university of groningen / faculty of science and engineering

Evolving Lindenmayer systems
in a competitive environment

Job J.V. Talle
Supervised by Jiří Kosinka, PhD & Prof Dr Bart Verheij

Abstract

Lindenmayer systems have been developed to model plant growth by repeatedly applying production rules to an initial axiom, and serve as a model for genetically driven growth processes found in nature. A simulation method is proposed to evolve their phenotypic representations in a competitive heterogeneous environment to further expand on this biological analogy.

1 Introduction

In addition to the established biological realism of Lindenmayer systems [1], a model is proposed to evolve systems as agents in an environment designed to maintain this analogy in several ways. Evolutionary algorithms have been deployed to optimize Lindenmayer systems before, in two-dimensional environments [2, 3] as well as in three-dimensional environments [4, 5]. Drawing on these past achievements, the focal points of the proposed model are

- simulating competition between realistically modelled agents by simulating sunlight occlusion and spatial scarcity,
- modelling an environment with unevenly distributed fertility to simulate natural boundaries and thereby facilitating divergent evolution, and
- allowing the temporary survival of sub-par agents to give them a chance to escape local optima.

When these mechanisms stay true to their biological counterparts, the simulated organisms should be able to achieve success through the same strategies that can be observed in real plants. The simulation is therefore designed to evoke realistic behaviour by simulating a realistic environment.

The Lindenmayer system syntax used in the experiments is described and demonstrated in Section 2, along with the algorithm translating the resulting sentences (genotypes) into three-dimensional shapes (phenotypes). Then, the method used to evolve agents using this syntax to create their plant-like representations is outlined in Section 3. Section 4 documents how the method is implemented for the performed experiments. Finally, the results of several experiments are presented in Section 5, and they are reflected upon in Sections 6 and 7.

2 Syntax

A large number of syntactic variations of Lindenmayer systems exists; a subset of existing rules was used in this method to allow for effective evolution, but the syntax is chosen to be versatile enough to produce a wide variety of phenotypes.

All production rules take the form $x \to y$, where both x and y are strings of symbols. While y can be empty, x can not. Rules are applied to axiom a, which is a nonempty string, to produce a'. When performing an iteration on these symbols, the algorithm iterates over a starting at the first symbol. At the iterator position, all matching production rules are gathered and a random one is chosen for application. The index of the iterator is then incremented by the number of symbols in x of the rule, and the symbols matched to x are replaced by y. When no production rules are applicable, the iterator increments by one without altering the symbol.

To illustrate this process, consider a very simple Lindenmayer system with axiom $a = A$ and a single production rule $r_1 = A \to ABA$. Iterating over the axiom twice to obtain a' and a'' gives

$$a = A,$$
$$a' = ABA,$$
$$a'' = ABABABA.$$

This system is deterministic, since at most one production rule is applicable at a time. It can be made stochastic by introducing the second production rule $r_2 = A \rightarrow AB$. In this case, multiple end results are possible. Figure 1 shows some of the possible outcomes of this system. For every A encountered, both r_1 and r_2 are applicable.

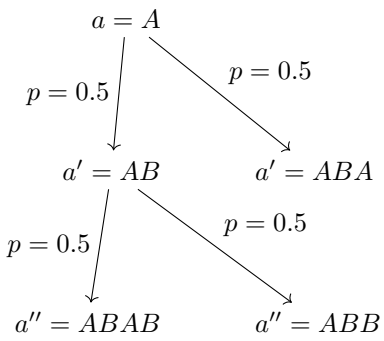

Figure 1: Two possible iterations of a stochastic Lindenmayer system. Iterating over a can result in either $a' = AB$ or $a' = ABA$. $a' = AB$ is expanded as well, leading to either $a'' = ABAB$ or $a'' = ABB$, and so on.

For complex stochastic systems, the number of possible outcomes rises exponentially. A single genotype may lead to a wide variety of phenotypes.

2.1 Phenotypic representation

A specific set of symbols is allowed to arise in the simulation, and different symbols (or combinations thereof) translate into different geometric representations. The string of symbols resulting from a number of iterations is rendered as a branching structure through so-called *turtle graphics*; the "turtle" moves according to the symbols encountered while iterating over all symbols sequentially. The turtle acts like a pen, drawing lines behind it as it moves through space. In our setting, the turtle draws branches, which are modelled as three-dimensional tubes instead of lines. The radius of the tube is proportional to the amount of structure that rests on it; plant branches get thicker near the roots.

Capital letters are interpreted as forward steps with a constant stride, lower case letters are ignored by the turtle. A seed symbol produces a seed at the turtle location. There are six rotation symbols, which rotate the turtle orientation along each axis in three dimensional space by a fixed amount, either in the positive or negative direction.

Branching brackets start or end a branch. Opening a branch does not affect the turtle, but when a closing bracket is encountered, the turtle jumps back to the position it was at when the corresponding opening bracket was encountered. Figure 2 shows the result of this modelling process.

Figure 2: A wireframe model of a three-dimensional branching structure modelled by a turtle on the left, with its shaded counterpart on the right. In this instance, the turtle initially points upwards.

A special opening bracket is the *leaf bracket*, which designates that the entire branching structure inside it will be interpreted as a *leaf*. A leaf is modelled by creating leaf material between the branches inside its structure and their child branches recursively. Leaves are not allowed to contain other leaves. By using this method, a leaf can contain more than one piece of leaf surface material; if many branches exist in the leaf structure, many different leaf shapes can be modelled.

Figure 3 contains a diagram of a possible leaf shape. The branches $a = \{a_0, a_1, a_2, a_3\}$, $b = \{b_0, b_1, b_2, b_3\}$ and $c = \{c_0, c_1, c_2\}$ exist, where a is the root branch starting at a_0. Branch b is a sub-branch of a, and leaf surface L_{ab} is modelled according to the aforementioned algorithm. Correspondingly, leaf surface L_{bc} is modelled from the point where c branches off b until both branches' end nodes. The result is a leaf structure containing three branches and two surfaces.

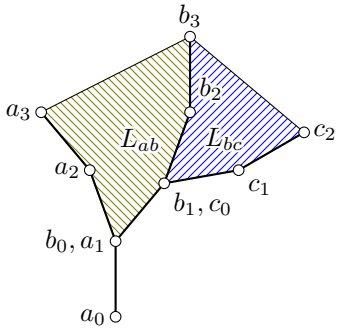

Figure 3: Leaf shape modelling.

The shape in Figure 3 can be constructed from a string of symbols using the aforementioned algorithm. The sentence producing it is

```
<A[++B[++C-C]-B-B]-A-A]
```

In this string,

- `-` rotates the turtle orientation $\frac{\pi}{9}$ radians to the left,
- `+` rotates $\frac{\pi}{9}$ radians to the right,
- `[...]` denotes a branch, and
- `<...]` denotes a leaf.

When `A`, `B` or `C` is encountered, the turtle moves forwards for a fixed distance, and the initial direction points the turtle upwards. In this sentence, the entire shape is surrounded by a leaf branch, and it contains b as a sub-branch, which in turn contains c as a sub-branch.

2.2 Alphabet

The experiments performed according to the method in Section 3 take place in a three-dimensional environment, but the syntax is almost equal to the one used to model the shape in Figure 3; the only difference is the use of four more rotation symbols. Six rotation symbols are used in total, two for each axis.

Within the simulation, an alphabet of possible symbols was chosen to model agents in three dimensions. Systems and sentences discussed from now on use this alphabet. Table 1 shows all symbols that may arise in the simulation together with their effects on the turtle.

Symbol	Turtle action
`A...Z`	Move forward
`a...z`	Ignore
`*`	Place a seed
`[`	Open a branch
`<`	Open a leaf
`]`	Close leaf or branch
`+`	Increment pitch
`-`	Decrement pitch
`/`	Increment roll
`\`	Decrement roll
`^`	Increment yaw
`_`	Decrement yaw

Table 1: The alphabet of possible symbols and their corresponding turtle actions.

This alphabet is mostly a three-dimensional extension of the syntax used in Figure 3, with the exception of the seed symbol. The use of seeds is explained in Section 3.3. In this Cartesian world coordinate system, *pitch* rotates around the Z-axis, *roll* rotates around the Y-axis and *yaw* rotates around the X-axis. Whenever one of the rotation symbols is encountered, the turtle rotates into the given direction with a preset angle; this angle is constant within a simulation.

3 Method

The simulation must be set up before experiments are performed; the parameters that are chosen in this phase strongly influence the course of agent development. The prerequisites of a simulation are categorized in three distinct modules, namely

- an environment, which contains the initial agents,
- a utility function used to rate agents' phenotypes, and
- a mutator containing the mutation function and its parameters.

3.1 Initializing the simulation environment

The simulation takes place on a plane where fertility varies locally. Fertility constrains the allowed growth for agents' systems. Higher fertility allows for more iterations of growth, and the number of symbols each iteration may add to a system correlates positively with fertility as well. In effect, agent size and complexity correlates positively with fertility.

An environment is defined as a rectangular terrain which may be sampled for fertility at any point on its surface. A terrain $t(x, y)$ yields a fertility factor in the range $[0, 1]$, where a higher value of t denotes higher fertility.

