Evolving L-systems in a competitive environment

Job Talle[0000−0003−1946−8407] and Jiří Kosinka[0000−0002−8859−2586]

Bernoulli Institute, University of Groningen, the Netherlands
jobtalle@hotmail.com, j.kosinka@rug.nl

Abstract. Lindenmayer systems (L-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 through competition in a heterogeneous environment to further expand on this biological analogy. The resulting simulations demonstrate evolution driven by competition, resulting in agents employing strategies similar to those found in nature.

Keywords: Lindenmayer system · L-system · competitive environment · plant evolution

1 Introduction

In addition to the established biological realism of Lindenmayer systems (L-systems for short) [15], a model is proposed to evolve systems as agents representing plants in an environment designed to maintain this analogy in several ways (Figure 1). Evolutionary algorithms have been deployed to optimize L-systems before, in two-dimensional environments [17, 21] as well as in three-dimensional environments [13, 3, 4]. Drawing on these past achievements, the focal points of the proposed model are:

– using the smallest possible subset of L-system syntax rules to avoid steering the algorithm towards a preferred direction,
– 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 reflect their biological counterparts, the simulated organisms should be able to succeed through strategies employed by real plants. Our simulation system is thus designed to evoke realistic behaviour by simulating a realistic environment.

Elaborate models have been developed to model ecosystems and interactions between them [18]. Our aim is not to model existing ecosystems top down, but

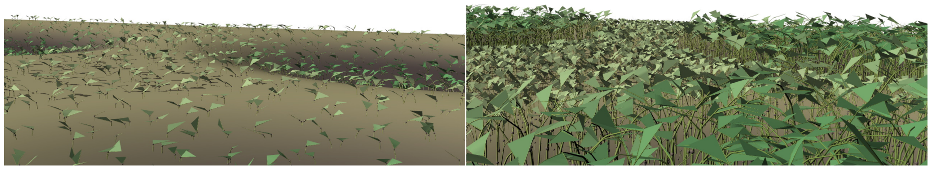


Fig. 1: *Two versions of a simulation at generation $g = 20000$ demonstrating the importance of competition: Density factor of* 2 ***(left)*** *and* 26 ***(right)****. All other parameters, including the initial state of the environment, are identical.*

rather to make a model that develops properties similar to real world ecosystems based on a minimal number of criteria analogous to the most important constraints that all plants encounter. Our main contribution is to show that even this minimalist and unbiased model gives rise, through competition, to realistic strategies employed in nature.

After discussing related work in Section 2, we present and analyse the results of our simulations in Section 3. We discuss our findings in Section 4 and conclude the paper in Section 5. The interested reader may find details regarding our syntax, method, and implementation in Appendix 1, 2, and 3, respectively.

2 Related Work

In [3], the authors also consider competition in a virtual plant community. They use a very elaborate model for plants and the environment they are put in, and their competition simulation is built directly using pre-defined tree populations (beech and oak). In contrast, our simulations do not predefine any specific plant structure, and rather start from equivalent and very minimal seeds. They also generate the next generation from a set of *elite members*, a fixed size set containing agents with the highest fitness. In contrast, agents in our method only reproduce locally, and define their fitness relative to their immediate neighbours. We show that even with our generic model, the plants develop similar strategies to those observed in [3], such as developing tall trunks to win the ‘arms race’.

The suitability of genetic algorithms for simulating plant evolution has been established [17, 25, 9]; see also [24, 10] for overviews. We build on top of these studies by introducing direct competition to the environment. Although we are not the first to consider competition in the setting of evolving plant agents in a realistic environment, existing approaches, including [1, 6, 7], only estimate sun exposure, and do so without considering individual leaves. In contrast, we simulate occlusion by placing agents with their leaves next to each other in a simulated 3D environment instead of approximating it.

The shapes generated by applied L-systems can be rated using a fitness function, after which selection can be applied to obtain more desirable shapes. Different selection criteria can be defined for two-dimensional shapes created by *turtle graphics* [21]. Our method extends these shapes to 3D, again using turtle graphics driven by L-systems (as opposed to other approaches such as Xfrog [16]

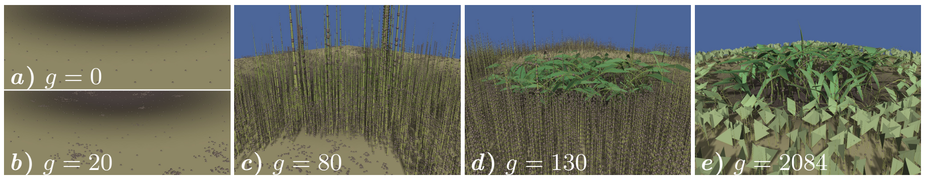


Fig. 2: *Five recognizable stages in the evolution process.* ***a)*** *The initial state of a simulation at* $g = 0$*, only producing agents with a single seed. The agents are evenly spaced in this figure, because no iterations have been simulated yet.* ***b)*** *The simulation at* $g = 20$ *after the first agents producing multiple seeds have evolved. Small clusters of these agents emerge and are spreading through the environment.* ***c)*** *At* $g = 80$*, 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.* ***d)*** *The first leaves have emerged in the simulation at* $g = 130$*. These agents are the first ones with a nonzero viability; they are rapidly replacing the existing population. These plants are not very efficient yet.* ***e)*** *Leafless agents no longer exist in the simulation at* $g = 2084$*. More complex plants develop in the fertile central valley, whereas simpler agents live on the edges of the environment. Simulating until* $g = 2084$ *took approximately* 40 *minutes on a standard desktop computer.*

which relies on predefined building blocks and user modelling). We use this not only for rendering, but also for determining sun exposure, based on grown leaf material. Although more precise models for computing sun exposure [5, 26, 2] and other features [22, 12, 23] exist, our rendering-based model strikes a balance between efficiency, and computational and implementation complexity while still giving rise to natural strategies based on competition.

