lower one in Experiment 4.1. It is worth noting that a benefit of Experiment 4.2 is that the model stays stable for ~20 steps longer as a result of the higher time steps, but this will factor will be completely mitigated when trained with pooling and regrowth measures. On the other hand, experiment 4.1 reaches (by visual verification) the target after 11 time steps, compared to 17 in Experiment 4.2.

Ex.#	Train Steps	Time step range	Batch loss Step 4000	Batch loss Step 12000	Time	Steps to target shape
4.1	12000	[15,18]	-4.5	-4.7	21min	11
4.2	12000	[25,30]	-4.9	-5.2	34min	17

Table 6.3: Experiment 4: Results

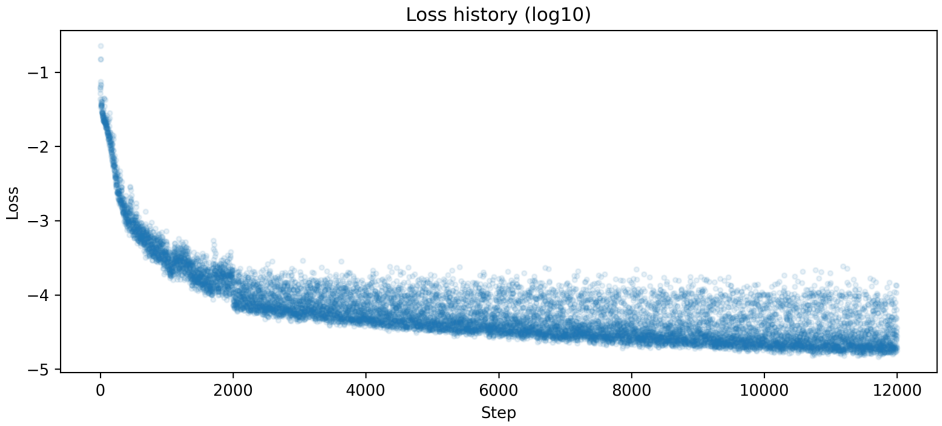


Figure 6.13: Loss, experiment 4.1

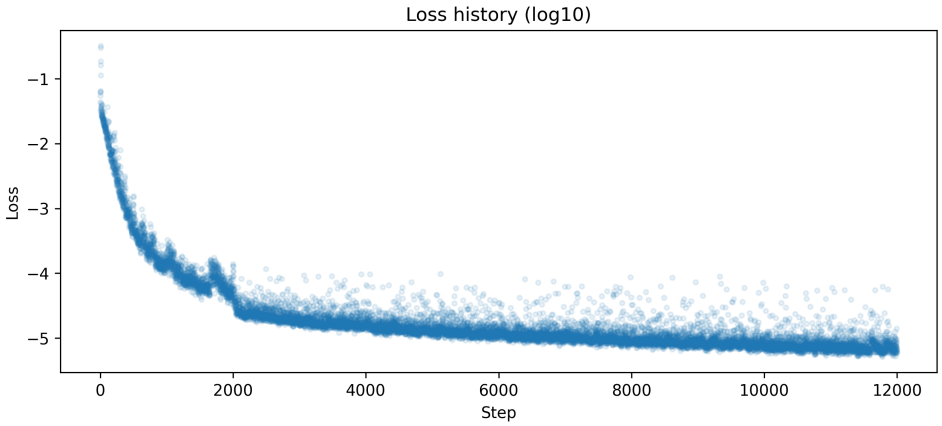
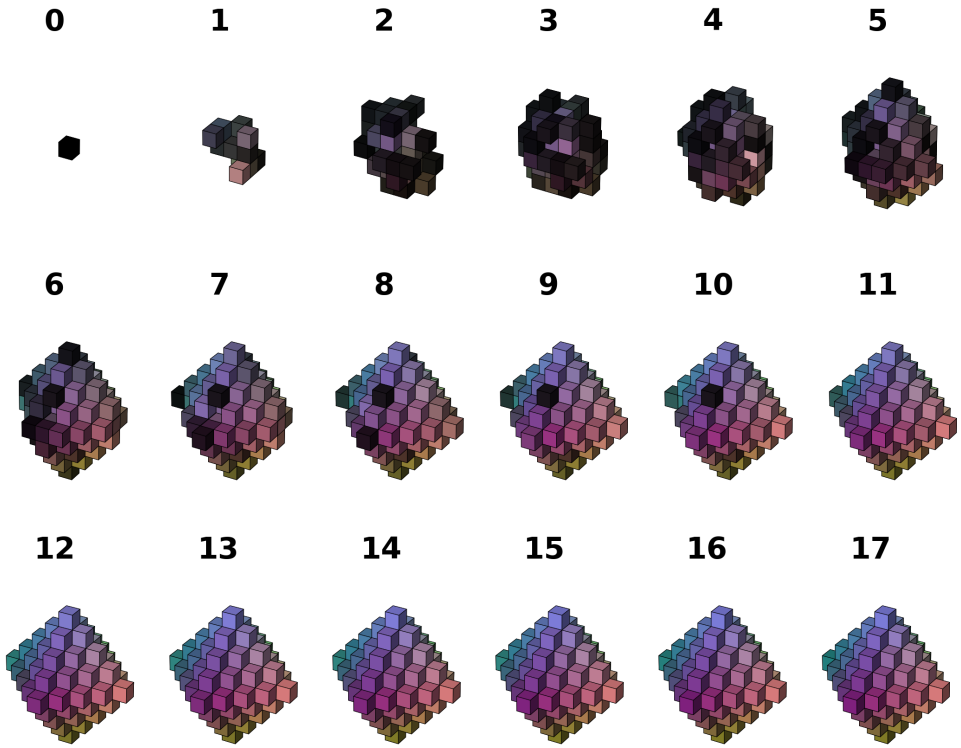
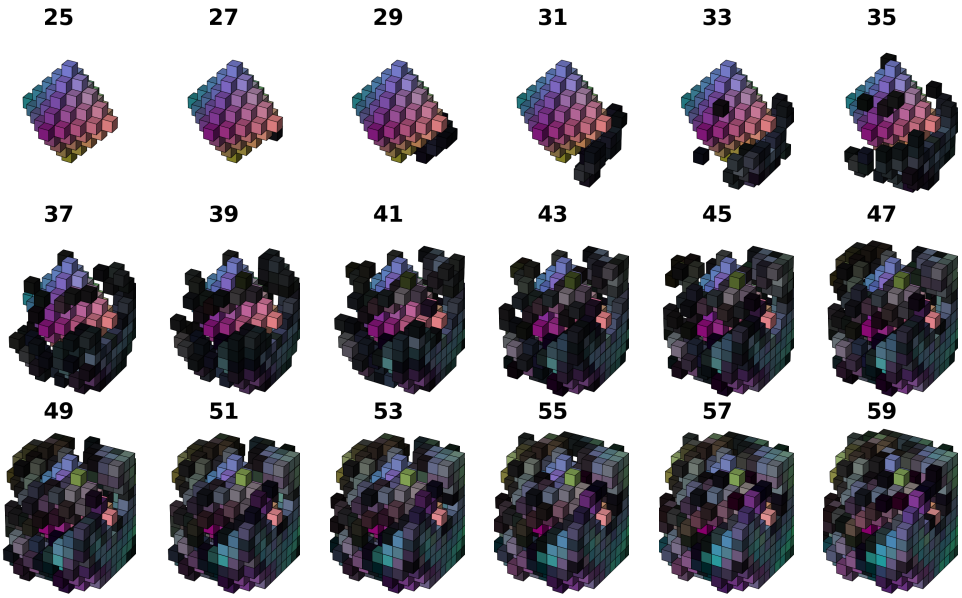


Figure 6.14: Loss, experiment 4.2

(a) Time step 0 to 17: The model grows from a seed to the target shape. Already in step 11 the model appears to have reached the target.



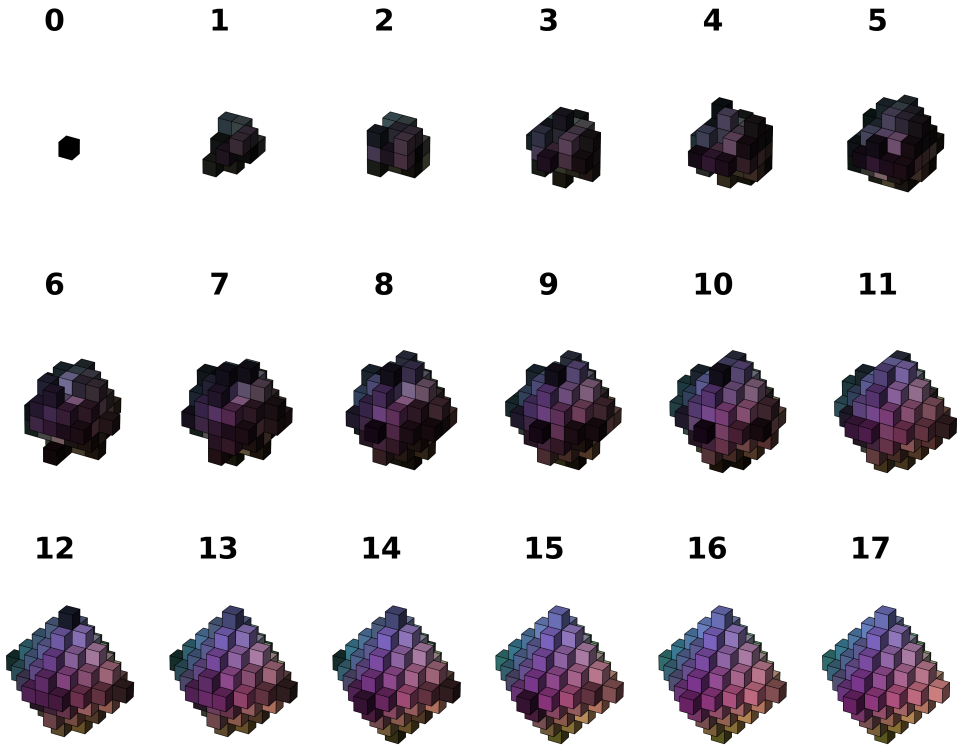
(b) Time step 18 to 24 is not displayed. The model stays stable and does not deviate from the target during these steps.



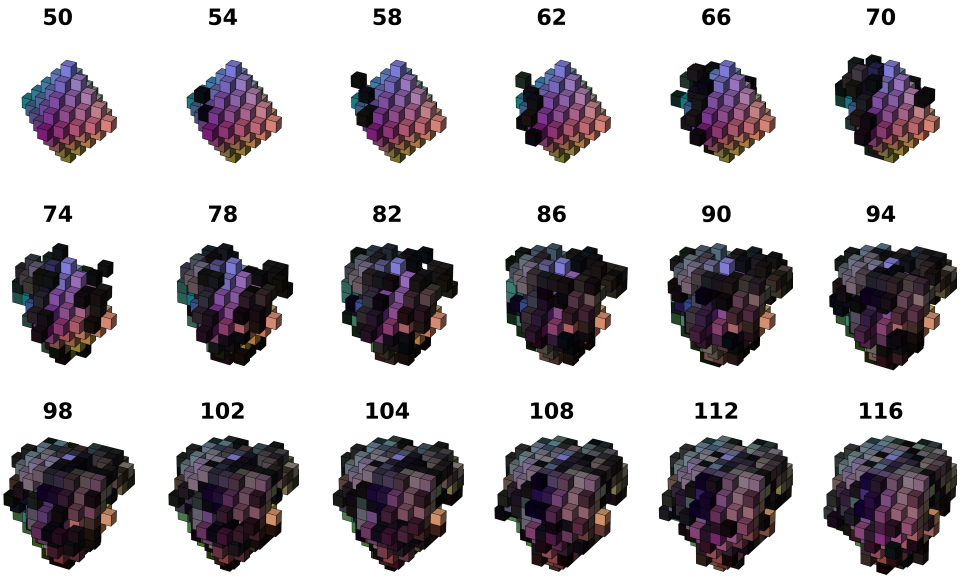
(c) As expected, the model becomes unstable. From time step 27 and out it grows uncontrollably throughout the remaining time steps

Figure 6.15: Experiment 4.1: Development per time step for a model simulated longer than trained.