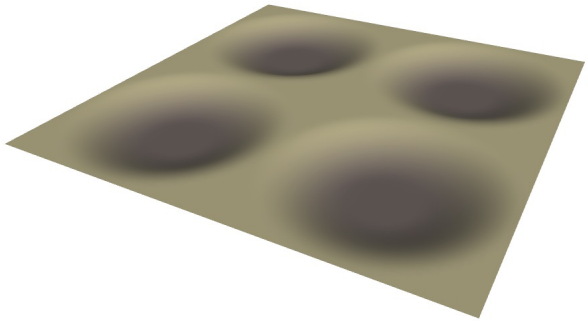

Figure 4: An environment with four fertile valleys.

Terrain functions that define infertile barriers separating lush valleys aim to promote divergent evolution of agents; simpler and smaller ancestors of agents may have crossed the low-fertility barriers, but the valleys may give rise to more complex agents that require a higher fertility that cannot migrate through scarce regions anymore and thereby speciate. Figure 4 shows a possible environment, where four darker highly fertile valleys are surrounded by an infertile plateau.

The formula $n = s + m \cdot i^2$ is used to calculate the number of symbols a system is allowed to have at a certain iteration while being generated; s is the initial number of allowed symbols a system may have (for any system, the axiom must not be larger than s). The parameter i is the iteration for which the maximum number of symbols needs to be sampled, and m is a positive number that may be increased to allow more growth. The parameters s and m are defined such that they correlate positively and linearly with t; if these values are high, agents are able to develop larger structures. The number of iterations a system will be applied for to generate the phenotype also scales linearly with terrain fertility; this is a property of the environment as well.

The environment should be *seeded* with an initial set of agents before the first generation is evaluated. To obtain the results presented in Section 5, simulations were at all times seeded with agents containing a Lindenmayer system without any rules and an axiom containing a single seed symbol (Section 3.3 explains how seeds are interpreted to produce the next generation of agents). This is indeed the minimal progenitive system that could possibly be used for seeding, since it does not contain any additional information or bias besides the ability to reproduce. Semantically, it mimics a unicellular organism reproducing through mitosis.

3.2 Assigning utility

An agent's utility determines the chance its offspring will make it to the next generation; the selection process is described in Section 3.3. The utility function in effect determines the course and resulting agents of the evolutionary process. Because only a limited number of agents may occupy a piece of land, they already compete for real-estate. To realistically model competition, agents are incentivized to compete for sunlight as well; while small grasses may be very efficient at first, they stand little chance of competing against overshadowing ferns that could arise in more fertile areas.

The utility of an agent is determined by the following factors:

- The amount of *sunlight exposure* of an agents' leaves. Well-lit leaves yield a higher score, while self-occluding leaf systems or leaves that do not align their surfaces towards the light source give low scores. The amount of energy a plant receives is derived from the amount of sunlight it receives, and a larger structure requires more sunlight to provide the energy required to build it. A larger agent will therefore require more sunlight exposure in order to get a good utility score. This is explained in Section 3.2.1.

- The *stability* of a plant. Stable structures are better. Tall plants will get a lower stability score, and will have to receive benefits in other aspects to justify this strategy. This is explained in Section 3.2.2.

- The *efficiency*. A very complex genotype (e.g. a system with many unused or duplicate production rules) requires more energy and lowers utility. Additionally, seeds are very expensive to produce; while growing a very large number of them greatly increases the reproductive chances of an agent, this costs a very large amount of energy and is therefore not a viable strategy in a competitive scenario. This is explained in Section 3.2.3.

These three components yield three utility scores, which are multiplied together to produce the final utility of an agent.

3.2.1 Rating sunlight exposure

The most important goal of an agent is to produce a successful plant; the energy to produce the phenotype comes from the sun. The more effectively sunlight is captured, the more efficient a plant is. In the simulation, sunlight is modelled by projecting parallel rays from several different locations representing the sun at different times during a day, and evaluating how many rays of sunlight land on each agents' leaves. Figure 5 shows a schematic representation of sunlight projection in a two-dimensional environment.

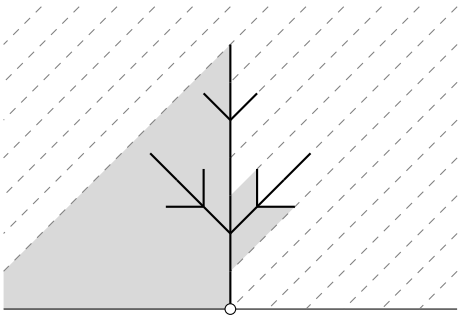

Figure 5: Sunlight is projected onto a grown Lindenmayer system from the top right, all lines in this system are treated as leaf surface. The dotted lines represent light rays. A shadow is cast by the plant, occluding its branches on the left side of its vertical axis entirely.

In this simplified two-dimensional example, most of the right side of the plant is hit by sunlight directly. Branches on the left are not directly hit when light comes from the top right. If multiple sunlight projections are performed on this shape, for example from the top and left top, almost all areas would be fully illuminated at some point during the evaluation.

The origin of light sources in a simulation strongly influences the phenotypes that will evolve; when light predominantly shines straight down (as would be the case in a ravine for example), flat horizontally aligned leaves will be preferable to other angles.

As described, the algorithm illustrated in Figure 5 only registers light exposure to material directly hit by the sun rays. This is not realistic for two reasons:

- Leaves are not fully opaque, but slightly transparent, and
- light rays tend to bounce off the surfaces they hit, illuminating things that are initially occluded as well.

To account for these effects, all leaf material has a certain *transparency factor* (which is constant throughout a simulation). This factor determines how many rays will hit the surface, and how many will pass through. If the transparency factor is 0.7, 70% of the sun rays hit a leaf they come into contact with, while 30% of them pass through and may hit anything behind it. This allows (partially) occluded leaves to produce energy as well, encouraging plants to develop denser leaf patterns to gather more energy.

Because a higher amount of leaf surface material tends to correlate with a higher sunlight exposure related utility score, plants are incentivized to produce very large leaves; this is not entirely fair. Indeed, the size of an agent is not just determined by the amount of branch material, but also by the amount of leaf material spanned between them (see Figure 3). The *leaf efficiency factor* e of an agent is therefore defined as:

$$e = \sum_{i=1}^{n_\mathrm{l}} \frac{1 - (f_\mathrm{e} \cdot a_i)^2}{n_\mathrm{l}},$$

where n_l is the number of leaves, a_i is the area of the i^{th} leaf on this agent, and f_e is a positive number that can be increased to give a higher penalty to big leaf shapes. This leaf efficiency factor is used to weigh the sunlight exposure utility; this means the utility from light exposure per surface area on a leaf is reduced for larger leaves.

The following formula gives the utility of sunlight exposure for an agent:

$$u_\mathrm{l} = \frac{h^p}{l} \cdot e.$$

The parameter h stands for the area of lit leaf material; this is the surface area of all leaf material as seen from the perspective of the sun after applying transparency (or the average surface of all sunlight projections if multiple are rendered). For instance, if an agent has a total leaf surface area of 1 on which sunlight falls unobstructed with a 45° angle, then h is $t \cdot \cos\frac{\pi}{4}$, where t is the transparency factor. l represents the number of symbols a grown agent consists of; dividing by l ensures that plants consisting of many symbols need a high amount of lit leaf material in order to be successful. Finally, p is a power applied to h, with $p \geq 1$. Increasing p effectively increases the number of symbols a system can support per exposed surface area as the total exposed surface area increases. This parameter exists because large plants need bigger support structures, while very small plants need very little or none. The sunlight exposure utility is multiplied by efficiency factor e.

3.2.2 Rating stability

Agent phenotypes are rated for their structural stability as well. The stability score is obtained from the *eccentricity* of a structure; this is the amount of deviation of the center of mass from the root of an agent. Figure 6 shows an agent phenotype with its center of mass visualized with respect to its root.

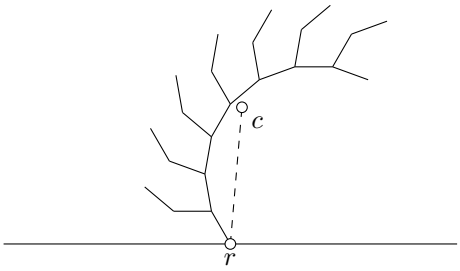

Figure 6: A two-dimensional agent with root r and center of mass c.

Using root r and center of mass c, the agent stability utility is

$$u_{\mathrm{s}} = \frac{1}{1 + f_{\mathrm{x}} \cdot ||c - r||^2},$$

where f_{x} is a positive number; it positively correlates with the utility score penalty given to unstable agents. Since c and r are three-dimensional in the simulation, agents that need to grow tall (along the vertical axis) to catch sunlight will need to reduce their instability along their other two axes with respect to their neighbours to stay competitive. On the other hand, small grasses and undergrowth will have very low or even negligible instability penalties, and can thus develop different growth strategies to obtain high utility scores.

3.2.3 Rating efficiency

Efficiency captures the effectiveness of a genotype and the cost of its reproductive faculties. The effect of this utility score is therefore twofold:

- it penalizes overly complex Lindenmayer systems, and
- it penalizes excessive seed production.