Non-homogeneous environments will cause agents with equal genotypes to develop different phenotypes [19]. If the environment does not permit it, a plant will not grow to its full potential. Plant growth is not hindered during the growth process since this would require significantly more complex L-systems [8], but rather by the environment and the viability score given afterwards. Evolutionary strategies in nature are often tailored to specific environmental conditions; our method aims to simulate this effect in order to encourage the differentiation of evolutionary strategies by which greater biodiversity develops.

3 Simulations and Results

When the simulation environment is initialized with the simplest reproducible agents (Appendix 2), namely agents with only a single seed symbol in the axiom (Appendix 1) and no production rules, the course of the evolutionary process (our implementation is available; see Appendix 3) goes through five distinguishable stages; see Figure 2.

a) 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 in the first generations.

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

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

d) The first leaves evolve. Their crude initial shapes often yield low viability (they are often too big, intersect each other or are occluded by the branching structures), but nevertheless lift their parents' viability over zero for the first time in the simulation. After all, according to the formula described in Appendix 2, all earlier leafless agents had a viability of zero.

e) 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 viability function. The viability 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 viability a little, but dramatically decreases their competitors' viability by occluding their sunlight. When a simulation consist of separated fertile areas, 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 7.

The simulation does not seem to converge once the last phase starts; due to the competitive dynamic, it is never a viable strategy to stop evolving. Agents keep "reacting" to strategies that emerge around them. The genotype keeps changing through mutation in this phase; after a few thousand generations, genotypes that were previously dominant cannot be found or recognized any longer. Agent viability is initially very low, since leaves have not evolved yet. When the first good leaves evolve, an optimum is reached. After this peak, the viability drops slightly again because competition causes agents to develop structures that yield lower viability, but increase their competitive strength.

3.1 Competition

One of the focal points of our method is to simulate competition among agents. The degree of competition that exists in a simulation can be influenced by chancing the density threshold of a simulation; when this number is very low, agents will not reproduce close to each other, preventing them from competing for sunlight. Figure 1 shows the influence of the density threshold on a simulation; the results of the absence of competition are severe.

Without competition, complex structures do not evolve. This can be attributed to the fact that growing larger plants or more leaves always lowers viability. The only reason for evolving complex strategies in this simulation is to

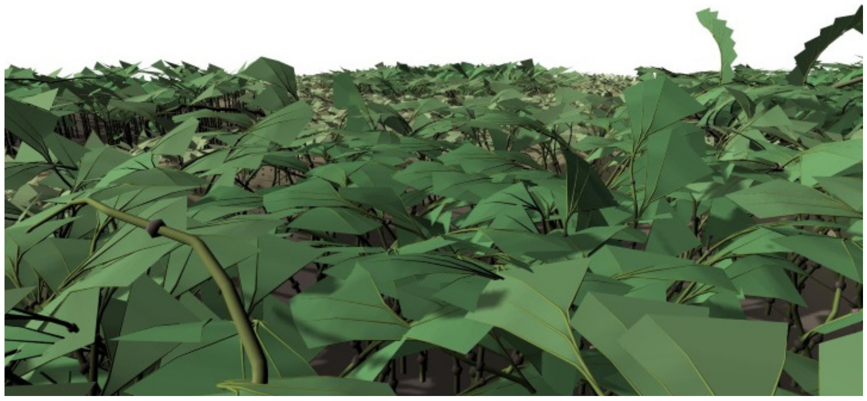


Fig. 3: *An environment resulting from a simulation with a high growth density factor. Competition causes plants to grow leaves wherever sunlight can be caught; no open areas exist. Some agents (top right) try to escape the canopy to avoid occlusion. When density increases, agents employ increasingly complex strategies.*

compete against other agents. At the less fertile areas around the valleys, agents remain simple as well, because large structures cannot grow there. Plant size and complexity thus increase where a combination of fertile ground and competition takes place. Figure 3 shows an environment with fierce competition. Multiple layers of broad leaves try to catch all available sunlight.

3.2 Leaves

As soon as leaves evolve in a simulation, agents start to compete for sunlight exposure. Figure 4, left, 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.

The plants in Figure 4, left, produce a leaf that is divided into segments; this increases viability, since the plant efficiency formula detailed in Appendix 2 penalizes large leaf areas. This strategy of segmenting leaves almost always arises.

Figure 4, right, shows the same simulation at a later stage, when competition between the plants in Figure 4, left, 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.