(a) Time step 0 to 17: The model grows from a seed to the target shape.



(b) Time step 18 to 49 is not displayed. The model stays stable and does not deviate from the target during these steps.



(c) Time step 50 to 120: Every 4 time steps is displayed to reduce visual clutter. In step 54 the model start to grow cell's deviation from the target shape, and in the following steps the model grows exponentially into chaos.

Figure 6.16: Experiment 4.2: Development per time step for a model simulated longer than trained.

6.5 Experiment 5: Asymmetry

Experiment 5 aims to verify the NCA's ability to grow asymmetric shapes and colors

The target shapes have so far mostly been symmetrical shapes. This symmetry could potentially be something the NCA could detect when iterating through the training steps, and as a result generate a rule table that is based on equal growth in all directions from the starting seed. We want to examine this by running two simple tests, the first one being growing an asymmetrical shape. The second test is of colors. So far the colors used has been a spectrum of RGB values distributed evenly across the canvas, which also follows a pattern the NCA could exploit. To examine this, we run a simple test of a model containing cells with colors breaking this pattern.

6.5.1 Asymmetric shape

Like in the previous experiments, we also add some complexity to the target model. We increase the dimensions of the canvas to 20x20x20, and chose a target model consisting of 968 cells. The model (Figure 6.17) is an up-scaled version of the one used in the previous section, with two extra sections added. The bottom section is only connected to the rest of the model with a single cell to create a critical point, since if the connecting cell does not correctly grow, the NCA might never be able to grow the bottom section. Additionally a random section is added at the top. Figure 6.19 displays a simualtion of the model after training. The model reaches the target shape around time step 50, and is like the previous experiments, visually indistinguishable from the target. Note that the cell connecting the bottom part acts as a bottleneck in the way the morphogenesis of the shape evolves, which confirms the critical path definitions made in Section 6.1. Additionally, we observe that each time step provides a substantial growth of new cells, which continues until it reaches the target. This verifies Equation 6.2 of *Min Time Steps*. If the model had several time steps where it lingered without growing any cells, this would indicate that the model could be trained with a lower amount of time steps. See table 6.4 for training results. The doubled canvas size led to over 7 times as many cells in the target shape as in the previous experiment, see Table 6.2. This had a huge impact on the computational cost, supplemented by the higher time step range that comes with a larger model. The 7000 steps took 3 hours and 41 minutes, which comes down to just over 31

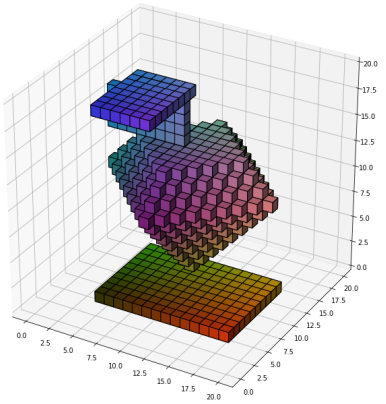


Figure 6.17: Target for Experiment 5.1

steps/minute. To provide some perspective for the loss history in Figure 6.18, the model reaches the target shape relatively accurately after just 2000 training steps, but with a few (dozens) of missing or misplaced cells, and the general distribution of colors correct, but not accurate to the exact RGB values.

Ex.#	Train Steps	Time step Range	Batch loss Final Train Step	Time	Steps to Target Shape
5.1	7000	[43,52]	-4.3	3h41min	50
4.2	12000	[25,30]	-5.2	34min	17

Table 6.4: Experiment 5: Training details

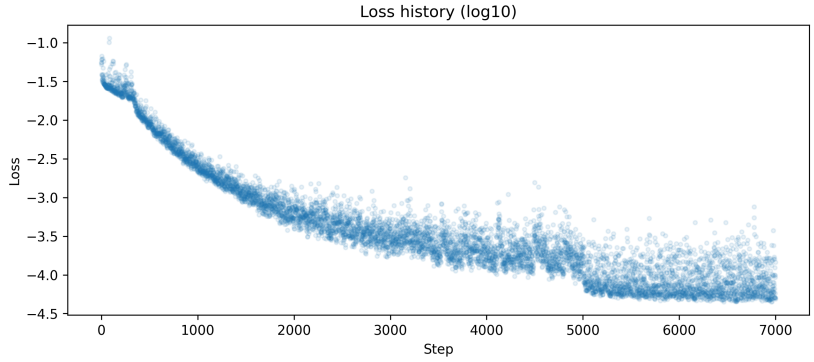


Figure 6.18: Loss, Experiment 5.1

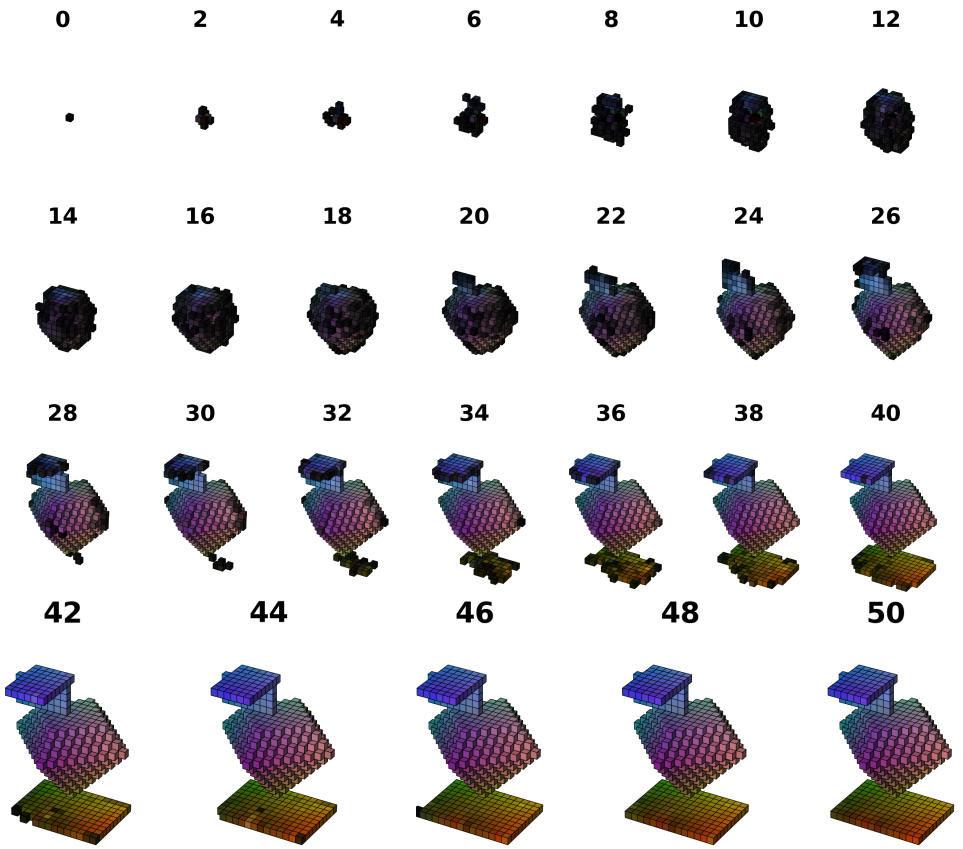
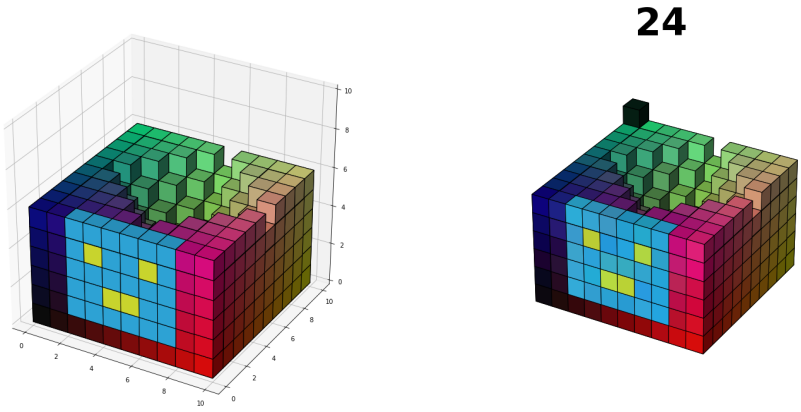


Figure 6.19: Experiment 5.1: Development per time step. The model reaches the target shape and color in frame 50. The bottom "platform" increases the minimum train steps as the model has to grow through a single cell acting as a bottleneck, before growing speeds up again as the platform increases in size.

6.5.2 Asymmetric colors

We run a simple test to confirm the NCA's ability to correctly grow cells with colors that does not follow any pattern. We create a recognisable smiley-face in yellow on the side of model 6.20a, which is reproduced in 6.20b after training the model. See Table 6.2 for parameters. The training was stopped training after 5000 steps (18min training time), as the research goal could be verified at this stage. Figure 6.20b confirms the ability to grow colors correctly.



(a) Experiment 5.2: Target. A recognisable smiley face is placed on the side of the model.

(b) Experiment 5.2: Grown model. The face has emerged and matches the target.

Figure 6.20: Experiment 5.2: Comparison of target and grown model.

6.6 Experiment 6: Complexity and size

Here we enable stability and regrowth measures, and compare performance on 3 models of different complexity.

In the previous experiments, the stability (pooling) and regrowth measures proposed by Mordvintsev et al. [17] has been disabled. We now enable these features during training of 3 targets of varying complexity, to evaluate how these measures perform on our 3D NCA, as well as measure of ability to simulate morphogenesis of these models. We chose three targets that each are challenging in a different way. See Table 6.2 for hyperparameters.

- Target 6.1: A model of a spider. The canvas is cropped to remove excess areas the model does not occupy. Normally this would be a 32x32x32 canvas, but is reduced down to 32x24x12 by removing excessive whitespace. This reduces the computational cost as smaller canvas means fewer cells operating, which leads to fewer perception vectors calculated and propagated through the update network. This model has one of the most challenging shapes however, as the 8 legs on the spider form a complicated geometry, and if poorly trained a CA will struggle to differentiate between the legs. The model totals 810 voxels.

- Target 6.2: A model of a tree. This target challenges the NCA with its high amount of voxels, and consist of 1151 voxels in total. The structure of the leaves on the model also provides a difficult challenge with its apparent randomness, something [27] encounters in their work. On the other side, this model has the lowest amount of voxel spaces in its canvas, totalling 4096. The seed is placed on the bottom

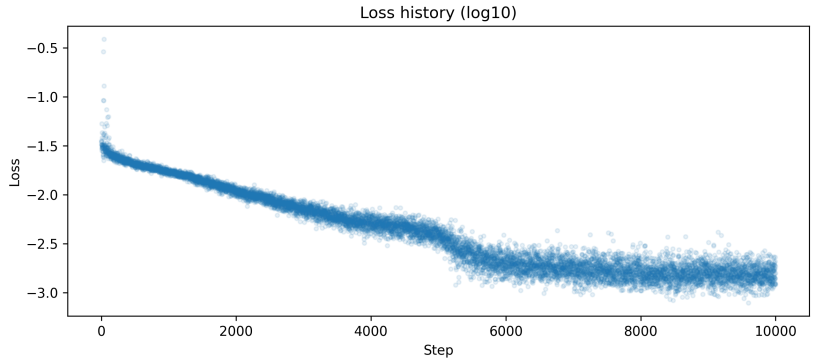
of the canvas to mimic how a tree grows from the ground and up. This effectively doubles the time step range, as the critical path now is from one end to the other, instead of having half the distance from the center of the model.