The following formula captures these functions:

$$u_{\mathrm{e}} = \frac{1}{(f_{\mathrm{r}} \cdot n_{\mathrm{r}}) \cdot (f_{\mathrm{s}} \cdot n_{\mathrm{s}})},$$

where n_{r} is the number of production rules the Lindenmayer consists of, n_{s} is the number of seeds that exists in the phenotype. f_{r} and f_{s} are positive numbers that may be increased to give higher penalties to high numbers of rules and seeds respectively.

3.2.4 Negative utility

The final utility of an agent, using the utility functions from Sections 3.2.1, 3.2.2 and 3.2.3, is $u = u_{\mathrm{l}} \cdot u_{\mathrm{s}} \cdot u_{\mathrm{e}}$. Agents with negative utility cannot be produced by this formula. There are however two exceptions for which the utility is set to -1, namely for

- agents growing underground, and
- agents over a certain size without any sunlight exposure.

The first point is obvious: since the simulation does not model roots, growing underground is by definition physically impossible. The second point is implemented to limit nonsensical phenotypes that emerge in the early phases of a simulation. The initial best strategy (when no leaves have evolved yet) is to reproduce as often and as far away as possible by growing tall structures (see Section 3.3) with many seeds. Growing such structures is not penalized by a poor sunlight exposure utility as it normally would, because this is zero for all leafless agents. To prevent unrealistic structures that support themselves without any photosynthesis, negative utility is assigned to leafless agents over a certain size.

3.3 Reproduction

Agents with a non-negative utility may produce seeds to reproduce themselves in the next generation (see Table 1). When all agents have been rated, their seeds are dispersed in an empty copy of the simulation environment to produce the next generation. Figure 7 shows how seeds are dispersed; a seed s at height h describes a cone with its apex at s and an angle α. The seed is placed on a random point on the base of the cone. The angle may vary among simulations, where more windy environments can be modelled by increasing α. Higher values of α also increase the chance of agents spreading their genotype across the natural boundaries illustrated in Figure 4.

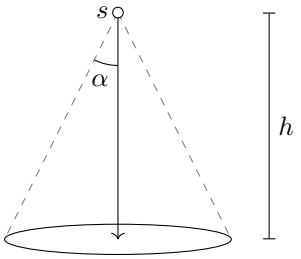

Figure 7: The seed s may fall anywhere within the base of the cone.

After seeds have been dispersed and positioned in the environment, they are ranked by their parent agents' utility. Seeds with equal utility among this order are shuffled to eliminate their positional bias. If an agent grows its seeds from top to bottom, the top seeds should not always get the chance to reproduce first; after all, their utility is exactly equal to the other seeds in that system.

The seeds produce new agents at their location, starting with the highest ranked seed. Because new agents can only be placed at locations that are not yet (too) occupied (see below), the next generation of agents consists of the best offspring, while poor agents usually (but not always) get no chance to reproduce.

To determine where agents can or cannot grow, a *density map* is maintained while seeds are placed in the environment. Figure 8 shows how agent density is assigned to the terrain surface. A grid is created for the area for which agent density needs to be known (this is normally the entire simulation environment described in Section 3.1), and every cell of this grid contains the number of agents overlapping that cell.

0	0	0	0	0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	0
1	1	0	1	1	1	1	1	1	1	1	0
1	1	2	1	1	1	1	1	1	1	1	1
1	1	2	2	1	1	1	1	1	1	1	1
1	1	3	3	2	2	1	1	1	1	1	1
1	2	3	3	2	2	2	1	1	1	1	1
1	2	2	3	2	2	2	1	1	1	1	0
1	2	2	2	2	2	2	1	1	1	1	0
1	2	1	1	1	2	2	1	1	0	1	1
0	0	1	1	1	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	1	1

Figure 8: A square portion of an agent density map. Four agents increment the cells they overlap.

The black dots represent newly placed agents, and the circle around them represents the area they occupy. The radius of the circles is chosen such that the area is equal to the area of the rectangle that the parent plant occupied in the previous generation, so with the width w and height h of the rectangular area occupied by the parent, the radius r is obtained by $r = \frac{\sqrt{w \cdot h}}{\pi}$.

Every simulation has a constant *density threshold*, which is the maximum number of agents that may overlap a grid cell for a seed to sprout when its location lies within that cell. This threshold therefore determines how dense vegetation is allowed to grow, and by extent how competitive agents need to be with respect to their neighbours; the more plants overlap, the harder they have to compete for sunlight. Conversely, simulations with a low density threshold barely experience this dynamic, and produce lower growing plants as a result. This effect is demonstrated in Section 5.

3.4 Mutation

Before the agents in the new generation are "grown", random mutation is applied to their genotypes. Production rules may be created, duplicated or removed. The probabilities of these three actions are constants of a simulation, and do not change over time. The axiom and the sentences that make up the production rules are mutated as well by inserting and removing symbols, or by creating branches or leaves around existing symbols. When mutating sentences, syntactic correctness is taken into account; every branch or leaf opening symbol has a corresponding closing symbol. Additionally, sentences may never grow larger than a preset limit to prevent overfitting; with no limit, the algorithm could simply evolve entire fixed plant structures in the axiom without ever creating any production rules.

A sentence is mutated by iterating over each symbol. At each iteration, there is a chance a mutation operation executes, while most symbols do not change. Figure 9 shows all possible mutations on sentences using the syntax outlined in Table 1. Mutations are always applied at the position of the iterator. Operations (1), (2), (3) and (5) may be applied when the iterator points to a symbol; operation (4) may only trigger at a branch open symbol and operation (6) may only trigger at a leaf open symbol. The probabilities for each of these rules to trigger whenever they are applicable are constant within a simulation.

(1) Add symbol = A<u>B</u>[C]<D] → AxB[C]<D]
→ ABx[C]<D]
(2) Remove symbol = A<u>B</u>[C]<D] → A[C]<D]
(3) Add branch = A<u>B</u>[C]<D] → A[B][C]<D]
(4) Remove branch = AB<u>[</u>C]<D] → ABC<D]
(5) Add leaf = A<u>B</u>[C]<D] → A<B][C]<D]
(6) Remove leaf = AB[C]<u><</u>D] → AB[C]D

Figure 9: Possible mutation operations on the sentence AB[C]<D]. The underscores in the left hand sides of the operations show the position of the iterator within the sentence. Mutation operations are applied at that location.

Operation (1) in Figure 9 creates new symbol x either before or after the symbol the iterator currently points to; there is a 50% chance for both outcomes. The symbol is randomly chosen from a multiset of possible new symbols. This multiset varies depending on the kind of sentence that is being mutated. Branch or leaf symbols may not occur in this multiset, since mutation rules (3), (4), (5) and (6) already cover branch and leaf mutation.

The symbol x is chosen from multiset E or N. E contains all symbols that may be *produced* by the Lindenmayer system (the symbols that can exist in the sentence obtained by applying production rules to the axiom for any number of iterations). N contains all possible symbols (that are not branch or leaf symbols) that may exist in a system according to Table 1. Figure 10 shows a Lindenmayer system and its corresponding E and N.

$$a = \texttt{B[+A][-A]BA}$$
$$r_1 = \texttt{A} \rightarrow \texttt{B[+A][-A]BA}$$
$$r_2 = \texttt{B} \rightarrow \texttt{BB}$$

$$E = [\texttt{B}, \texttt{B}, \texttt{B}, \texttt{B}, \texttt{B}, \texttt{B}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{+}, \texttt{+}, \texttt{-}, \texttt{-}]$$
$$N = [\texttt{A}\ldots\texttt{Z}] \uplus [\texttt{a}\ldots\texttt{z}] \uplus [\texttt{*}] \uplus R$$

Figure 10: Multisets E and N of a Lindenmayer system with axiom a and production rules r_1 and r_2, where R is the multiset containing all rotation symbols defined in Table 1.

When x is placed in the left hand side of a production rule, it will be chosen from E. This ensures that production rule conditions do not contain symbols that cannot occur in sentences produced by a system, these rules would never be applicable. If it is placed in the axiom or the right hand side of a production rule, it is possible to introduce symbols that do not yet exist in the Lindenmayer system. In this situation x will be chosen from N or E. The possibility of picking x from E instead of N in this case is introduced to promote reusing existing symbols in the system. This increases the chances of introducing stochastic rules, while also promoting self-similarity by increasing the chance that existing production rules will trigger on newly added symbols.

The composition of N in Figure 10 is such that R and [*] are quite rare. In the simulation, these sets are added to N several times to increase their chances of being added to a system when a new symbol is created. The number of repetitions of these sets is a constant setting of the mutator, which may be used to speed up the introduction of seeds and branch rotations as the evolution progresses.

4 Implementation

The *LGen* software [6] built to perform the experiments using the method described in Section 3 was implemented in the *C++11* programming language and targets both Windows and Linux platforms. Figure 11 shows the different modules that make up the software. *LGen* performs, stores and loads experiments and configurations. *LParse* parses Lindenmayer systems according to the syntax set forth in Section 2, generating the strings of symbols that are translated into agent phenotypes. *LRender* models and renders populated three-dimensional simulation environments using *OpenGL* according to the algorithm described in Section 2.1, and provides information about these renderings (ėġsunlight exposure ratios, structure size, and seed locations). *LGen* and *LParse* are both used by *LGen*, but do not require or reference each other.