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 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

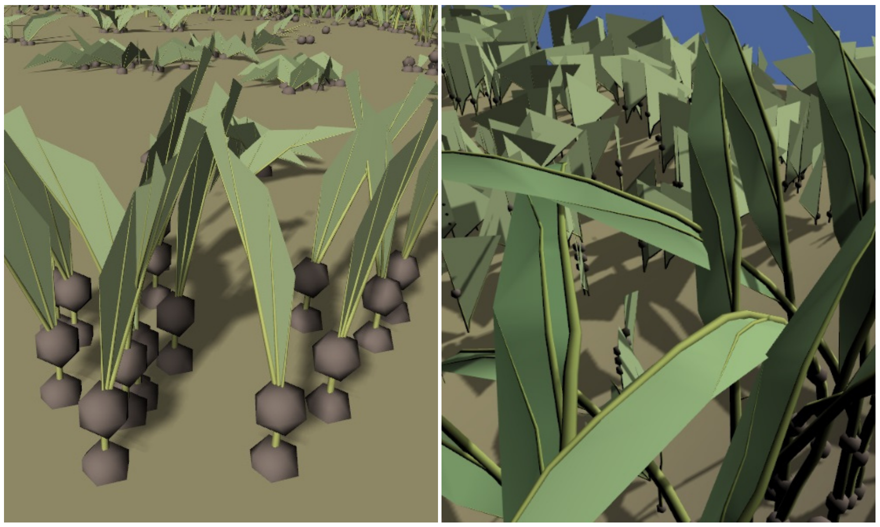


Fig. 4: ***Left:*** *Agents that develop early on in a simulation are often small and vertical. This rendering shows a small colony of low growing plants. At this stage, competition starts to play a role; the plants cast shadows on their neighbours, reducing the amount of sunlight they receive. These plants grow in clusters, since their seeds (brown spheres) are located close to the ground, preventing them from spreading far away.* ***Right:*** *Their descendants are taller and more competitive, while remaining structurally similar.*

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 Figure 4 evolved from small plants, the first leaves in a simulation can develop on tall plants as well. Figure 5a) 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 6, left, shows an agent taken from such an environment.

3.3 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. Figure 5b) shows a rendering of a simulation during an early stage where the first leaves occur only once on their parent plants. In contrast, Figure 5c) shows agents developing multiple similar leaves per agent.

In simulation environments where multiple fertile areas exist, different genotypes will be dominant in different areas. They may however share a common

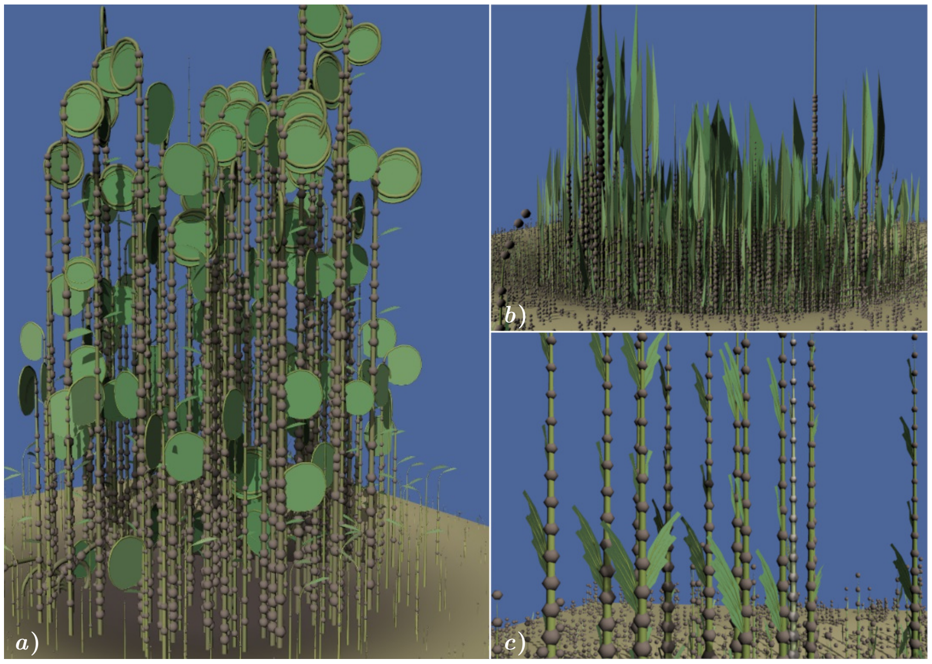


Fig. 5: ***a)*** *The leaves on these agents have developed early on during a simulation on agents with phenotypes similar to those in Figure 2c). The leaves are curled up unnecessarily, and the number of seeds produced on the stalk is inefficiently high.* ***b)*** *The first leaves have developed in this simulation. They are narrow and segmented, and they do not repeat themselves within the structure.* ***c)*** *Similar to b), but instead of developing a single leaf per plant, stalks contain multiple similar leaves instead.*

successful ancestor that managed to spread its seeds over the infertile barriers dividing the environment. Figure 7 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 second agent from the left has concentrated its leaf mass as high as possible, likely to prevent occlusion. The last agent in the row keeps increasing its leaf sizes as they develop: older leaves have bigger surface area.

Because of the stability criterion described in Appendix 2, agents only grow tall when required, and if they do, their structure will form mostly directly above their center of gravity. Figure 6, right, shows how an agent with a large complex structure managed to keep its eccentricity low.