- Target 6.3: A model of a fish. Even though this model consists of the fewest amount of voxels, 620, it has the largest canvas, 32x32x32. This comes with a heavy computational cost, and is assumed to be the hardest model to train, despite the low amount of voxels, and relatively simple shape in terms of edges and randomness.

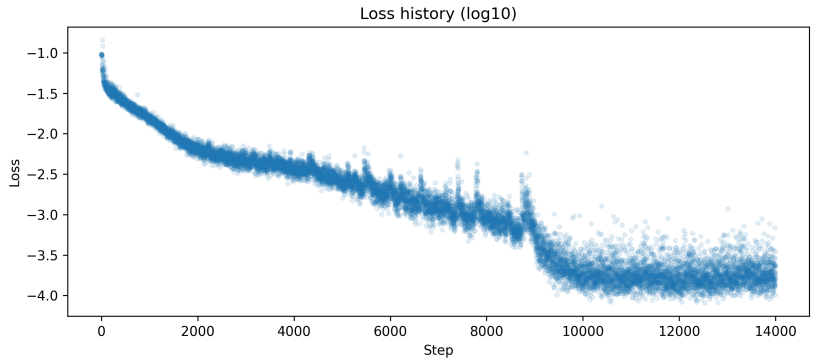
Following is an iteration of each model from seed to target shape, as well as a demonstration of their stability and regrowth capabilities.

Ex.#	Train Steps	Time step Range	Batch loss Final Train Step	Time	Steps to Target Shape
6.1	10000	[63,76]	-3.0	3h	66
6.2	14000	[45,54]	-3.8	3h 31min	53
6.3	7000	[45,54]	-3.5	11h 15min	60

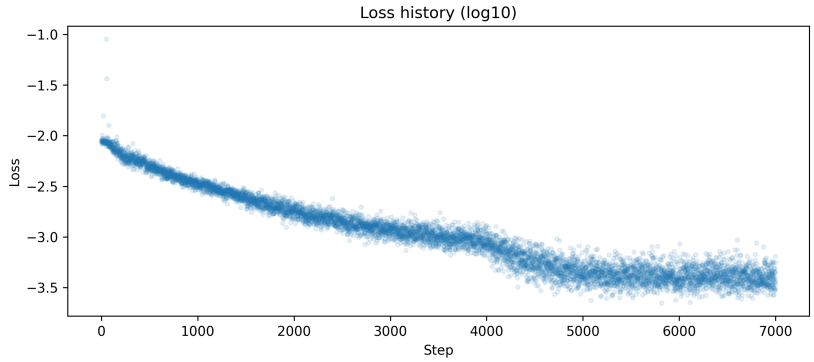
Table 6.5: Experiment 6: Training details. Loss is monitored during training. When the loss stops improving the training is manually stopped.



(a) Loss, Experiment 6.1

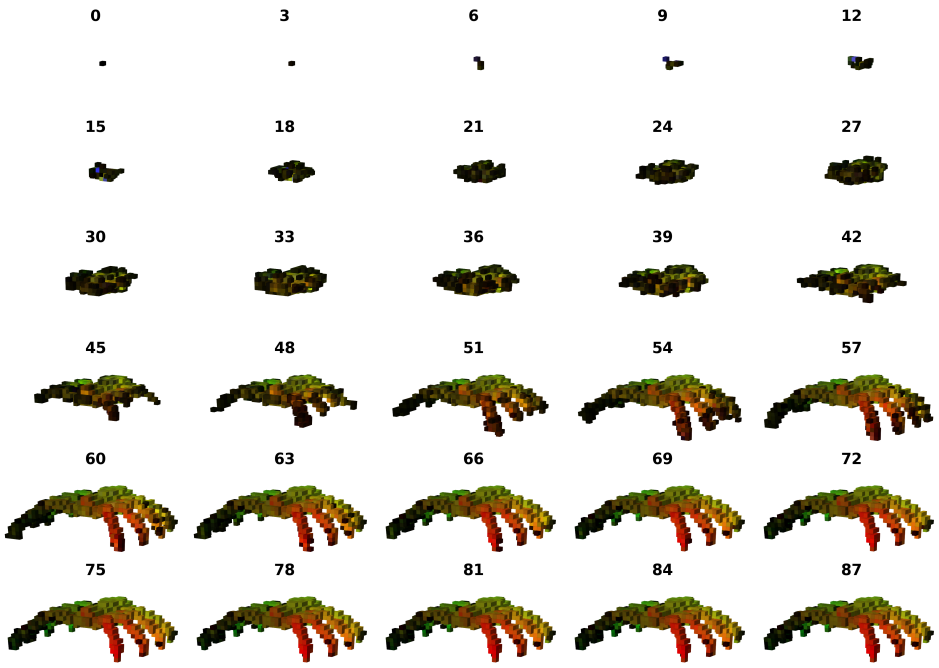


(b) Loss, Experiment 6.2



(c) Loss, Experiment 6.3

Figure 6.21: Loss for Experiment 6.1, 6.2 and 6.3. We observe similar development of loss. Experiment 6.3 had a significant increase in training time, and taking into account how the loss evolves over the final 2000 training steps, the training could haven been ended at 5000 steps, shortening the time significantly. However, the two other experiments also follow this pattern where they are trained roughly 30% longer than necessary, meaning their relative difference in training time persist.



(a) Experiment 6.1: Simulated morphogenesis of a spider. The trained NCA is updated for 90 time steps, starting with a single seed. The models reaches its target shaped at round time step 60, and stays stable from that point and out. Time step 90 is enlarged below for comparison to the target.

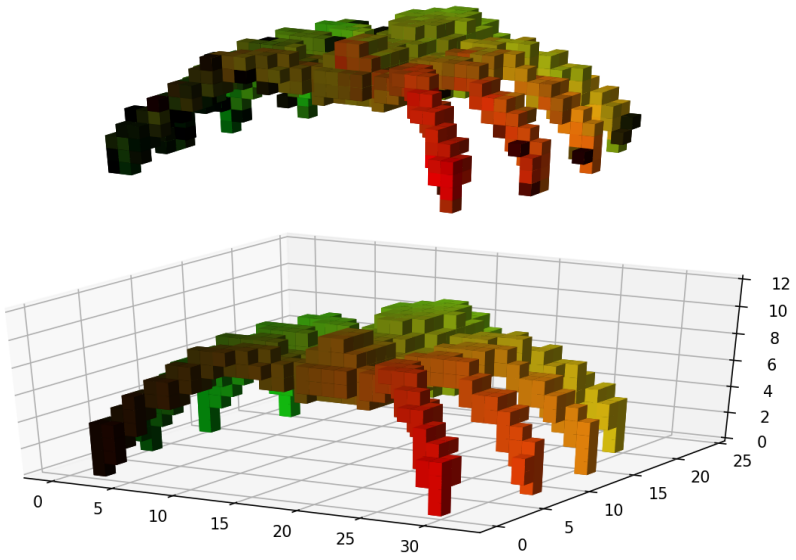


Figure 6.22: Experiment 6.1: Comparison of grown model(top, time step 90) and target model (bottom). A few voxels can been seen emerging on the legs of the grown model, which do not appear in the target shape. If looking closely at the face of the spider, a small difference in colors can be seen. Otherwise, the two shapes are almost identical. Impressively, the CA is able to grow 8 separate legs with their respective shade of color. We assume the minor deviance from the target shape would be omitted with fine-tuning of hyperparameters.

Figure 6.23: Experiment 6.1.1: Regrowth of damaged limbs. After the model has finished growing and stabilised, all 4 legs on the right hand side is removed in step 101 (first frame). Over the next 50 frames, the legs grow out. The new limbs do not match the exact shape they had earlier, but have a clear resemblance. The larger the removed section, the more the model struggles to regrow. Small sections get replaced with a high accuracy. A test (not visualised) removing just one leg, resulted in the leg being grown back to its original shape without any noticeable difference.

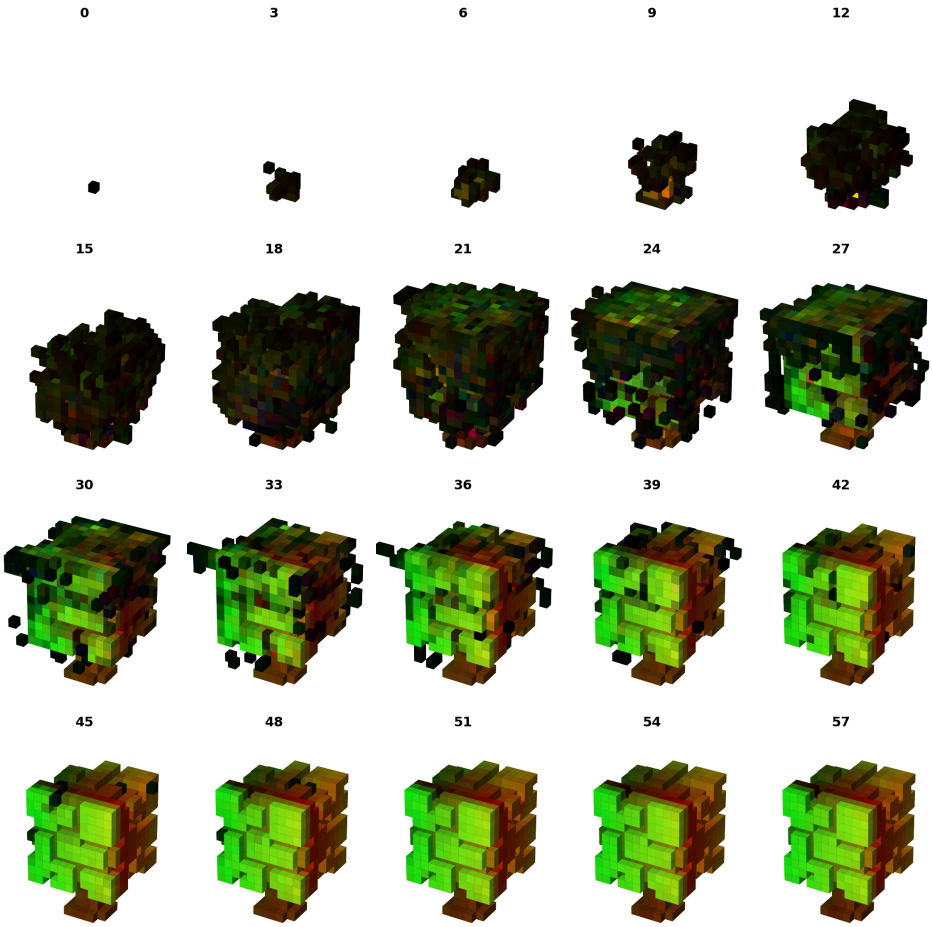


Figure 6.24: Experiment 6.2: Simulated morphogenesis of a tree. A tree is grown from a single seed to its complete shape in 60 time steps. We observe a rapid growth in the mid-early stages, where the model actually grows past the target shape in terms of size. In the following steps the model shrinks again the correct shape emerges. The apparent random way the leaves are formed has proven to be a challenging aspect for NCA's but the model impressively reaches a stable state where it is indistinguishable from the target, see Figure 6.25 for a side-by-side comparison.