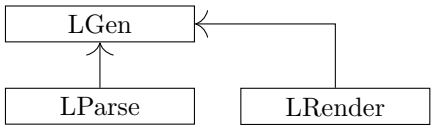

Figure 11: Structure of the *LGen* software.

4.1 Stages

The algorithm is divided into several stages per generation. Figure 12 shows the stages in order of execution that are required to simulate a generation.

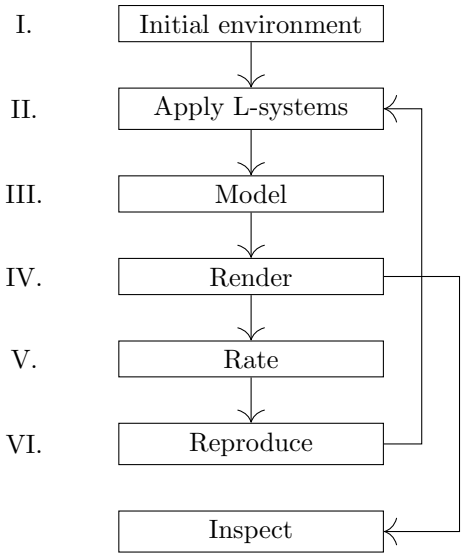

Figure 12: The stages of the simulation loop.

Stage I initializes the environment, including for example its terrain function and seed dispersal angle (see Section 3.3). In practice, this step is done manually by the user. The initialized simulation can be stored for later use, or to perform different experiments on a set of equal initial conditions.

Subsequently, Stage II creates a string of symbols for each agent by applying their system's production rules to their axioms for a number of iterations that depends on the fertility of the terrain at their respective locations. This process is multithreaded, since no interaction between agents takes place at this stage.

During Stage III, a three-dimensional model of the environment and all its agents is created by applying the modelling rules detailed in Section 2.1. Again, this stage uses multithreading to speed up the modelling process. Summarizing information about agent models, like their size and center of gravity, is stored for use in Stage V.

At Stage IV, the GPU is employed to render the scene modelled in Stage III from different angles, where each angle is one of the sunlight directions. Light exposure characteristics for each agent are stored for use in Stage V.

Stage V assigns utility to every agent in the simulation. The utility function takes information gathered by Stages III and IV as well as the Lindenmayer system that gave rise to this agent into account.

Finally, Stage VI creates a list of all seeds produced in the environment, ranks the seeds by the utility assigned to their parents by Stage V, and disperses the seeds according to the algorithm described in Section 3.3. New agents are created from these seeds after mutating their systems. The simulation can now loop back to Stage II and evolve the newly created generation.

The results of the simulation can be queried whenever Stage IV has finished; the software can render the simulation at its current stage, and individual agents can be inspected by querying their Lindenmayer systems and utility score. When presenting a rendered environment to the user, a high-detail scene is modelled; the program uses lower quality agent models while simulating to increase performance.

4.2 Randomization

Randomization is used at several points in the simulation:

- Stochastic Lindenmayer systems (as described in Section 2) may need to choose a random production rule from a list.
- The initial direction of the turtle when modelling an agent is always pointing upwards, but the rotation around the vertical axis is randomized.
- When dispersing seeds (see Section 3.3), randomization is used to displace them.
- Mutating Lindenmayer systems makes heavy use of randomization; small random chances trigger certain mutations.

The implementation uses *Mersenne twisters* [7] for randomization. An important feature of the program is that its randomizer can be *seeded*, after which the simulation is completely deterministic; when running the same seeded setup twice, the same results emerges. This feature makes simulation runs reproducible, but it reduces storage space as well. When a simulation of a high number of generations needs to be written to disk completely, it only stores the state every n generations along with the state of its randomizer. Consider a stored simulation of at least 4000 generations. If $n = 1000$ and the 3511^{th} generation needs to be inspected, the system loads the 3000^{th} generation instead (which was saved since $n = 1000$), and simulates for another 511 generations to obtain the state as it was at the 3511^{th} generation.

4.3 Leaf area projections

To implement the sunlight exposure algorithm described in Section 3.2.1, the tree-dimensional scene produced in Stage III (see Figure 12) is rendered using an orthographic projection. Figure 13 shows two renderings; on the left, the modelled scene is rendered

for viewing purposes using a perspective projection. On the right side, the parallel projection is used to detect sunlight exposure in the same scene. Using this projection ensures that the produced pixels can be seen as hits by the parallel rays of sunlight in Figure 5.

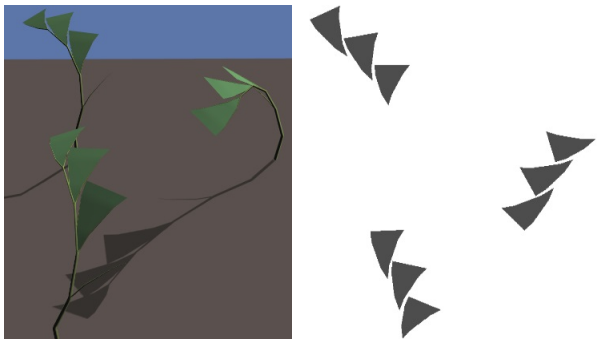

Figure 13: A three-dimensional rendering of three agents on the left, and the corresponding exposure projection taken from approximately the same angle on the right. Note that the phenotype has evolved such that the leaves do not occlude each other, while still being as compact as possible.

In this *exposure projection*, any pixels that contain leaves get a dark color, while the rest of the image stays white. The exposure area of agents can be obtained by counting the "hit" pixels in this rendering for each agent. Pixels do not only contain information about whether a leaf was hit there, but they also encode which agent was hit. Additionally, the algorithm uses a depth buffer; the leaves closest to the camera (which represents the sun in this model) receive sunlight while occluded leaves do not. If the transparency factor is smaller than 1 however, every pixel of exposed material has a chance equal to that factor of not being rendered, allowing occluded leaves to receive sunlight as well.

5 Results

When the simulation environment is initialized with the simplest reproducible agents, namely agents with only a single seed symbol in the axiom and no production rules, the course of the evolutionary process goes through five distinguishable stages.

1. The first very simple agents produce tiny phenotypes. In most agents, no production rules to grow the systems exist yet, and if they exist, the rules usually do not produce growth. Axioms contain one or only a few symbols. During reproduction, some agents lose the ability to produce a single seed, causing the number of agents to drop slightly during the first generations of evolution.

2. Some agents develop methods to produce more than one seed, and the number of agents in the environment begins to increase. This is the first successful strategy that agents use to compete; they are always more successful than their predecessors regardless of utility, because they can rapidly populate all free space in the environment while agents with a single seed can reproduce at most once.

3. Agents start to develop tall structures, enabling them to disperse their seeds over large areas.

4. The first leaves evolve. Their crude initial shapes often yield low utility (they are often too big, intersect each other or are occluded by the branching structures), but nevertheless lift their parents' utility over zero for the first time in the simulation. After all, according to the formula described in Section 3.2.1, all earlier leafless agents had a utility of zero.

5. The agents start to produce better leaves, and will often develop the ability to produce multiple leaves as well, depending on the configuration of the utility function. The utility scores increase significantly. At this point, agents will start to compete for sunlight with their neighbours. If multiple agents with leaves yield the same utility score without neighbour occlusion, they outcompete their neighbours by growing slightly taller; this lowers utility a little, but dramatically decreases their competitors utility by occluding their sunlight. When a simulation consist of separated fertile area's, different phenotypes emerge in each of them. Sometimes, agents manage to "escape" their habitat and spread their genotype to a different area. This effect is demonstrated in Figure 19.

Renderings of an example simulation showing each of these five stages are shown in Figures Figures 25 to 29 respectively; they can be found in Appendix A. The configuration used to define this simulation is included in Figure 30 in the Appendix A.

The simulation does not seem to converge once the last phase starts; because of the competitive dynamic, it is never a viable strategy to stop changing. Agents keep "reacting" to strategies that emerge around them. The genotype keeps changing through

mutation during this phase; after a few thousand generations, genotypes that where previously dominant cannot be found or recognized any longer.

5.1 Competition

One of the focal points of this method is to simulate competition among agents. The degree of competition that exists during a simulation can be influenced by chancing the density threshold of a simulation (as described in Section 3.3); when this number is very low, agents will not reproduce close to each other, preventing them from competing for sunlight. Figure 14 shows the influence of the density threshold on a simulation; the results of the absence of competition are severe.

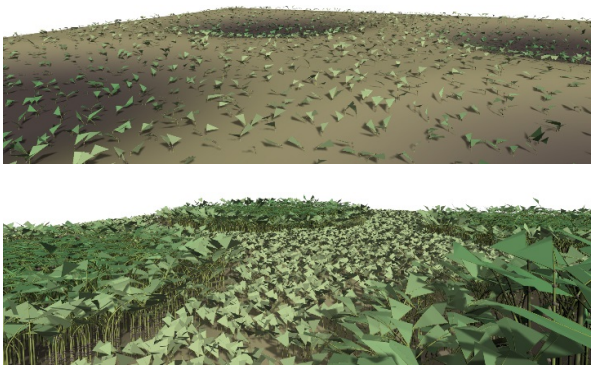

Figure 14: Two versions of a simulation at $t = 20000$ demonstrating the importance of competition. The top version has a density factor of 2, the bottom version has a density factor of 26. All other parameters, including the initial state of the environment, are the same.