3.4 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 8

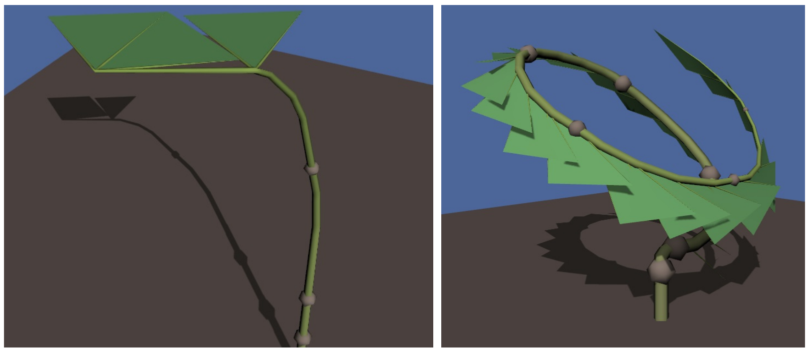


Fig. 6: ***Left:*** *This agent has evolved in an environment where light only shines directly down. It is taken from a simulation with many competing agents, so the leaf is placed on a high stalk to prevent occlusion.* ***Right:*** *An agent with a lot of branch and leaf material. Note that the plant structure is inclined such that the center of gravity is approximately above the root, reducing the eccentricity demonstrated in Figure 12, right.*

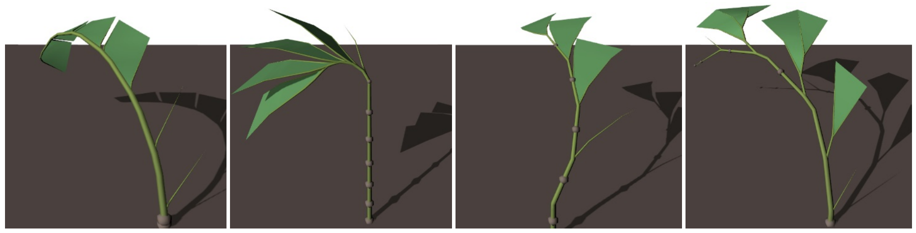


Fig. 7: *Four different agents with a common ancestor that existed* 1000 *generations earlier. All agents share the same structural strategy; new leaves form at the root, starting as a branch and growing into a leaf after multiple iterations. Various specializations have emerged.*

shows an agent that has developed a similar strategy to the lily of the valley; a vertical stem contains the reproductive systems (berries), while broad leaves emerging from the same root spiral around the central axis.

Figure 9 shows a group of agents that have developed thin fanning leaves that tend to align on the same plane, a strategy also seen in palm leaves.

Figure 10 shows a close up rendering of evolved leaves compared to real leaves. The leaf efficiency factor described in Appendix 2 causes well nerved leaves to yield higher viability than unsupported large leaf areas, resulting in well structured leaves similar to the ones found in nature.

4 Discussion and future work

While the goals (as outlined in Section 1) of our method were met, we now discuss several of its features and potential future improvements to our tool [27].

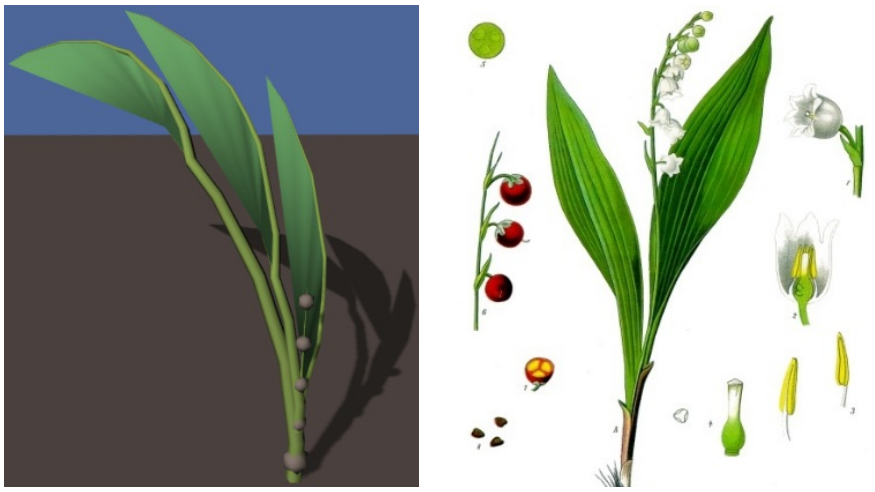


Fig. 8: *An illustration of a real "lily of the valley" flower on the right [14] compared to an evolved agent.*

The viability function in this simulation method remains constant. 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. With such a system, agents could be incentivized to produce robust strategies that remain successful when shared conditions change rapidly. An example in our context would be a global drought; terrain fertility could drop globally in a short time, and restore itself afterwards.

The implementation does not model growth over time. Agents develop completely before being compared to each other, which prevents strategies related to growth speed and life span from being relevant.

No *crossover* takes place in our reproductive model; this is a process whereby agents are not merely mutated clones of their parents, but rather a mutated mix of two parents [11]. Crossover could be implemented in this simulation by 'pollinating' each seed before reproduction. This can be simulated by finding a nearby similar agent and mixing these two systems through crossover.