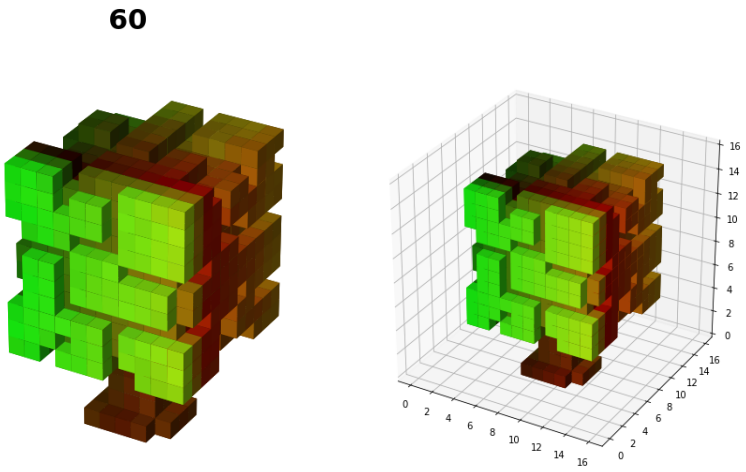


Figure 6.25: Experiment 6.2: Comparison of grown model (left side, after 60 updates of the CA), and target model on the right. Notice how the pattern of the leaves are correctly grown.

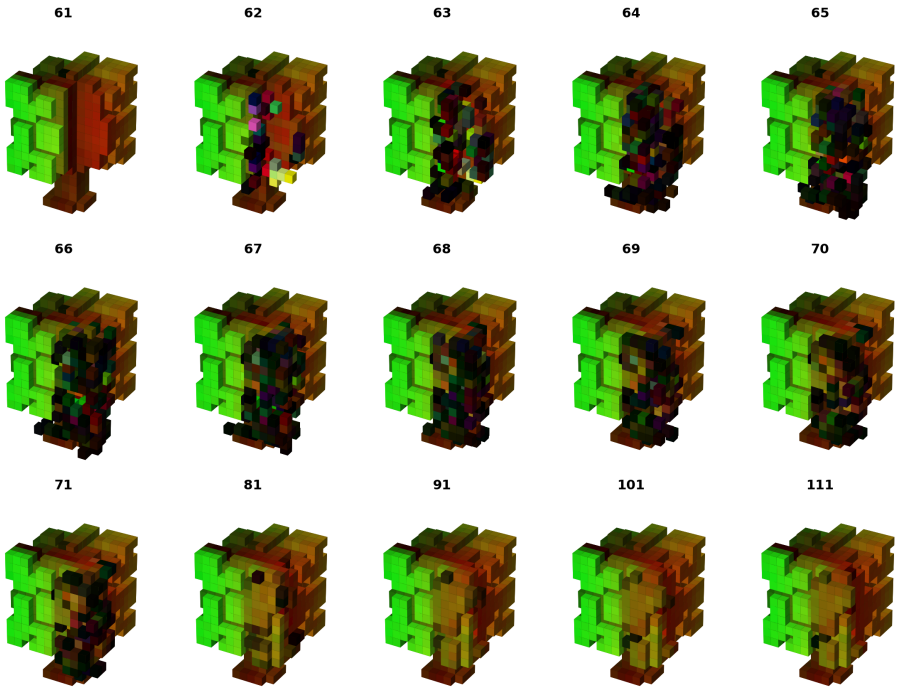


Figure 6.26: Experiment 6.2.2. After growing from a seed to the target shape in time step 60, a part of the tree is removed in the next time step. The figures displays how the part is grown back frame by frame. The last 4 subfigures each jumps 10 steps ahead, to show that some regeneration happens slowly even after a while. The overall right shape is grown, but with the detailing in the leaves and the coloring slightly off. Like in the previous experiment, the model's regrowth-capabilities are better if a smaller segment is removed. We believe that more extensive damage training would improve its regrowth further.

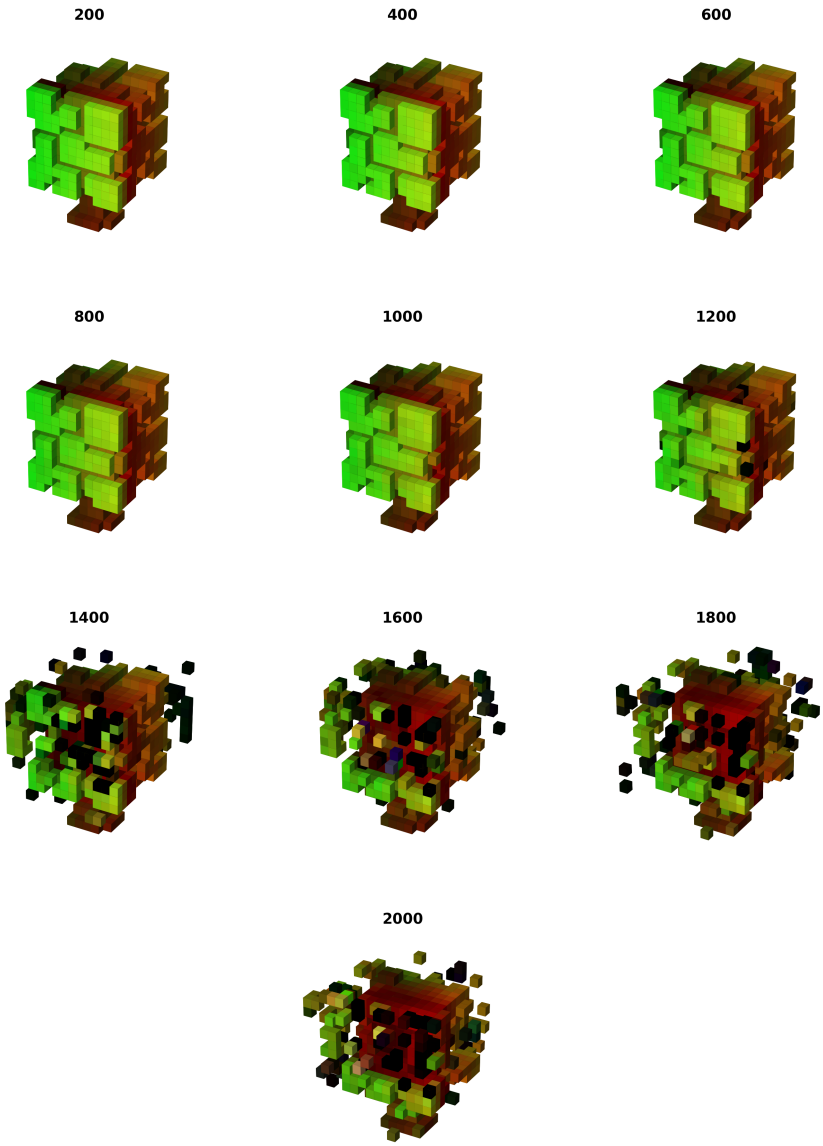


Figure 6.27: Experiment 6.2.3: Extensive stability test. The NCA has evolved 200 time steps in each figure. We observe that the model stays completely stable for over 1000 time steps. At step 1200 the first signs of instability emerge as 2 voxels appears. Between this and step 1400, the model explodes into instability, although interestingly the core of the model seems to persist while the leaves have dissolved. The model does not show complete time-invariance, but is stable long enough to indicate that this comes down to fine tuning the weights of the rule generating network. We suggest increasing the pool size, and increasing the amount of damage that is applied to the model during training to battle this, paired with increased training time.

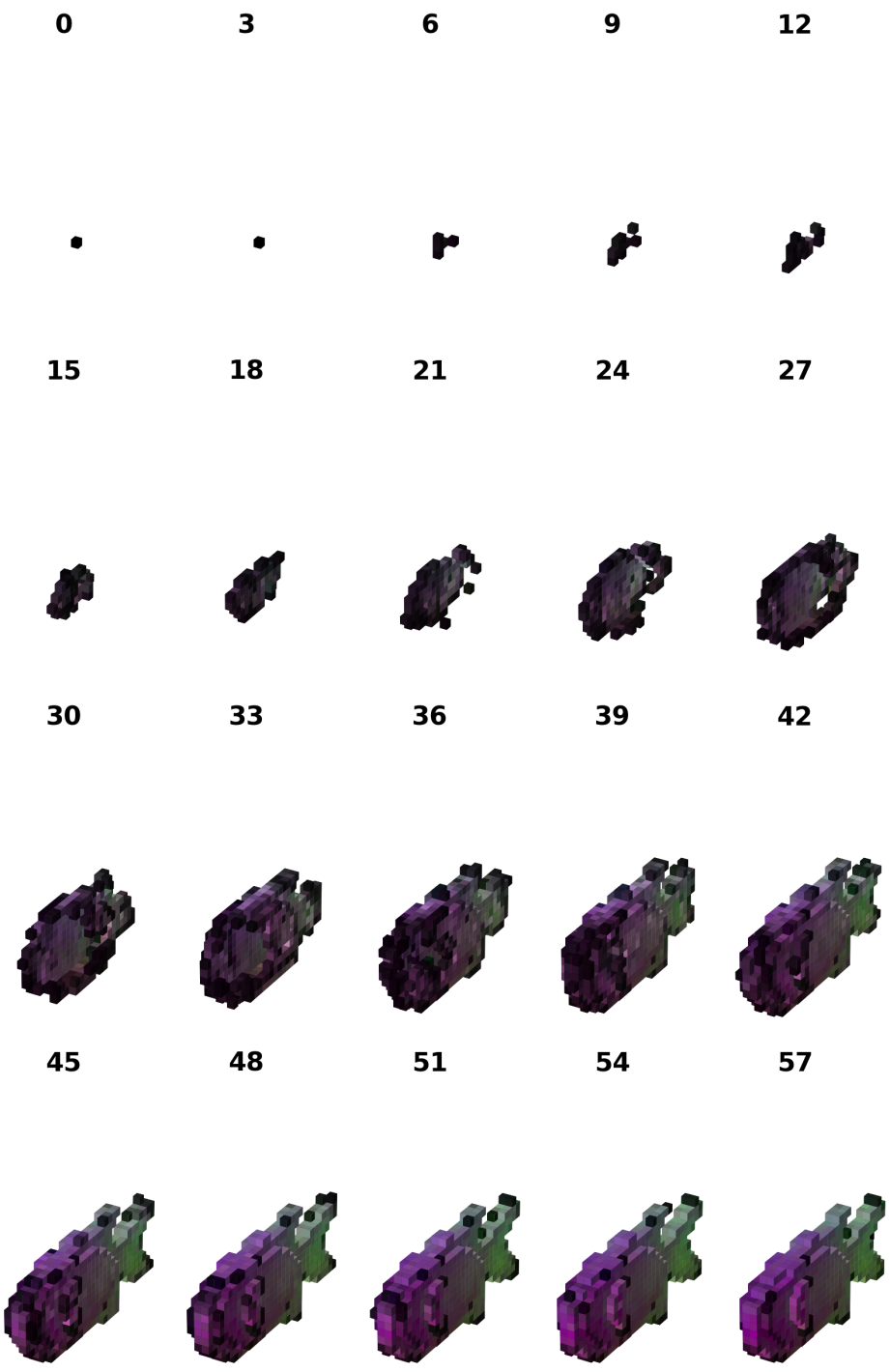


Figure 6.28: Experiment 6.3.1: Simulated morphogenesis. Over 60 time steps the model grows from a single seed to its complete shape. A few black voxels can be seen around the model, which also stays there if grown further. Like mentioned in the two previous experiments, this is a matter of training parameters. The training time was almost tripled for this model compared to the two previous experiments (11h), ref. Table 6.5. See below for comparison to targe shape.