Without competition, complex structures do not evolve. This can be attributed to the fact that growing larger plants or more leaves always lowers utility. The only reason for evolving complex strategies in this simulation is to compete against other agents. At the less fertile areas around the valleys, agents remain simple as well, because large structures cannot grow there (see Section 3.1, which details how agent structure sizes are limited and growth is slowed down as fertility decreases). Plant size and complexity therefore increases where a combination of fertile ground and competition takes place.

Figure 15 shows an environment where fierce competition takes place. Multiple layers of broad leaves try to catch all available sunlight.

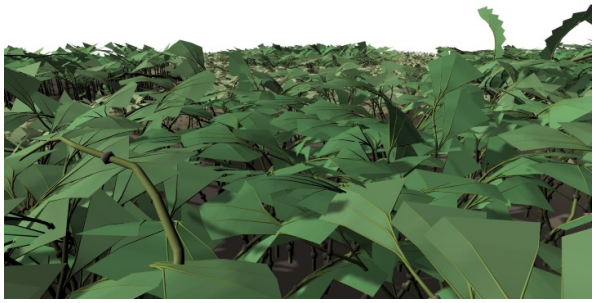

Figure 15: An environment resulting from a simulation with a high density factor. Competition causes plants to grow leaves wherever sunlight can be caught, and no open areas exist. Some agents in the top right try to escape the canopy to avoid occlusion.

5.2 Leaves

As soon as leaves evolve in a simulation, agents start to compete for sunlight exposure. Figure 16 shows plants at an early stage during a simulation; they have recently evolved leaves, and do not yet grow tall in order to compete with their neighbours. They do however produce two seeds instead of one, which causes them to spread through the environment unless a better agent prevents them from reproducing.

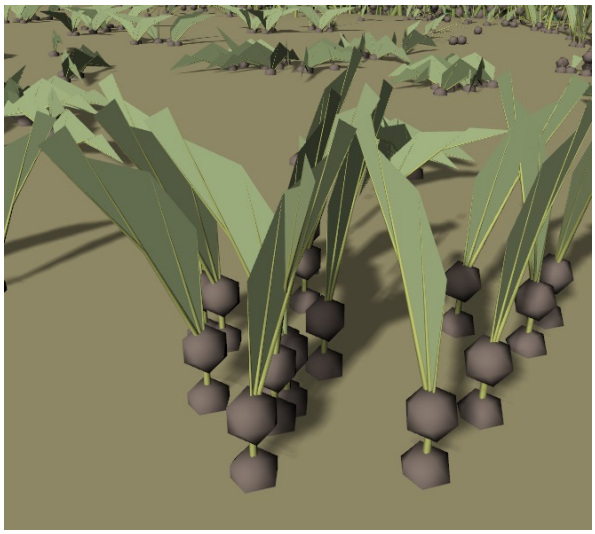

Figure 16: Agents that develop early on in a simulation are often small and vertically oriented. This rendering shows a small colony of low growing plants. At this stage, competition starts to play a role; the plants cast shadows on their neighbours, reducing the amount of sunlight they receive. These plants grow in clusters, since their seeds (the brown spheres) are located close to the ground, preventing them from spreading far away.

The plants in Figure 16 produce a leaf that is divided into segments; this increases utility, since the formula from Section 3.2.1 penalizes large leaf areas. This strategy of segmenting leaves almost always arises in simulations.

Figure 17 shows the same simulation at a later stage, when competition between the plants in Figure 16 causes them to compete with one another. This rendering is taken at one of the more fertile locations in the environment, where the agents are able to grow larger and develop competitive strategies.

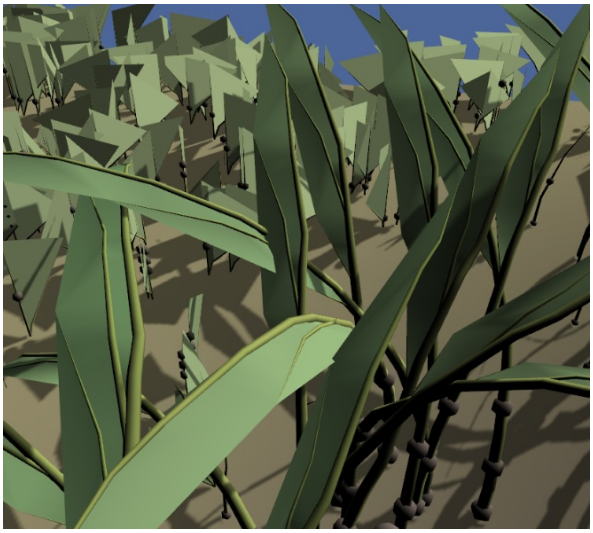

Figure 17: The descendants of the plants in Figure 16 are taller and more competitive than their ancestors, while remaining structurally similar to their ancestors.

The same segmented leaves have grown larger where the ground is fertile enough to support larger structures. Since plants need to compete for sunlight because of their proximity, the leaves now start higher on the structures. Their shapes do no longer grow vertically, but bend towards the sky as well in order to catch more light. The common ancestor these plants have evolved from would no longer thrive among them, since its descendants would catch most of the light before it reaches the agents closer to the ground.

While the leaves in Figures 16 and 17 evolved from small plants, the first leaves in a simulation can develop on tall plants as well. Figure 33 in Appendix A shows a rendering of a simulation where the first leaves have evolved on tall inefficient plants; their more successful descendants are smaller and more efficient.

When sunlight only shines directly down (as opposed to multiple light directions that also cast some light from the sides), plants align their leaves horizontally. Figure 18 shows an agent taken from such an environment.

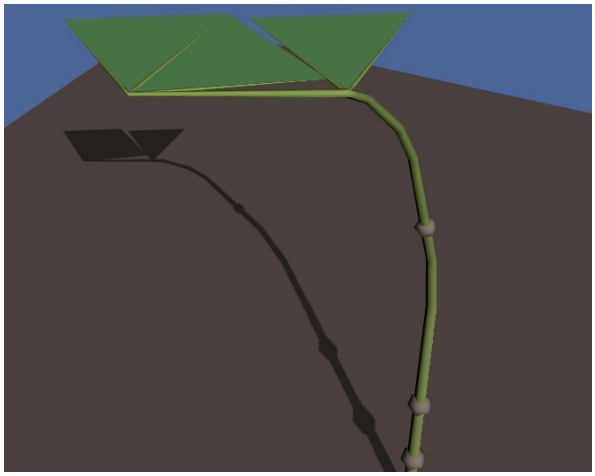

Figure 18: This agent has evolved in an environment where light only shines directly down. It is taken from a simulation with many competing agents, so the leaf is placed on a high stalk to prevent occlusion.

5.3 Structural strategies

Initially, most agents develop as a single branch with one or more leaves connected to it. The tendency to produce phenotypes with many similar leaves or few dissimilar leaves develops at an early stage during the simulation. Figure 31 in the appendix shows a rendering of a simulation during an early stage where the first leaves occur only once on their parent plants. In contrast, Figure 32 shows agents developing multiple similar leaves per agent.

In simulation environments where multiple fertile areas exist, like the one shown in Figure 4, different genotypes will be dominant in different areas. They may however share a common successful ancestor that managed to spread its seeds over the infertile barriers dividing the environment. Figure 19 shows the result of such an event. These four agents share the same growth pattern, but employ it slightly differently to adapt to the different contexts they have evolved in. The top right agent has concentrated its leaf mass as high as possible, likely to prevent occlusion. The agent right below keeps increasing its leaf sizes as they develop, the oldest leaves have the biggest surface area.

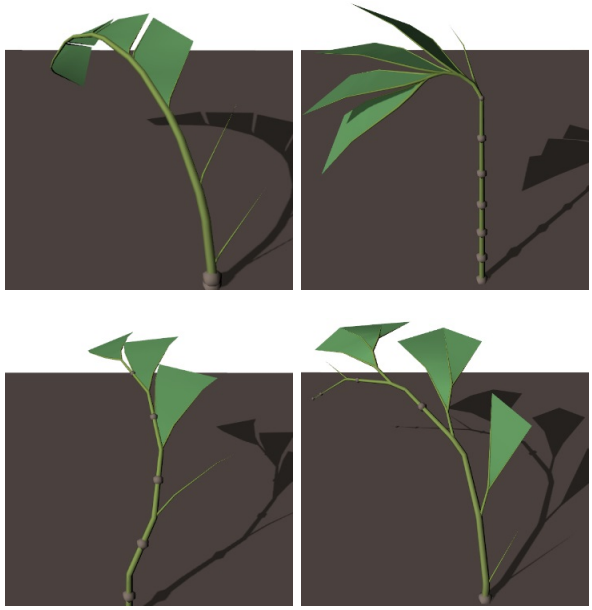

Figure 19: Four different agents with a common ancestor that existed 1000 generations earlier. All agents share the same structural strategy; new leaves form at the root, starting as a branch and growing into leaves after multiple iterations. Various specializations have emerged.