In realistic environments, selective pressure applies to reproductive organs as well. By rating seeds by attractiveness to pollinators (e.g. benefiting seeds surrounded by bright colours), agents will be incentivized to develop flowers and compete on a visual level.

Sunlight exposure can be measured in many different ways [5, 26, 2]; our model treats all leaf surfaces as if they are equal, while in reality leaves differ in opacity, colour, reflectivity and other properties. The lighting model can be extended to allow different leaf types to develop.

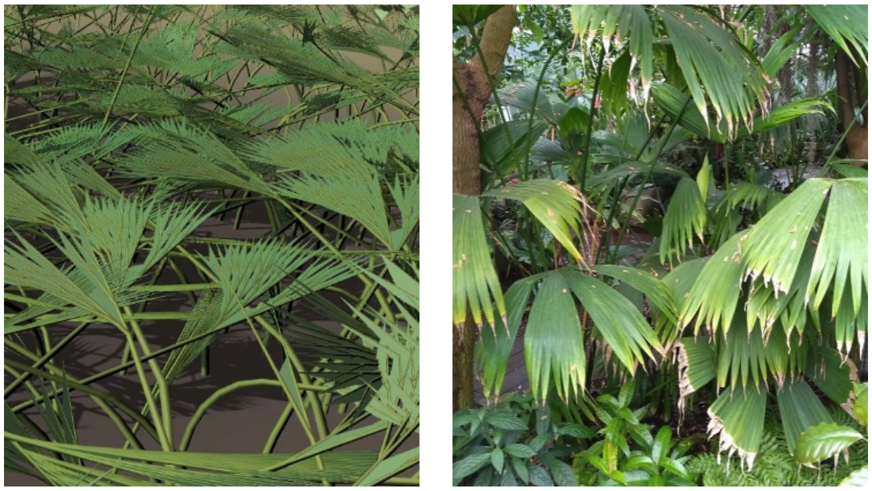


Fig. 9: *Palm leaves photographed in a greenhouse of the Oxford botanic garden on the right compared to a population of evolved agents on the left.*

5 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 (see Section 3.4). Competition plays a critical role in the development of strategies; agent fitness is defined relative to the surrounding agents by calculating leaf occlusion.

The evolutionary process itself displays realistic biological characteristics as well, e.g. divergent evolution (see Figure 7) and competition (see Section 3.2). The heterogeneous environment results in multiple different "species" of agents competing with one another. Each agent species performs optimally within a range of environmental properties.

One of the aims of our method is to allow the temporary survival of sub-par agents to allow for different agents to evolve without immediately culling low viability agents. viability plays a role in selection, but reproductive systems, terrain fertility and vegetation density play critical roles as well; Figure 2 a)–c) shows agents with zero viability competing on other grounds.

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

Acknowledgements: This paper is based on the first author's BSc thesis at the University of Groningen.

Appendix 1: Syntax

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

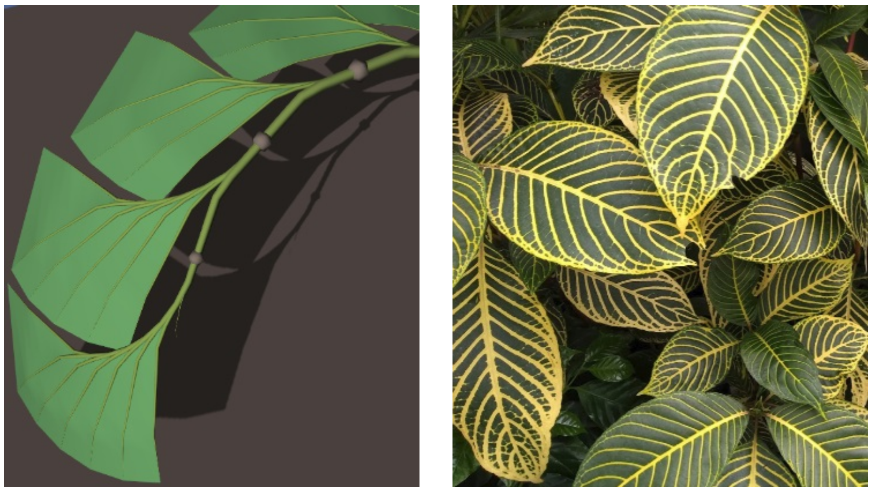


Fig. 10: *Clearly nerved leaves photographed in the Oxford botanic garden (**right**) juxtaposed with evolved leaf nerves (**left**). Note that both leaf structures separate their leaf surfaces at regular intervals, and that older leaves have more nerves.*

All production rules take the form $x \to y$, where both x and y are strings of symbols and y can be empty. 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 only increments.

To illustrate this process, consider a very simple L-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

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

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 $r_2 = A \to AB$. In this case, multiple end results are possible.

For every A encountered, both r_1 and r_2 are applicable.

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

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 *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. 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 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 at a closing bracket the turtle jumps back to the position it was at when the corresponding opening bracket was encountered.

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; if many branches exist in the leaf structure, many different leaf shapes can be modelled.