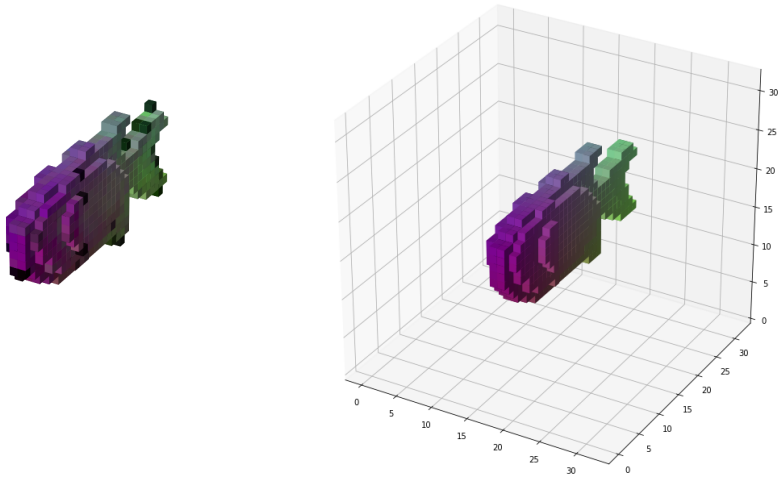


Figure 6.29: Experiment 6.3.1: Comparison of grown model(left, step 100) and target model (right). Despide the canvas size of 32x32x32, the model has stabilised at the target shape, with a handful extra voxels scattered around.

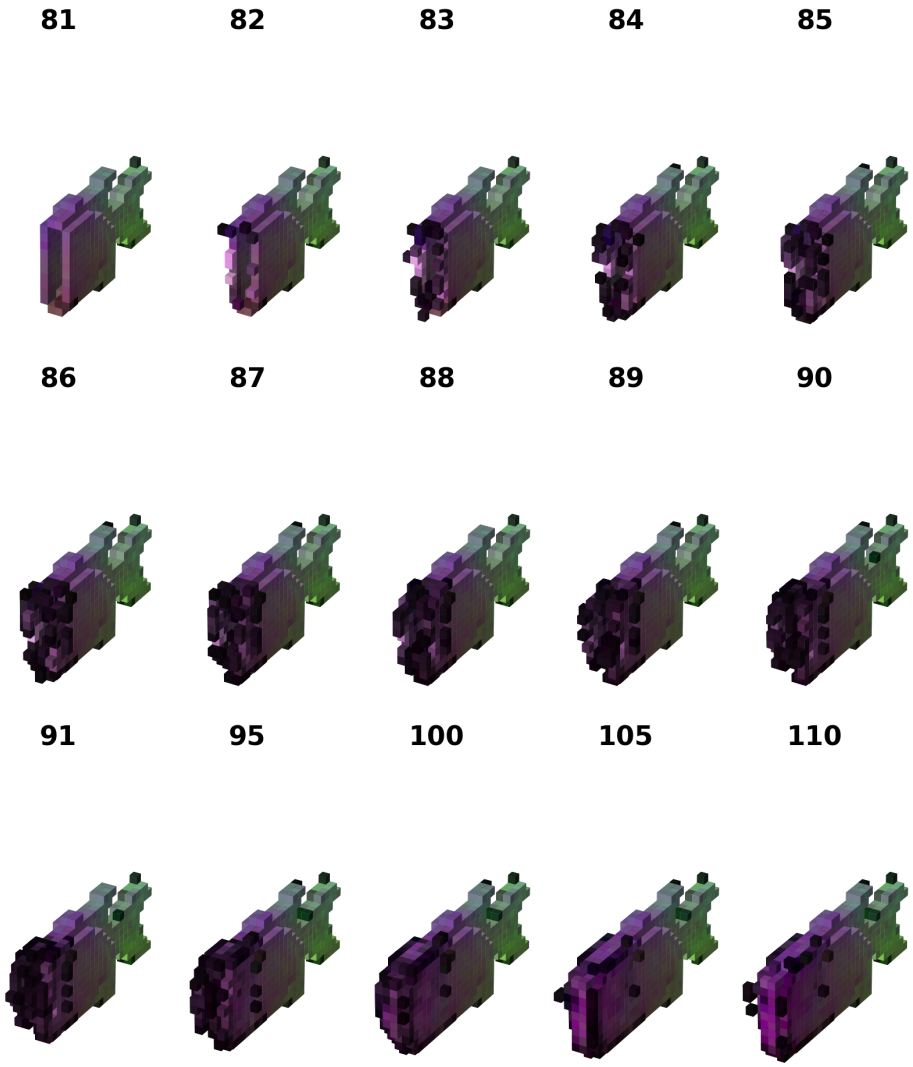


Figure 6.30: Experiment 6.3.2: Regrowth of damage. Interestingly, this experiment revealed a mistake made when creating this model's target. When removing half of it to demonstrate regrowth, the model revealed a hollow interior (Frame 81). The interior of models is usually filled like explained in Section 4.2.4, under bullet point 4. We keep the result as it adds an interesting attribute. The model is not able to regrow its rounded front after being damaged, and grows past it into a more square front. However the regrown shape still has a indisputable similarity to the original shape. Like the previous damage experiments, the displayed example shows a larger portion removed than what the model sees during training, and is more stable for smaller sections of damage.

6.7 Experiment 7: Guided morphogenesis and oscillating behavior

This experiment aims to verify the proposed novel methods for guided morphogenesis and oscillating behavior.

Two experiments are conducted for guided morphogenesis, and one for oscillation. The results are presented with their subsequent figures.

6.7.1 Guided morphogenesis

To introduce this experiment, we start off by repeating the gist of the proposed novel solution. We introduced the term guided morphogenesis to refer to the process of interfering with the growth between seed and target by introducing constraints. The constraints are added in form of checkpoints the model must reach before finally reaching its target shape. See section 4.8 for further details. We choose the previously seen target shape of a tree and remove its branches, leaving only the trunk. This becomes a sub-target (checkpoint) the model much reach before growing to its final target of the original complete tree, see Figure 6.31. The tree model is chosen for demonstrating the usefulness of guided morphogenesis with an analogy of the process of a tree regrowing its leaves. Additionally the model consists of the highest amount of voxels of all the models used in this thesis, meaning a successful demonstration would confirm a high level of performance. The result is presented below with figure 6.33.

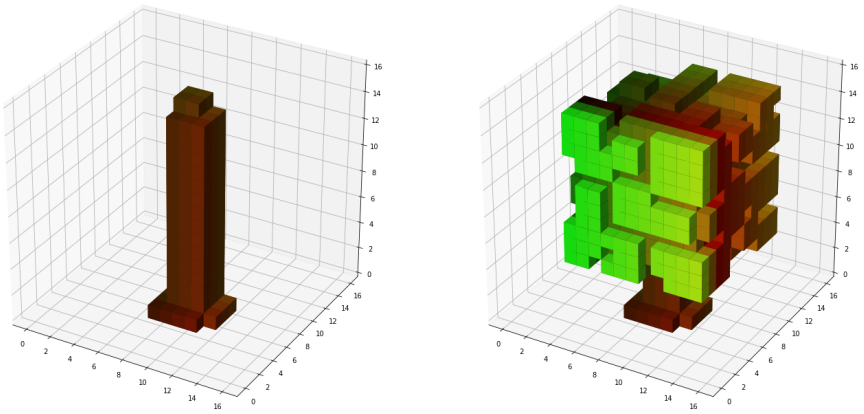


Figure 6.31: Guided morphogenesis: Sub-target(left) and target(right).

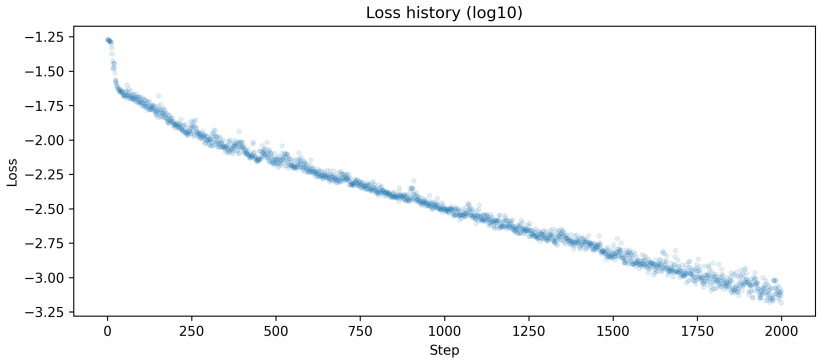


Figure 6.32: Loss, Experiment 7.1. This loss develops differently than the losses seen in the previous experiments, as training is cut off before the loss flattens out. This is a result of observing the model during training, and after 2000 training steps it had reached an accuracy we were happy with for demonstrating the functionality of this experiment. Normally the training would continue until the loss stops improving.

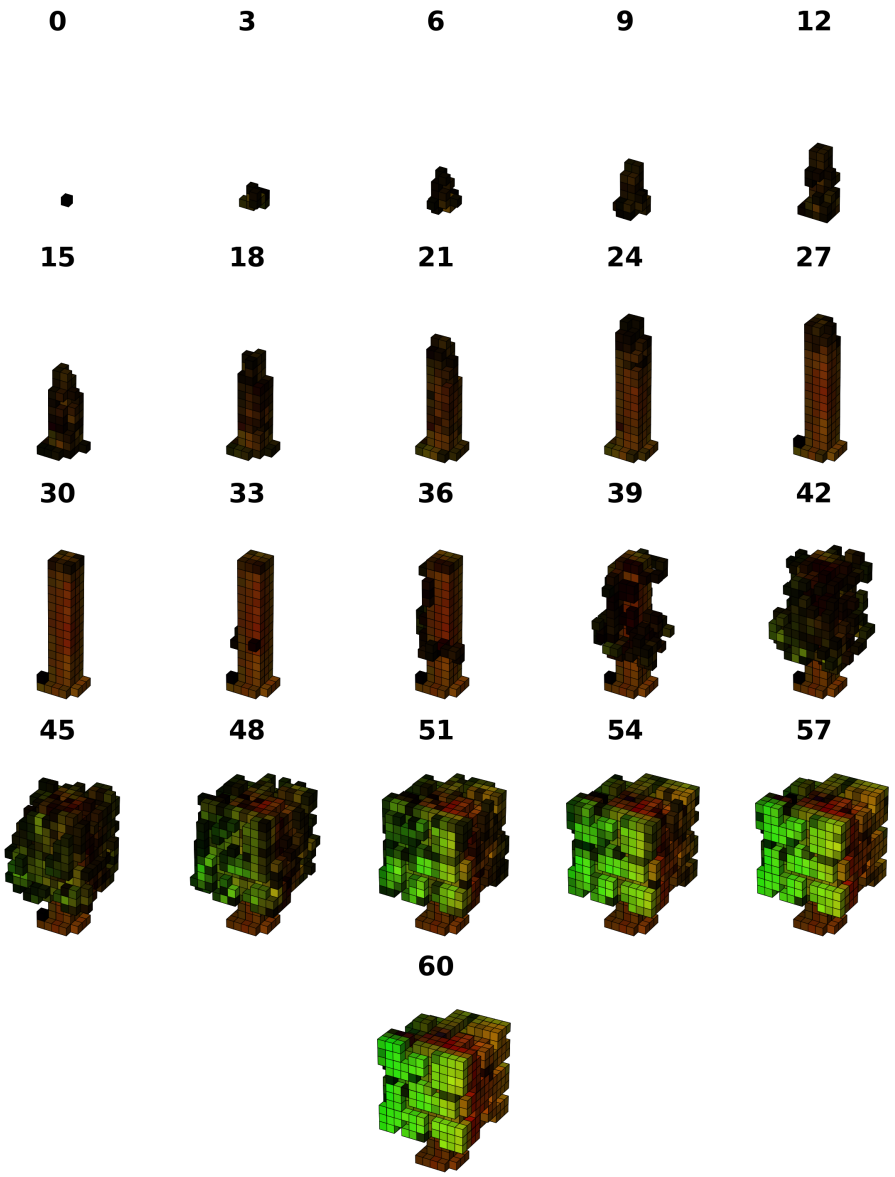


Figure 6.33: Experiment 7.1: Guided morphogenesis. The tree is forced to first grow the tree trunk before the leaves emerge. This allows for more advanced simulations by enabling the opportunity of incorporating external rules, restrictions or checkpoints to the morphogenesis. At time step 30, a target of just the trunk is added. This prevents any other voxels from emerging until that checkpoint has been passed, at which point the leaves of the tree starts growing. The model reaches the target shape with an equal accuracy as the tree model trained without guidance, and confirms the proposed method.