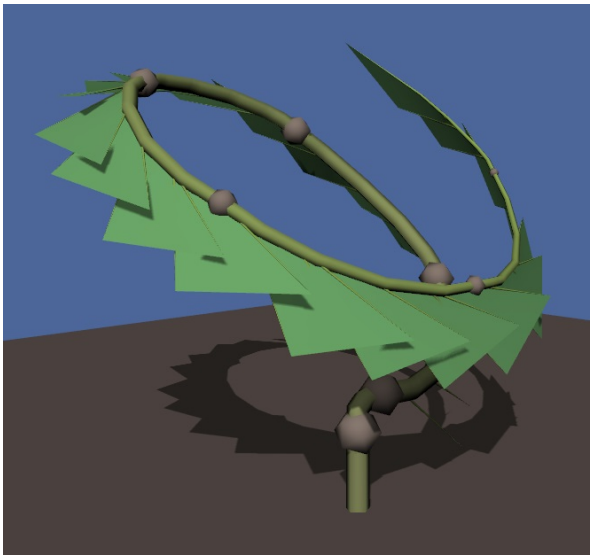

Figure 20: An agent with a lot of branch and leaf material. Note that the plant structure is inclined such that the center of gravity is approximately above the root, reducing the eccentricity demonstrated in Figure 6.

Because of the stability criterion described in Section 3.2.2, agents only grow tall when required, and if they do, their structure will form mostly directly above their center of gravity. Figure 20 shows how an agent with a large complex structure managed to keep its eccentricity low.

5.4 Natural counterparts

Running the simulation for a sufficient number of generations gives rise to agents that employ effective strategies analogous to those found in nature. Figure 21 shows an agent that has developed a similar strategy to the lily of the valley plant; a vertical stem contains the reproductive systems (berries in this case), while broad leaves emerging from the same root spiral around the central axis.

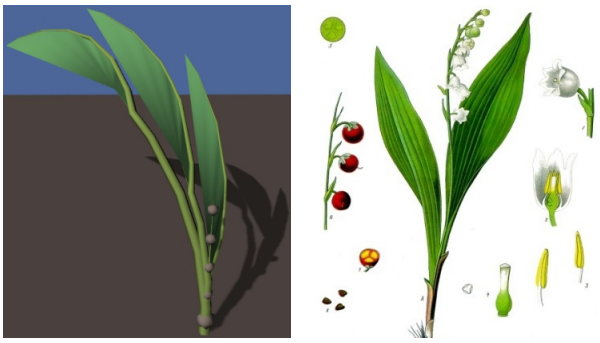

Figure 21: An illustration of a real "lily of the valley" flower on the right [8] compared to an evolved agent on the left.

Figure 22 shows a group of agents that have developed thin fanning leaves that tend to align themselves on the same plane, a strategy also seen in palm leaves.

Figure 22: Palm leaves photographed in a greenhouse of the Oxford botanic garden on the right compared to a population of evolved agents on the left.

Figure 23 shows a close up rendering of evolved leaves compared to real leaves. The leaf efficiency factor from Section 3.2.1 causes well nerved leaves to

yield higher utility than unsupported large leaf areas, resulting in well structured leaves similar to the ones found in nature.

Figure 23: Clearly nerved leaves photographed in the Oxford botanic garden on the right juxtaposed to evolved leaf nerves on the left. Note that both leaf structures separate their leaf surfaces at regular intervals, and older leaves have more nerves.

6 Discussion

6.1 Default leaf shapes

The syntax described in Section 2 and illustrated in Figure 3 uses a relatively complex method to define leaves. The process of modelling "good" leaves is arguably more complex than modelling the structures that support them.

To evaluate whether the complexity of the proposed leaf modelling process does not impair the course of the simulation, a *default leaf symbol* can be introduced; when the turtle encounters this symbol, a triangular leaf is created at its position and orientation. The chance to create leaf brackets while mutating is set to zero. Figure 24 shows a rendering of a simulation using this alternative method compared to a simulation using the standard method. When default leaf shapes are used, agents with leaves evolve very quickly. Given enough time however, a simulation using the old method produces very similar agents. This shows that agents with multiple similar leaves evolve quicker when default leaf shape symbols are used, but similar results can be achieved without it; the proposed leaf modelling method does not seem to impair the course of the simulation.

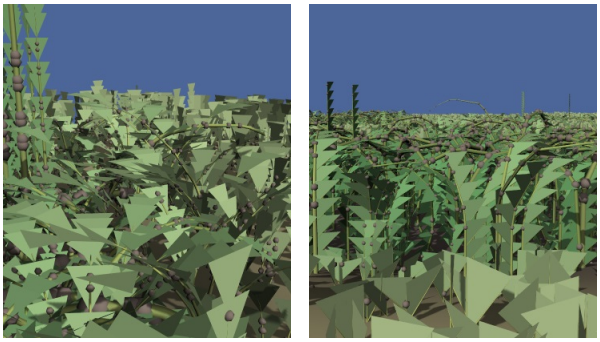

Figure 24: The simulation on the left at $t = 40$ uses a default leaf symbol to model triangular leaves. The simulation on the right at $t = 800$ uses the normal leaf modelling syntax from Section 2.1 and produces very similar agents to the alternative method, although it takes more generations to evolve them.

6.2 Further development

The proposed method combined with the implementation [6] provides a framework that supports further development.

6.2.1 Climate

The utility function in this simulation method remains constant throughout the simulation. While agents adapt to each other and the fertility of the area they grow on, they do not adapt to a climate system, which varies greatly over time in reality. If such a system were implemented, agents could be incentivized to produce robust strategies that remain successful when shared conditions change rapidly. An example in the context of this implementation would be a global drought; terrain fertility could drop globally in a short time, and restore itself slowly afterwards. Complex agents that do not thrive when being grown for fewer iterations would go extinct, while smaller more robust agents get a chance at success they did not have before.

A simpler method that can be implemented to promote robust populations is a simulated forest fire that starts at a random place and burns away dense vegetation until a sparse area is reached.

6.2.2 Continuous evaluation

The implementation evaluates the scene as a whole before erasing it and producing an entirely new generation (see Figure 12). This treats all agents in the scene as if they all reach the exact same age before reproducing. In reality, the length of a plant's lifespan and the time at which reproduction takes place

may be part of its evolutionary strategy; some plants grow very old and reproduce rarely (like cacti), while others have very short lifespans during which they reproduce very often (like grass or moss).

The simulation could be evaluated at every iteration of growth instead. Seeds could be dispersed at any iteration of growth, allowing agents to reproduce many times during their lifespan, or to concentrate their energy towards growth initially and reproduce later, like most real organisms do. Agents can die when they receive too little sunlight for a time, or when they decide to do so (only very few organisms cannot die from old age, since living too long is often a poor evolutionary strategy).

6.2.3 Reproductive systems

The used reproductive model is simplified to a high degree. No *crossover* takes place; this is the process whereby new agents are not merely mutated clones of their parents, but rather a mutated mix of two parents [9]. This allows for combining evolved traits from two parent agents in their offspring. Crossover could be implemented in this simulation by "pollinating" each seed before reproduction. This can be simulated by finding a nearby similar agent (with a Lindenmayer system that is not too different from the agents' own system, effectively determining whether the two agents are the same species) and mixing the properties of these two systems through crossover.

Furthermore, all seeds in the simulation are treated equally. In realistic environments, selective pressure applies to reproductive organs as well. Flowers for example do not merely compete by growing healthy plants, but also by developing colourful flowers that attract pollinators. This can be implemented in the proposed simulation by "decorating" seeds with flower petals, and multiplying their utility by an *attractiveness factor* based on how much the seeds stand out in their environment. Many agents produced by the current method produce seeds somewhere on their branches, because there is no disadvantage in doing so; in reality, these seeds would be hidden behind leaves and stand little chance of pollination.

7 Conclusion

We have demonstrated that the criteria used by the simulation give rise to agents that exhibit strategies similar to those shown by real plants in the natural environment it aims to imitate (see Section 5.4).

The evolutionary process itself displays realistic biological characteristics as well, e.g. divergent evolution (see Figure 19) and competition (see Section 5.2). The heterogeneous environment results in multiple different "species" of agents, that may reach each others habitat and outcompete one another.

One of the aims of this method is to allow the temporary survival of sub-par agents to allow for different agents to evolve without immediately culling low utility agents. The method does indeed allow for the survival of agents based on multiple criteria; utility plays a role in selection, but reproductive systems, terrain fertility and vegetation density play a critical role as well (Figures 25, 26 and 27 show agents with zero utility competing on other grounds).

On these aspects, the biological analogy of Lindenmayer systems has been extended by evolving them through the proposed simulation method.

References

[1] A. Lindenmayer, "Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs," *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 300 – 315, 1968.

[2] C. MacKenzie and P. Prusinkiewicz, "Artificial evolution of plant forms," *Proceedings of the Fifth Annual Western Computer Graphics Symposium*, 1993.