Figure 11 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. The shape in Figure 11 can be constructed from a string using the above algorithm. The sentence producing it is: `<A[++B[++C-C]-B-B]-A-A]`, where

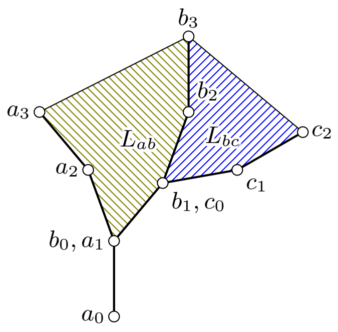
- `-` rotates the turtle orientation $\frac{\pi}{9}$ radians to the left,
- `+` rotates $\frac{\pi}{9}$ radians to the right,
- `[...]` denotes a branch, and
- `<...]` denotes a leaf.

The initial direction is upwards. At `A`, `B` or `C`, the turtle moves forwards for a fixed distance. 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.

Alphabet

The experiments performed according to the method in Appendix 2 take place in a three-dimensional environment, but the syntax is almost equal to the one used to model the shape in Figure 11; the only difference is the use of four more rotation symbols and a seed symbol. Figure 11, right, shows all symbols that may arise in the simulation together with their effects on the turtle.

The chosen alphabet contains as few symbols as possible to ensure that properties evolve because they are beneficial to agents, and not because we prefer agents to develop in that way (see Section 1). To test whether the alphabet is



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

Fig. 11: ***Left:*** *Leaf shape modelling.* ***Right:*** *The alphabet of possible symbols and related turtle actions.*

expressive enough, an extension to the alphabet in Figure 11 is implemented with an extra symbol, which will produce a single triangle shaped leaf. Using this symbol can show whether the simplicity of the standard alphabet prevents leaf features from developing. The results of this extension are detailed at the end of Appendix 2.

Appendix 2: Method

The simulation must be set up before experiments; the parameters that are chosen in this phase influence the course of agent development. The prerequisites of a simulation are categorized in three distinct modules: an environment, which contains the initial agents; a viability function used to rate agents' phenotypes; and a mutator containing the mutation function and its parameters.

Initializing the simulation environment

The simulation takes place on a surface 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.

An environment is defined as a rectangular terrain which may be sampled for fertility at any point. A terrain $t(x, y)$ yields a fertility factor in the range $[0, 1]$, where a higher value of t denotes higher fertility. More fertile terrain is lowered to visually resemble 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 higher fertility that cannot migrate through scarce regions anymore and thereby speciate. Figure 1 shows a possible environment, where 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. n relates to iteration i in a way that allows for an exponential increase in plant mass. 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 3, simulations were at all times seeded with agents containing an L-system without any rules and an axiom containing a single seed symbol. 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.

Assigning viability

An agent's viability determines the chance its offspring will make it to the next generation. The viability function in effect determines the course 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 too; small grasses may be efficient at first, but they stand little chance of competing against overshadowing ferns.

The viability 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 occluded leaves or leaves that do not align 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 viability score.

The *stability.* Stable structures are better. Tall plants will get a lower stability score, and will have to receive benefits in other aspects.

The *efficiency.* A very complex genotype (e.g. a system with many unused or duplicate production rules) requires more energy and lowers viability. 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 large amount of energy and is thus not a viable strategy in a competitive scenario.

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

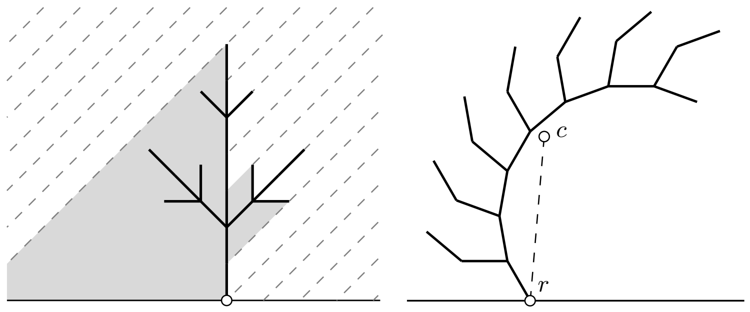


Fig. 12: ***Left:*** *Sunlight is projected onto an agent from the top right, all lines in this system are treated as leaf surface. The dotted lines represent light rays. A shadow is cast by the plant, occluding its branches on the left side of its vertical axis entirely.* ***Right:*** *A two-dimensional agent with root r and center of mass c.*

Rating sunlight exposure 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 the day, and evaluating how many rays of sunlight land on each agents' leaves. Figure 12, left, shows a schematic representation of sunlight projection in a two-dimensional environment.

The origin of light sources in a simulation strongly influences the phenotypes that will evolve; when light shines predominantly straight down (as would be the case in a ravine), flat horizontally aligned leaves will be preferable to other angles. The 2D algorithm illustrated in Figure 12, left, only registers light exposure to material directly hit by the sun rays. This is not realistic because real leaves are not fully opaque, and because light rays tend to bounce off the surfaces they hit, illuminating things that are initially occluded.

To account for these effects, leaf material has a certain *opacity factor*. This property determines how many rays will hit the surface, and how many will pass through. If the opacity 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 viability 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 11). The *leaf efficiency factor e* of an agent is therefore defined as:

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

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 viability; this means the viability from light exposure per surface area on a leaf is reduced for larger leaves.

The viability of sunlight exposure of an agent is set as

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

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 opacity (or the average surface of all sunlight projections if multiple are rendered). 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 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 small plants need very little. The sunlight exposure viability is multiplied by efficiency factor e.