Next we demonstrate another property of the guided morphogenesis: Simulating a diminishing structure. As usual, we start with a single cell, which is grown into a target shape. However, we now want the model to dissolve, and simulate the development back to a single cell. The fully

grown shape becomes the sub-target, and a single cell becomes the target shape.

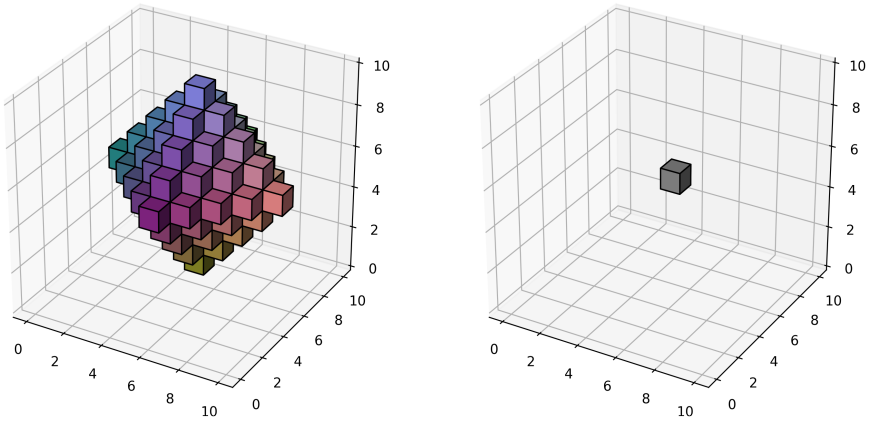


Figure 6.34: Guided morphogenesis: Sub-target (left) and target (right) for simulating a diminishing figure. Here we aim to grow from a single cell, into the sub-target, and then back to a single cell, simulating a diminishing structure.

Figure 6.35: Loss, Experiment 7.1.2. On the contrary to the previous experiment, this model is trained until the loss stops improving significantly.

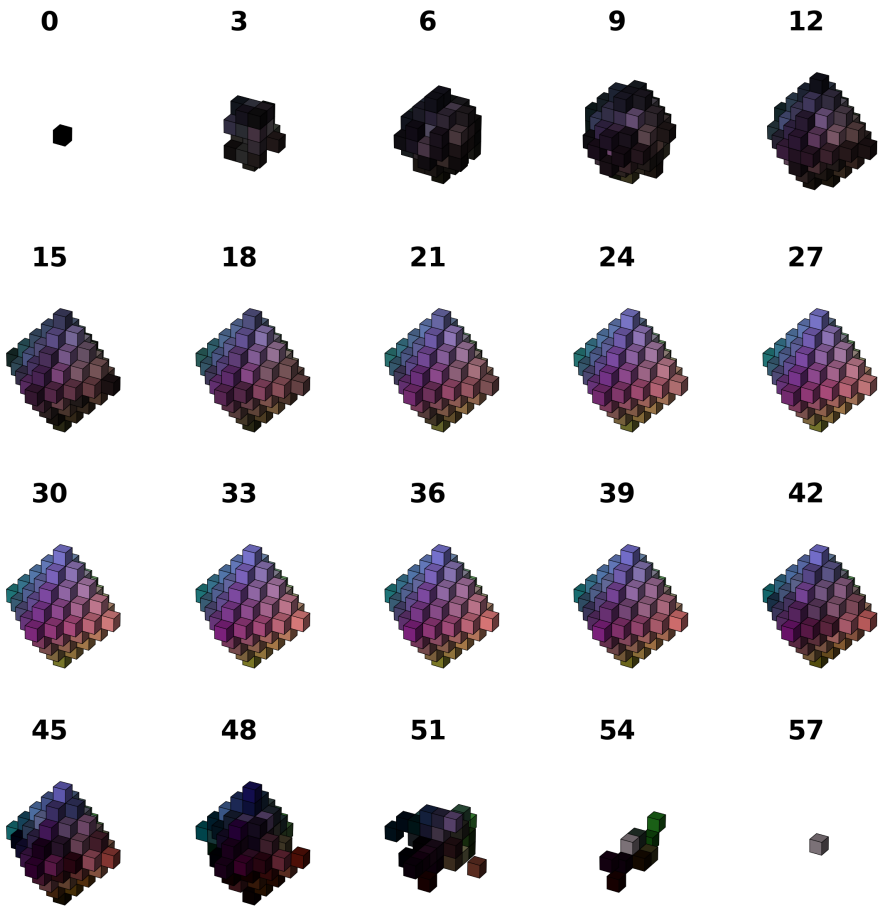
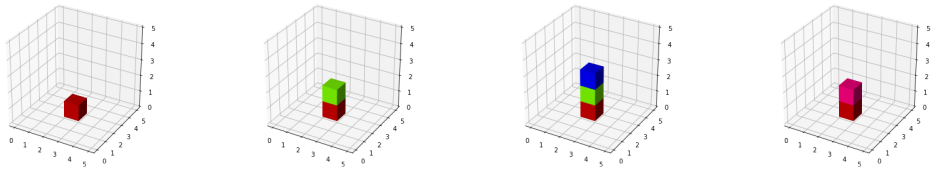


Figure 6.36: Experiment 7.1.2: Simulating a diminishing structure. A model is grown from a single seed into a sub-target (time steps 0 to 30). After this, the model is trained to return to a single seed. The cells die gradually, ending with just the center voxel alive. This proves another capability of the guided morphogenesis: simulating the "life-span" of a model, where it simulates a process involving both growth and diminishing voxels.

6.7.2 Oscillating motion

Again, we refer to section 4.8 for the theory behind the proposed method. This is an experiment taking place in the very early stages of defining the method. As a result this is conducted on the lowest possible level, to confirm its potential and provide insight for future development, much like Experiment 1 where only the most fundamental properties are examined. While theoretically any form or shape can be trained to exhibit an oscillating motion, we chose a simple pillar-like structure, see Figure 6.37. In contrary to the other experiments, this does not have a final target shape. The sub targets follow each other in a loop where target 4 is followed by target 1.



(a) Sub-target 1 (b) Sub-target 2 (c) Sub-target 3 (d) Sub-target 4

Figure 6.37: The 4 sub-targets of the oscillating motion. Sub-target 1 follows sub-target 4, creating a loop. 2 and 4 are differentiated by the colors in the top cell.

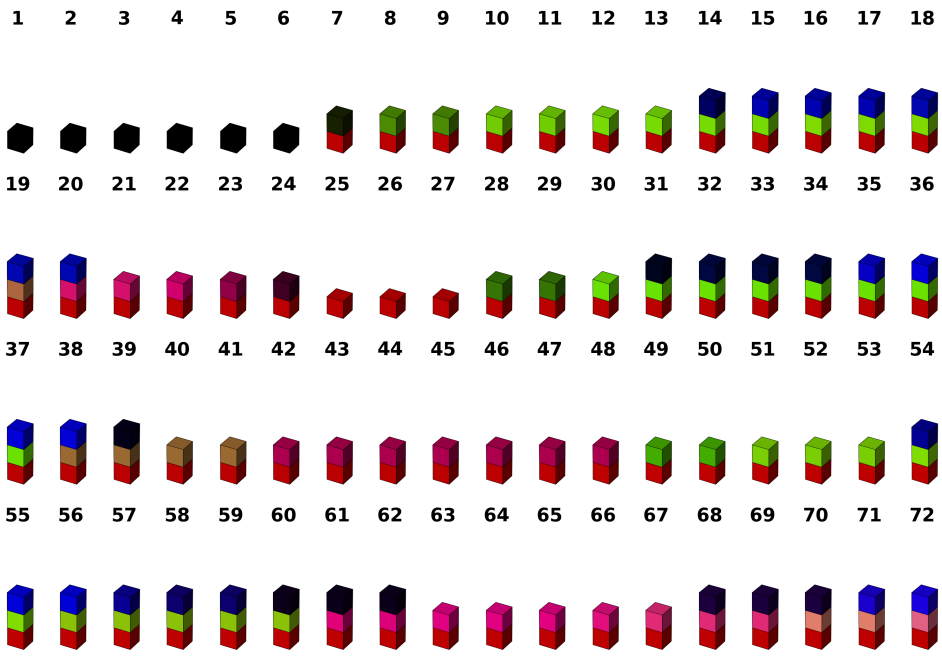


Figure 6.38: Experiment 7.2: Oscillating pillar. As an extension to the guided morphogenesis functionality, we propose a method for creating a CA able to exhibit an oscillating motion for any given time. The presented example does not yet exhibit an oscillating motion completely invariant to the time the simulation is ran for. Note that corresponding time steps are marked above each frame.

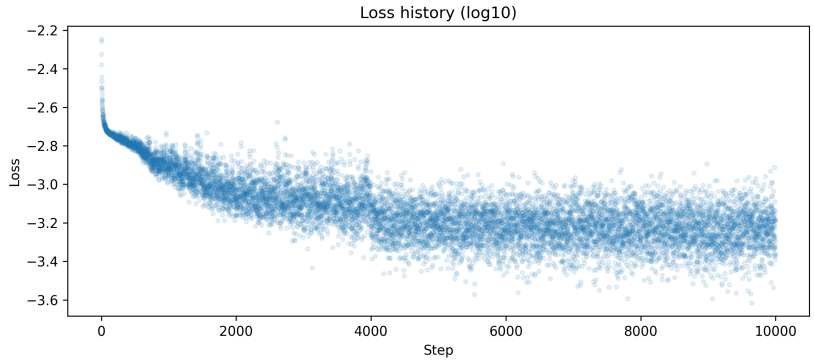


Figure 6.39: Loss, Experiment 7.2

During this experiment, the stochastic nature of the NCA is disabled, meaning no cells has their update randomly discarded. This is done to have a less complex environment which makes observing behavior on a per-cell level more intuitive. When studying only a small time range for a specific behavior, the random dropout of cell updates could cause ambiguous results. We are also moving away from the morphogenesis simulation, and moving more towards studying the possibilities of the simulation environment itself, which relaxes the requirement of desynchronised cell updates.

The model is trained for 10 000 training steps, see Figure 6.39 for loss history. We take note that the loss is spread out in a range between -3.0 to -3.5, and no improvement is seen over the last 6000 steps, the (visually observed) high variance in batch-wise the loss combined with the lack of improvement, indicates that the model struggles to learn. The results in Figure6.38 confirms this, as the desired oscillating behavior is not consistently exhibited.