[3] G. Ochoa, "On genetic algorithms and Lindenmayer systems," 1998.

[4] C. Jacob, "Evolving evolution programs: Genetic programming and L-systems," pp. 107–115, 1996.

[5] T. Burt, "Interactive evolution by duplication and diversification of L-systems," 2013.

[6] J. J. V. Talle, "Lgen," May 2019. Available at `https://github.com/jobtalle/lgen`.

[7] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, pp. 3–30, Jan. 1998.

[8] F. E. Köhler, "Lily of the valley illustration," June 2019. Available at `https://en.wikipedia.org/wiki/Lily_of_the_valley`.

[9] J. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

Appendix A

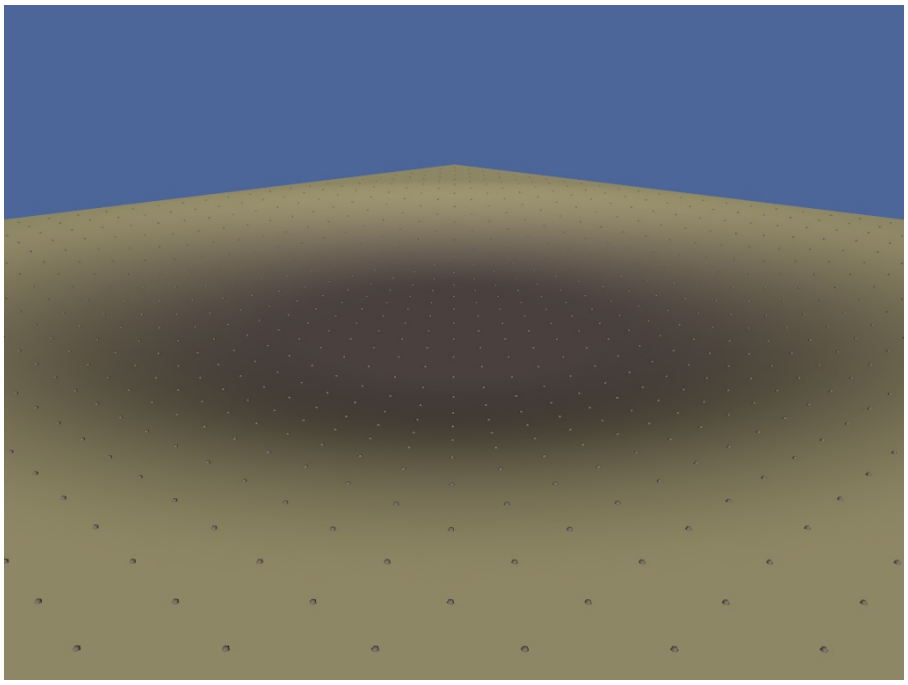

Figure 25: The initial state of a simulation at $t = 0$, only producing agents with a single seed. The agents are evenly spaced in this figure, because no iterations have been simulated yet.

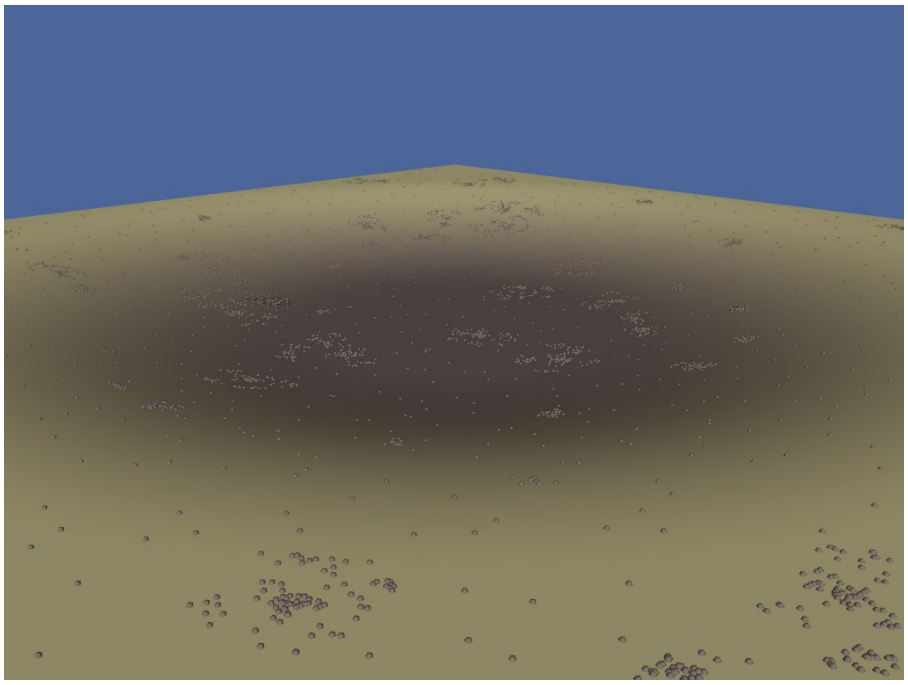

Figure 26: A simulation at $t = 20$ after the first agents producing multiple seeds have evolved. Small clusters of these agents emerge and are spreading through the environment.

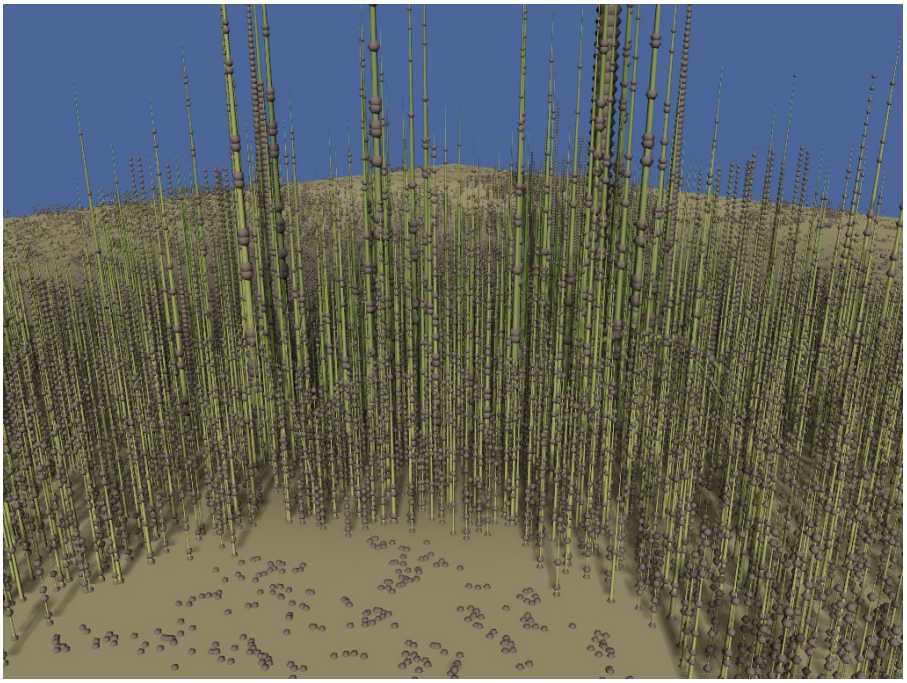

Figure 27: At $t = 80$, agents develop tall growing structures to disperse their seeds over a large area. This rendering shows the simulation at a stage right before the tall structures overgrow the entire environment.

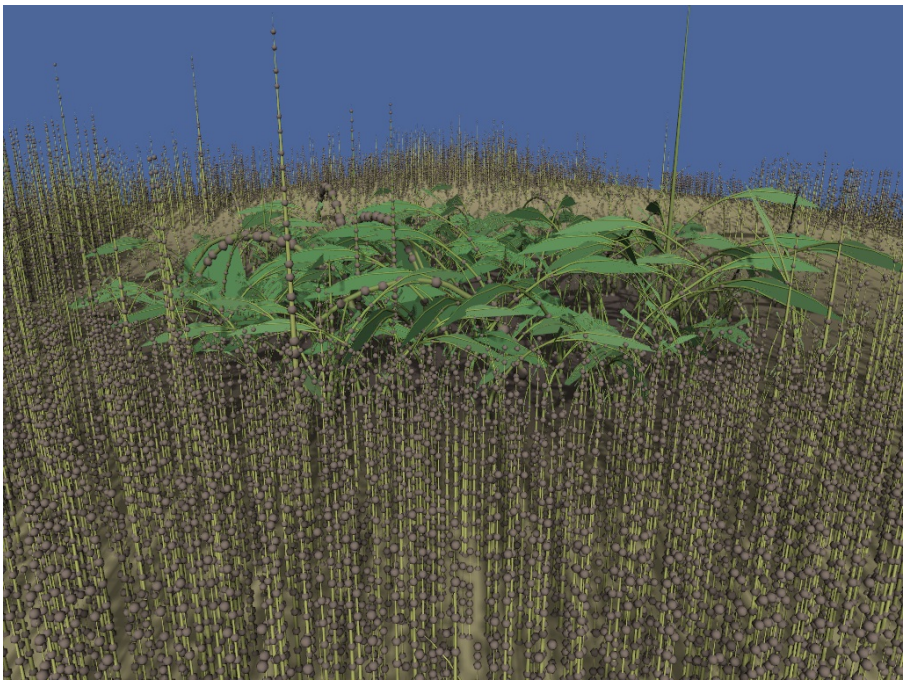

Figure 28: The first leaves have emerged in the simulation at $t = 130$. These agents are the first ones with a nonzero utility; they are rapidly replacing the existing population. These plants are not very efficient yet.