Rating stability 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 12, right, shows an agent phenotype with its center of mass visualized with respect to its root.

Using root r and center of mass c, the agent stability viability 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 viability 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 negligible instability penalties, and can thus develop different growth strategies to obtain high viability scores.

Rating efficiency Efficiency captures the effectiveness of a genotype and the cost of its reproductive faculties. The effect of this viability score is therefore twofold: it penalizes overly complex L-systems, and it penalizes excessive seed production as well. The following formula captures these functions:

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

with n_{r} the number of production rules of the L-system, n_{s} is the number of seeds that exist 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.

Negative viability The final viability of an agent is $u = u_\mathrm{l} \cdot u_\mathrm{s} \cdot u_\mathrm{e}$. Agents with negative viability cannot be produced by this formula. There are however two exceptions for which the viability 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 early in 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 with many seeds. Growing such structures is not penalized by a poor sunlight exposure viability as it normally would, because this is zero for all leafless agents. To prevent unrealistic structures that support themselves without any photosynthesis, negative viability is assigned to leafless agents over a certain size.

Reproduction

Agents with a non-negative viability may produce seeds to reproduce themselves in the next generation. When all agents have been rated, their seeds are dispersed in an empty copy of the simulation environment to produce the next generation. Figure 13, left, 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.

After seeds have been dispersed and positioned in the environment, they are ranked by their parent agents' viability. Seeds with equal viability 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.

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, similar to the approach of [1]. Figure 13, right, 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), and every cell of this grid contains the number of agents overlapping that cell.

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.

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 thus determines how dense vegetation is allowed to grow, and by extension how competitive agents need to be

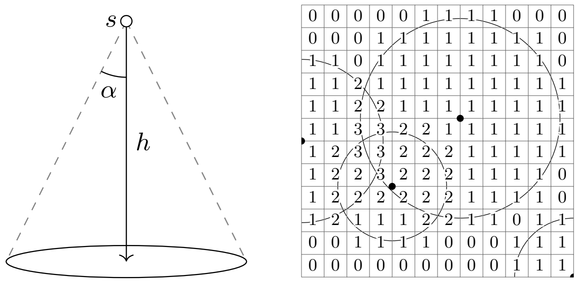


Fig. 13: ***Left:*** *The seed s may fall anywhere within the base of the cone.* ***Right:*** *A part of an agent density map. Four agents increment the cells they overlap.*

with respect to their neighbours. Simulations with a low density threshold barely experience competition, and thus produce lower growing plants; see Section 3.

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 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. Moreover, 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 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 14 shows all possible mutations on sentences using the syntax outlined in Figure 11. 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.

Operation (1) in Figure 14 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 L-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 Figure 11. Figure 15 shows an L-system and its corresponding E and N.

$$
\begin{align}
\text{Add symbol} &= \texttt{A}\underline{\texttt{B}}\texttt{[C]<D]} \rightarrow \texttt{A}x\texttt{B[C]<D]} \tag{1}\\
&\phantom{= \texttt{AB[C]<D]}} \rightarrow \texttt{AB}x\texttt{[C]<D]} \nonumber\\
\text{Remove symbol} &= \texttt{A}\underline{\texttt{B}}\texttt{[C]<D]} \rightarrow \texttt{A[C]<D]} \tag{2}\\
\text{Add branch} &= \texttt{A}\underline{\texttt{B}}\texttt{[C]<D]} \rightarrow \texttt{A[B][C]<D]} \tag{3}\\
\text{Remove branch} &= \texttt{AB}\underline{\texttt{[}}\texttt{C]<D]} \rightarrow \texttt{ABC<D]} \tag{4}\\
\text{Add leaf} &= \texttt{A}\underline{\texttt{B}}\texttt{[C]<D]} \rightarrow \texttt{A<B][C]<D]} \tag{5}\\
\text{Remove leaf} &= \texttt{AB[C]}\underline{\texttt{<}}\texttt{D]} \rightarrow \texttt{AB[C]D} \tag{6}
\end{align}
$$

Fig. 14: *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 in the sentence; mutation operations are applied at that location.*

$$
\begin{aligned}
a &= \texttt{B[+A][-A]BA}\\
r_1 &= \texttt{A} \rightarrow \texttt{B[+A][-A]BA}\\
r_2 &= \texttt{B} \rightarrow \texttt{BB}\\
E &= [\texttt{B}, \texttt{B}, \texttt{B}, \texttt{B}, \texttt{B}, \texttt{B}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{A}, \texttt{+}, \texttt{+}, \texttt{-}, \texttt{-}]\\
N &= [\texttt{A}\ldots\texttt{Z}] \uplus [\texttt{a}\ldots\texttt{z}] \uplus [\texttt{*}] \uplus R
\end{aligned}
$$

Fig. 15: *Multisets E and N of an L-system with axiom a and production rules r_1 and r_2. R is the multiset containing all rotation symbols defined in Figure 11.*

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 L-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 15 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 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.