The first 6 steps shows the seed before growth starts. In the next 7 steps, the model reaches sub-target 2. At this point, the first seed cell transforms into the correct color as well. In the following 7 steps the model reaches the next sub-target (3), with a red cell on the bottom, green cell in the middle, and a blue cell on the top. Over the next 4 steps the model reaches sub-target target 4, consisting of a red bottom cell and a pink middle cell. Following this is 3 steps where the model reaches sub-state 1, with only a red bottom cell. The CA has now reached time step 45. After this it once again reaches sub-targets 2 and 3. So far the model has exhibited the correct behavior, reaching all the sub states in a correct order of [seed, 2, 3, 4, 1, 2, 3, 4]. Ideally the model should have transformed from the seed state to sub-state 1 at the very beginning before reaching sub-state 2. At this point its unsure if this is a result of poorly training or a logical issue. In time step 49, the model fails as it break the oscillating behavior by going from sub-target 4 directly to sub-target 2 – skipping sub-target 1. After this it correctly passes through sub-targets 3 and 4, before it again breaks the oscillating behavior by going from sub-state 4 to 3 in time step 68.

A recap of the states the models went through is [2, 3, 4, 1, 2, 3, 4, **2**, 3, 4

,**3**]. The transitions underlined in bold are the ones breaking the oscillating pattern. In the next steps, the model drifts off and slowly becomes unstable. We consider this result as promising, because the behavior exhibited is very close to oscillating. This also makes sense in light of the development of loss during training.

Chapter 7

Discussion

The following sections discuss of the results in light of the research goals and comparison to related work.

7.1 Discussion

This section connects the results to the research goals and discusses the outcome.

7.1.1 3D NCA

We propose a framework, 3D NCA, capable of simulating morphogenesis of three-dimensional structures. The functionality and performance is evaluated through a set of experiments, each designed to confirm or refute capabilities of increasing magnitude, following the natural testing steps during development. Let us consider each experiment's results in chronological order.

Time steps

The first experiment corresponds to the 4th research goal of deriving a formula for determining a reasonable range of simulation time steps to use during training. A few research papers [22, 27, 36], using different variations of cellular automatas in the context of growing images and shapes was published during the work of this thesis, but none of them provide any insight into this process and some do not clarify how many updates the CA goes through. To help this field of study to keep its momentum, we consider it important to create a baseline for determining such hyperparameters. The proposed formula (Equation 6.2) is formulated as generally as possible to be applicable for different types of CAs aiming to simulate growth of a set target, and scales to any dimensionality, provided that the seed-to-edge distance is calculated (or estimated) corresponding to the number of dimensions and type of neighborhood, e.g. calculating the Euclidean distance versus metrics such as the taxicab metric. While the the definition of a critical path is derived, the final version of the

formula is adjusted to use an estimate rather than the true length of the critical path. We believe this is the best approach for two reasons. First off, the stochastic nature of our framework will always introduce randomness into the updates of a CA, meaning there will in our case never be one true lowest value. Secondly, the effort of precisely calculating the critical path length does in most cases, especially in 3D, not justify the benefits the ideal minimum time step yields. For 1D, 2D and 3D shapes, estimating the distance based on the proportions of the canvas can be done relatively accurately. The formula is used throughout the experiments for a consistent way of determining time steps, and does at no point appear to be under-performing. Experiment 4.2 reflects upon the effect of adjusting the time steps with the conclusion that the formula works, and need only be adjusted if the design goal of the simulation dictates it.

7.1.2 Verifying functionality of the proposed architecture

Experiment 2 to 6 follows the natural testing steps done during development, verifying low level functionality before more extensive testing is done. The level of success is evaluated through monitoring of L2 loss and visual inspection of the models. The visual inspection becomes the main factor for determining how well it performs, and is sufficient since the nature of the target shapes chosen allows for easy evaluation. Even more so, because the hyperparameters of the neural network is not fine-tuned to each model, we are not interested in the exact degree of precision our models achieve, but rather if they converge towards the target shape or not. A model that can be observed having just a fraction of misplaced cells, ref. Figure 6.22, would given the correct training parameters be able to reach the shape. The third factor is the stochastic nature of biology and morphogenesis relaxing the requirement of per-cell precision, shifting the focus to a more overall evaluation – in biology, a fish is still a fish even if two neighboring cells have swapped position. If another measurement metric had been deemed necessary, we would have implemented a check of true positive and negatives, and false positives and negatives, for each position in the canvas. True negatives would not provide much information as the models mostly consist of whitespace with large areas of dead cells. The false positives, false negatives and true positives would however be interesting, as long as scaled to the size of the models before used as grounds of comparison to other target shapes. A large canvas with 1 false positive is performing a lot better than a small canvas with 1 false positive, making this a delusive metric if comparing widely different sized models. Based on the attention to detail, the colors of each cell could be either excluded from this calculation, divided into ranges where a certain leeway is allowed, or be specific down to the exact RGB value. The latter would be useful if fine-tuning hyperparameters to optimize the model.

Throughout experiments 2 to 5, a cuboid of 4 voxels, a rectangular cuboid of 25 voxels, a diamond of 129 voxels and an unsymmetrical shape of 968 voxels is grown from a seed into the target shape, all reach the target shapes with 100% accuracy in terms of alive voxel placement. This

is thoroughly inspected visually. The colors are also indistinguishable for the human eye, which we consider to be over the threshold of justifying implementation of further metrics, again in reference to the biological analogy. The experiments verifies the framework's ability to grow and reproduce details on a per-cell level, both for shapes containing only a handful voxels in total, and for models consisting of up to a thousand voxels.

Experiment 6 trains more realistic models of a spider, tree and fish, each consisting of respectively 810, 1150 and 620 voxels. Like the previous experiments, these models also converge towards the target shape. However, they do have a varying degree of misplaced voxels scattered around their surface. As stated in the experiments, this comes down to fine-tuning hyperparameters during training, and we conclude that the framework is capable of reaching a per-cell level of precision even for these shapes.

We observe that the computational cost scales drastically with an increase in size of the canvas and model. First and foremost, the canvas size affects the cost by increasing the amount of total cells (both dead and alive), secondly, this cost is further increased by the amount of time steps the CA is updated. At every time step the CA updates, the amount of cells having their perception vector calculated is $height * width * depth$ of the canvas, see Table 3.1 for examples of how this scales based on the canvas size.

The current implementation is somewhat limited by its RAM usage. The number of voxels in a model does not affect this, as this is set by the size of the canvas and the batch size. If doubling the canvas size of our largest used canvas to 64^3 and keep the batch size of 8, the 16GB RAM available would not be sufficient, and run into an OOM error(out of memory) during allocation. The program would however run if we reduced the batch size to 2, at the cost of an slower convergence towards the target shape, which again leads to a longer total training time. A canvas of this size consists of a staggering 262 144 cells.

It is also worth noting that the framework is in these cases actually not just correctly reproducing the grown voxels, but also all the dead cells are part of the correctly grown shape. This is all the true negatives, which affects the computational cost in an equal manner, as all cells are evaluated during an update step. Performance should therefore also be seen in light of total voxels spaces in the canvas, not just the target shape. For a framework like the one we propose, the target shapes often occupy a minority of the canvas. An approach to reducing the cost in future work could be to only evaluate cells that are within a certain range of already alive cells, saving the cost of evaluating large sections containing only dead cells. This would not interfere with the analogy if simulating morphogenesis, as cells can only grow as a part of the already established structure.

7.1.3 Guided morphogenesis

We propose a novel solution allowing guidance of the morphogenesis through adding checkpoints the model must pass through. This broadens the scope of what an NCA can be used to simulate. Two use-cases are demonstrated, in the first one where we force a tree to grow its trunk before any leaves emerge. This demonstrates a use-case where we can implement constraints based on prior insight. If we already know certain stages an evolving multicellular organism passes through during growth, this proposal allows for incorporating these into the simulated morphogenesis, pushing the simulation closer to reality. The second use-case we demonstrate is to simulate a diminishing structure. This also broadens the scope of simulations the NCA can perform, and takes advantage of a CA's property as a system fit not only to simulate growth, but also how cells die. We have not been able to find any research of NCA's that evolves around this concept and therefore consider this to be one of the main contributions of this thesis.

We demonstrate a per-cell level of precision for guided morphogenesis, matching the precision of the normal morphogenesis of the previous section. An interesting observation is the development of the loss history of the tree models grown with and without guided morphogenesis (Figure 6.21b and 6.32). After 2000 training steps, the guided version had reached a (log10) loss of -3.2, whereas the normal version had a loss around -2.4. Keeping in mind that these losses are corresponding to different targets, the guided's target contains the normal one's plus an additional shape, and should inherently be more punishing in terms of loss values. However, the guided version gets a significantly better loss after an equal amount of training steps, leading us to believe that in this case the guided morphogenesis actually enhances the training. Without further investigation, we assume this is a result of the NCA benefiting from a narrower search space as the sub-target prevents the NCA from exploring widely incorrect shapes in the early stages.

A challenge with this method is the need of having detailed knowledge of these intermediate states, and having to model them, so they can be used during training of the model. Depending on the simulation, these states might be highly complex to model, or we may simply lack the prior knowledge of how the organism evolves.

7.1.4 Oscillating behavior

Additionally we propose a novel solution to simulating a shape exhibiting an oscillating behavior. Likewise to the previous section, no research has been found of this in the field of NCA's. Oscillation is however a central concept in traditional cellular automatas. The nature of their strict rule tables and discrete cell state representation allows for stability that is hard to reproduce with neural cellular automatas with a continuous cell state representation, as small inaccuracies can increase over time and can cause the model to drift off. In the previously mentioned Convway's Game of

Life [6], oscillators has been found in almost all periods. For instance, example **e** in Figure 2.3, labeled as a blinker, is an oscillator.

The proposed solution is not at a state where it is able to flawlessly exhibit an oscillating behavior, as it must be able to reproduce the pattern indefinitely, whereas at the current state, it deviates slightly from the pattern after one and a half iteration, which causes a ripple effect eventually rendering the model unstable. Whether this can be solved by simply tuning training parameters, or if a change in logic is needed, is at this point not clear. Regardless, we find the result interesting and consider it a contribution to evolving NCAs.

7.2 Comparison to related work

The main inspiration of this thesis [17], presents their result in a somewhat unconventional way, leaving accurate comparisons of performance challenging. However, we can compare the level of overall ability so simulate morphogenesis. Our extension matches their degree of successfully and accurately growing a single cell into a target. Their solution does not focus on the scale they are able to do the simulation on, and the stick to the same canvas size for all experiments. Taking Table 3.1 into consideration, which shows the exponential scaling of cells operating in a 3D space compared to 2D, we are able to simulate morphogenesis operating with a far higher amount of cells. This is not necessarily directly more challenging in terms of logic, but it has a devastating effect on computational cost. As a result, we refrain form training our models extensively, and in a few cases end training before the grown shape is indistinguishable from the target shape, simply as a result of how time consuming the process would be.

As mentioned, Sudhakaran et al. [27] also proposed a 3D NCA while the work of this thesis was ongoing, with the same research goal of extending [17] to 3D, which is something we could not have anticipated. Their implementation is able to grow models consisting of over 3000 cells, which is almost three times the amount of cells found in the largest model tested in this thesis. In hindsight, testing an equally sized model would have provided a good basis of comparison. However, we have no reason to believe our framework should not be able to grow models of such size, especially considering how our computational cost primarily is affected by the size of the canvas. If we compare canvas sizes, their larges canvas is of size 33x27x31, which totals 27 621 total cells. The largest canvas tested in this thesis is of size 32x32x32, which totals 32 768 cells operating. The effect of this was previously elaborated in Section 7.1.2, and leads us to believe the frameworks has at least the same magnitude of performance. Note again that the RAM usage discussed in Section 7.1.2 would not be affected by a higher amount of cells in the target models, as the implementation would use the same amount even if all the 32 768 cells were a part of the target shape.