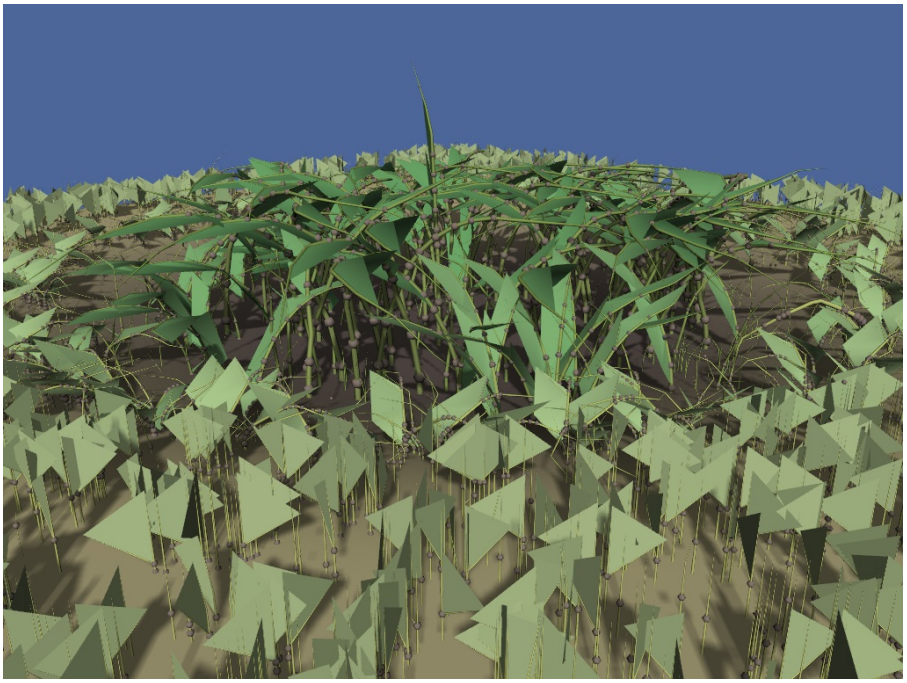

Figure 29: Leafless agents no longer exist in the simulation at $t = 2084$. More complex plants develop in the fertile central valley, simpler agents live on the edges of the environment.

```
density: 27.000000
generation: 0
initial: {...}
randomizer: {...}
mutator: {
  branch-add: 0.002000
  branch-remove: 0.002000
  leaf-add: 0.002000
  leaf-remove: 0.002000
  rule-add: 0.001000
  rule-duplicate: 0.003000
  rule-remove: 0.004000
  symbol-add: 0.005000
  symbol-chance-constant: 0.200000
  symbol-chance-new: 0.200000
  symbol-chance-rotation: 0.350000
  symbol-chance-seed: 0.050000
  symbol-chance-step: 0.400000
  symbol-remove: 0.005000
}
```

Figure 30: The configuration of the demonstrated simulation is printed in plain text. The initial state, rendered in Figure 25, is omitted; this is simply an evenly spaced layout of single seed agents. The state of the randomizer is also omitted on account of its size.

Appendix B

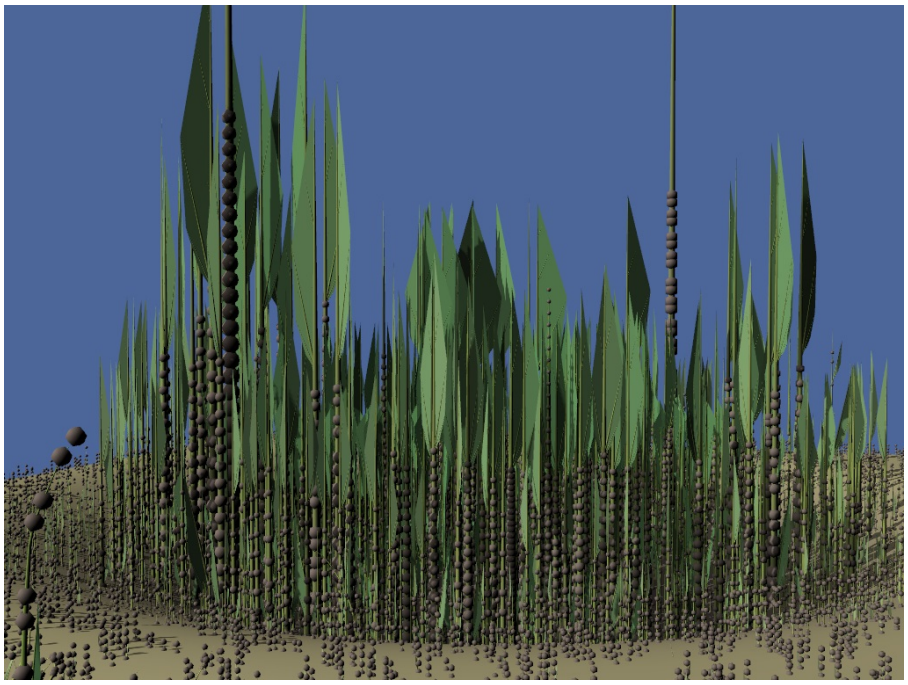

Figure 31: The first leaves have developed in this simulation. They are narrow and segmented, and they do not repeat themselves within the structure.

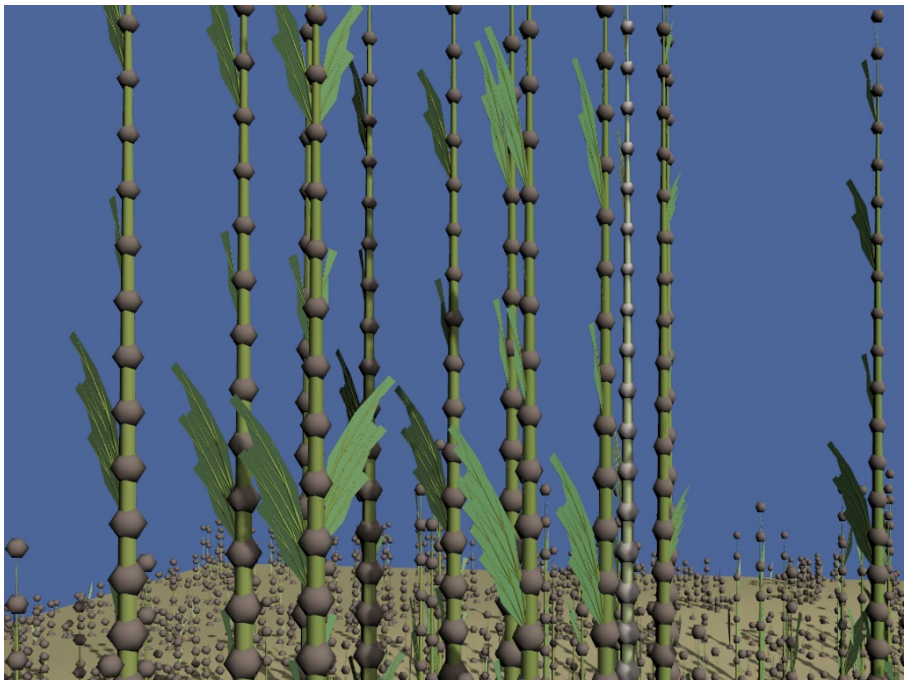

Figure 32: This simulation is in a similar stage to the simulation rendered in Figure 31, but instead of developing a single leaf per plant, stalks contain multiple similar leaves instead.

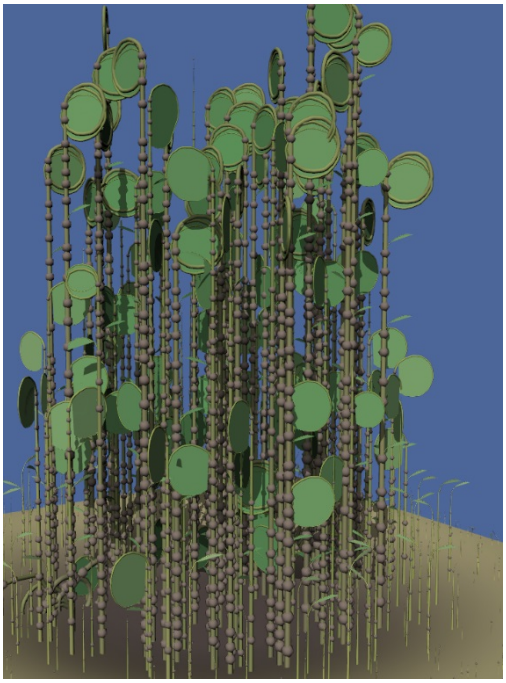

Figure 33: The leaves on these agents have developed early on during a simulation on agents with phenotypes similar to those in Figure 27. The leaves are curled up unnecessarily, and the number of seeds produced on the stalk is inefficiently high.