While they have documented growth of a model consisting of more voxels, they refer to the level of accuracy as almost identical to the targets,

whereas our experiments had multiple examples of models being grown to be totally identical to the target shape.

Both frameworks are based on the logic proposed by Mordvintsev et al, and both uses 3D convolutional layers in the rule-generating network. They use two configurable layers in their update-network, and adjust the size of these between the two configurations (32, 32) and (64, 64). As explained earlier, we keep the architecture of our update network static for all models. Similar approaches is taken to the perception step, where they use regular 3D convolution, and we implement depthwise 3D convolution inspired by MobileNets. Their frameworks takes a different approach as they predict a discrete amount of cell types, where each cell type corresponds to a building block in their simulation environment, and is represented by a one-hot vector in the cell state. Loss is calculated differently, as they treat it as a multi-class classification problem using cross entropy with an Intersect Over Union (IOU) cost. Common for both frameworks is the adaption of the stability and regrowth measures proposed by [17], which both successfully incorporates. Another common ground is the ability to grow the internal structure of a 3D shape, even though in two different ways, as their framework grow a discrete cell type, whereas our framework grow a cell described by an RGB value. This is something [36] does not provide with their probabilistic generative CA, which instead exceeds at high-resolution shapes.

Compared to related work, the core functionalities of the NCA is analysed and documented more thoroughly by verifying underlying low level functionality in small, isolated experiments. This approach helps demystifying complex simulations.

Chapter 8

Conclusion

8.1 Conclusion

A neural cellular automata operating in three dimensions, able to grow from a single cell into a given shape, size, and color is proposed. Its functionality is documented through a range of experiments, in sum verifying high levels of functionality. With this, 3D morphogenesis is simulated on a per cell-level precision, with some models rendering indistinguishable from their targets.

A novel solution named guided morphogenesis is proposed. It allows interfering with the morphogenesis by adding checkpoint states the model must grow through before reaching its final shape. This broadens the scope of simulations NCAs can be used for, and demonstrates the ability to both grow and diminish a structure. Prior knowledge can be injected into the simulated morphogenesis, closing the reality gap further.

Stepping away from morphogenesis, we propose a novel solution of training a 3D NCA to exhibit an oscillating motion. The proposed method shows potential, but does not reach a state where a pattern repeats itself indefinitely. However, we consider this a contribution building a basis for future work of incorporating core functionality of traditional CAs.

A formula is derived for consistently determining a reasonable range of simulation time steps to use during training of systems of similar architecture. The formula is verified throughout the experiments. This contributes to the research field of NCAs by establishing a generalised guideline for setting a hyperparameter.

The proposed framework is evaluated in light of related work, and we conclude with an overall matching performance of growth, stability and regenerative capabilities, but with a contribution of novel functionality not seen elsewhere.

8.2 Future work

A logical step for future work would be to systematically examine how hyperparameters can be tuned to achieve more precise, larger and more stable models. In this thesis we leave these parameters outside the scope

of examination, which affects the performance. Specifically the correlation between canvas/model sizes and optimal number of hidden layers in the rule-generating network, and the corresponding size for each layer, would provide useful insight, alongside a mapping of the effects of scaling pool and batch sizes.

Performance-wise, we mention that the computational cost could be greatly reduced by only considering the cells that are part of a neighborhood of already alive cells. We imagine this would be a critical point for large scale simulations.

The guided morphogenesis functionality has only had its very basic functionality explored, and we imagine this is a tool which if developed further has potential to make an impact on how computer science can be used to simulate different aspects of morphogenesis. The approach would benefit from a more extensive mapping of its capabilities and limitations. An interesting approach would be to introduce checkpoints measuring less specific properties than a shape, such as total alive cells as a certain time step, or a given distribution of colors regardless of their placement. The proposed solution for generating an oscillating behavior with an NCA did not reach a level where it was successful, and although promising, future work should reevaluate the logic before taking the approach any further.

For creative futuristic ideas, we play with the biological analogy of a trained 3D NCA as a zygote. We let the weights of the network represent the genome, which holds all the information of the organism. Since a zygote is a combination of two parents' genomes, could we, following this logic, create new diverse pre-trained NCAs by combining the weights of two parent NCAs? Furthermore, if this is done to a population of NCA's, could we then create generations of NCAs, capable of simulating a diversity of morphogenesis and other phenomena?

Bibliography

[1] Embryo Project Encyclopedia (2010-06-14). *"John von Neumann's Cellular Automata"*. http://embryo.asu.edu/handle/10776/2009. Accessed: 2022-01-20.

[2] Plamen P. Angelov and Xiaowei Gu. 'Deep rule-based classifier with human-level performance and characteristics'. In: *Information Sciences* 463-464 (2018), pp. 196–213. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2018.06.048. URL: https://www.sciencedirect.com/science/article/pii/S0020025518304894.

[3] H. Arata et al. 'Free-form shape modeling by 3D cellular automata'. In: *Proceedings Shape Modeling International '99. International Conference on Shape Modeling and Applications*. 1999, pp. 242–247. DOI: 10.1109/SMA.1999.749346.

[4] Mario Botsch et al. *Polygon mesh processing*. CRC press, 2010.

[5] Wilfried Elmenreich and István Fehérvári. 'Evolving Self-organizing Cellular Automata Based on Neural Network Genotypes'. In: *Self-Organizing Systems*. Ed. by Christian Bettstetter and Carlos Gershenson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 16–25. ISBN: 978-3-642-19167-1.

[6] Mathematical Games. 'The fantastic combinations of John Conway's new solitaire game "life" by Martin Gardner'. In: *Scientific American* 223 (1970), pp. 120–123.

[7] Xavier Glorot, Antoine Bordes and Y. Bengio. 'Deep Sparse Rectifier Neural Networks'. In: vol. 15. Jan. 2010.

[8] Andrew B. Goryachev and Moisés Mallo. 'Patterning and Morphogenesis From Cells to Organisms: Progress, Common Principles and New Challenges'. In: *Frontiers in Cell and Developmental Biology* 8 (2020). ISSN: 2296-634X. DOI: 10.3389/fcell.2020.602483. URL: https://www.frontiersin.org/article/10.3389/fcell.2020.602483.

[9] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[10] Kaiming He et al. 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification'. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[11] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704 . 04861 [cs.CV].

[12] Miller J.F. 'Evolving a Self-Repairing, Self-Regulating, French Flag Organism.' In: *Deb K. (eds) Genetic and Evolutionary Computation – GECCO 2004.* (2004). DOI: https://doi.org/10.1007/978-3-540-24854-5_12.

[13] M. I. Jordan and T. M. Mitchell. 'Machine learning: Trends, perspectives, and prospects'. In: *Science* 349.6245 (2015), pp. 255–260. DOI: 10.1126/science.aaa8415. eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8415. URL: https://www.science.org/doi/abs/10.1126/science.aaa8415.

[14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[15] Daniel Maturana and Sebastian Scherer. 'VoxNet: A 3D Convolutional Neural Network for real-time object recognition'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 922–928. DOI: 10.1109/IROS.2015.7353481.

[16] Melanie Mitchell. 'Ubiquity Symposium: Biological Computation'. In: *Ubiquity* 2011.February (Feb. 2011). DOI: 10.1145/1940721.1944826. URL: https://doi.org/10.1145/1940721.1944826.

[17] Alexander Mordvintsev et al. 'Growing Neural Cellular Automata'. In: *Distill* (2020). https://distill.pub/2020/growing-ca. DOI: 10.23915/distill.00023.

[18] Alexander Mordvintsev et al. 'Thread: Differentiable Self-organizing Systems'. In: *Distill* (2020). https://distill.pub/2020/selforg. DOI: 10.23915/distill.00027.

[19] J. von Neumann. *Theory of Self-Reproducing Automata*. Champain, IL: University of Illionois Press, 1966.

[20] Stefano Nichele et al. 'CA-NEAT: Evolved Compositional Pattern Producing Networks for Cellular Automata Morphogenesis and Replication'. In: *IEEE Transactions on Cognitive and Developmental Systems* 10.3 (2018), pp. 687–700. DOI: 10.1109/TCDS.2017.2737082.

[21] Ettore Randazzo et al. 'Adversarial Reprogramming of Neural Cellular Automata'. In: *Distill* (2021). https://distill.pub/selforg/2021/adversarial. DOI: 10.23915/distill.00027.004.

[22] Alejandro Hernandez Ruiz, Armand Vilalta and Francesc Moreno-Noguer. 'Neural Cellular Automata Manifold'. In: *CoRR* abs/2006.12155 (2020). arXiv: 2006.12155. URL: https://arxiv.org/abs/2006.12155.

[23] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409 . 1556 [cs.CV].

[24] Alvy Ray Smith. 'Real-time language recognition by one-dimensional cellular automata'. In: *Journal of Computer and System Sciences* 6.3 (1972), pp. 233–253. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(72)80004-7. URL: https://www.sciencedirect.com/science/article/pii/S0022000072800047.

[25] Irwin Sobel. 'An Isotropic 3x3 Image Gradient Operator'. In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[26] Shuran Song and Jianxiong Xiao. 'Sliding Shapes for 3D Object Detection in Depth Images'. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 634–651. ISBN: 978-3-319-10599-4.

[27] Shyam Sudhakaran et al. *Growing 3D Artefacts and Functional Machines with Neural Cellular Automata*. 2021. arXiv: 2103.08737 [cs.LG].

[28] Lyne P. Tchapmi et al. *SEGCloud: Semantic Segmentation of 3D Point Clouds*. 2017. arXiv: 1710.07563 [cs.CV].

[29] Pedro A. Tsividis et al. *Human-Level Reinforcement Learning through Theory-Based Modeling, Exploration, and Planning*. 2021. arXiv: 2107.12544 [cs.AI].

[30] Alexandre Variengien et al. 'Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent'. In: *CoRR* abs/2106.15240 (2021). arXiv: 2106.15240. URL: https://arxiv.org/abs/2106.15240.

[31] Eric W Weisstein. *"Moore Neighborhood." From MathWorld–A Wolfram Web Resource.* https://mathworld.wolfram.com/MooreNeighborhood.html. Accessed: 2022-01-20.

[32] Eric W Weisstein. *von Neumann Neighborhood, From MathWorld–A Wolfram Web Resource.* https://mathworld.wolfram.com/vonNeumannNeighborhood.html. Accessed: 2022-01-20.

[33] Stephen Wolfram. *A New Kind of Science*. English. Wolfram Media, 2002. ISBN: 1579550088. URL: https://www.wolframscience.com.

[34] N. Wulff and J A Hertz. 'Learning Cellular Automaton Dynamics with Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by S. Hanson, J. Cowan and C. Giles. Vol. 5. Morgan-Kaufmann, 1993. URL: https://proceedings.neurips.cc/paper/1992/file/d6c651ddcd97183b2e40bc464231c962-Paper.pdf.

[35] Rongtian Ye, Fangyu Liu and Liqiang Zhang. *3D Depthwise Convolution: Reducing Model Parameters in 3D Vision Tasks*. 2018. arXiv: 1808.01556 [cs.CV].

[36] Dongsu Zhang et al. 'Learning to Generate 3D Shapes with Generative Cellular Automata'. In: *CoRR* abs/2103.04130 (2021). arXiv: 2103.04130. URL: https://arxiv.org/abs/2103.04130.

[37] Xuan Zhang et al. 'AlignedReID: Surpassing Human-Level Performance in Person Re-Identification'. In: *CoRR* abs/1711.08184 (2017). arXiv: 1711.08184. URL: http://arxiv.org/abs/1711.08184.