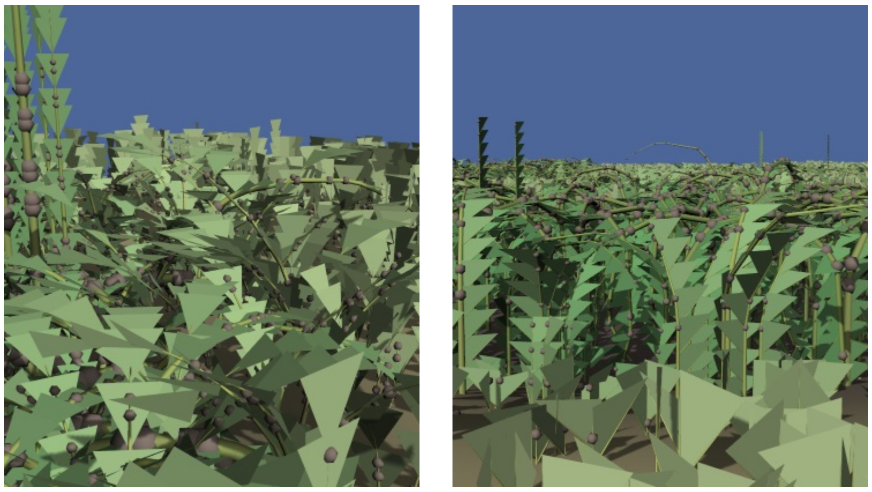


Fig. 16: ***Left:*** *A simulation at* $g = 40$ *using a default leaf symbol to model triangular leaves.* ***Right:*** *A simulation at* $g = 800$ *using the normal leaf modelling syntax from Appendix 1 produces very similar agents to the alternative method, although it takes more generations to evolve them.*

Default leaf shapes

The syntax described in Appendix 1 and Figure 11 uses a relatively complex method to define leaves. The process of modelling "good" leaves is arguably more complex than modelling the supporting structures.

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 (see Appendix 1); 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 16 shows a rendering of a simulation using this alternative method (left) compared to a simulation using the standard method (right). When default leaf shapes are used, agents with leaves evolve very quickly. Given enough time however, a simulation using the default 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.

Appendix 3: Implementation

The *LGen* software [27] built to perform the experiments using the method described in Appendix 2 was implemented in the *C++11* programming language and targets both Windows and Linux platforms. Figure 17 shows the different modules that make up the software. *LGen* performs, stores and loads experiments and configurations. *LParse* parses L-systems according to the syntax set

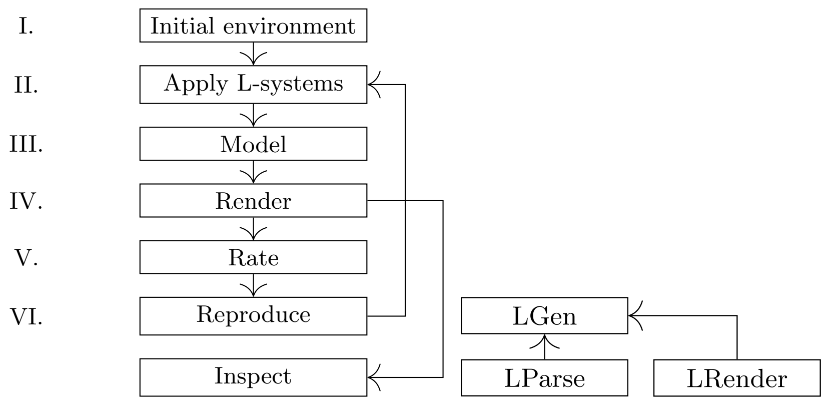


Fig. 17: ***Left:*** *Stages of a simulation.* ***Right:*** *Structure of the LGen software.*

forth in Appendix 1, generating the strings of symbols that are translated into agent phenotypes. *LRender* models and renders populated 3D simulation environments using *OpenGL* according to the algorithm described in Appendix 2, and provides information about these renderings (e.g. sunlight exposure ratios, structure size, and seed locations). *LGen* and *LParse* are both used by *LGen*, but do not require or reference each other.

Stages

The algorithm is divided into several stages per generation. Figure 17 shows the stages, in order of execution, that are required to simulate a generation.

Stage I initializes the environment, including for example its terrain function and seed dispersal angle. 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.

In Stage III, a 3D model of the environment and all its agents is created by applying the modelling rules detailed in Appendix 1. Again, this stage uses multithreading to speed up the modelling process. Summarizing information about agent models (size, center of gravity, etc.) 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, which assigns viability to every agent in the simulation. The viability function takes information gathered by Stages III and IV as well as the L-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 viability assigned to their parents by Stage V, and disperses the seeds. New agents are created from these seeds after mutation. 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 agents can be inspected by querying their L-systems and viability scores. 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.

Randomization

Randomization is used at several points in the simulation: stochastic L-systems may need to choose a random production rule from a list; the initial direction of the turtle when modelling an agent is upwards, but the rotation around the vertical axis is randomized; when dispersing seeds, randomization is used to displace them; mutating L-systems make heavy use of it.

The implementation uses *Mersenne twisters* [20] 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 emerge. 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.

Leaf area projections

To implement the sunlight exposure algorithm, the 3D scene produced in Stage III (see Figure 17) is rendered using an orthographic projection. Figure 18 shows two renderings; on the left, the modelled scene is rendered for viewing purposes using a perspective projection. On the right, the parallel projection is used to detect sunlight exposure in the same scene. This ensures that the produced pixels can be seen as hits by the parallel rays of sunlight in Figure 12, left.

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 opacity 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.

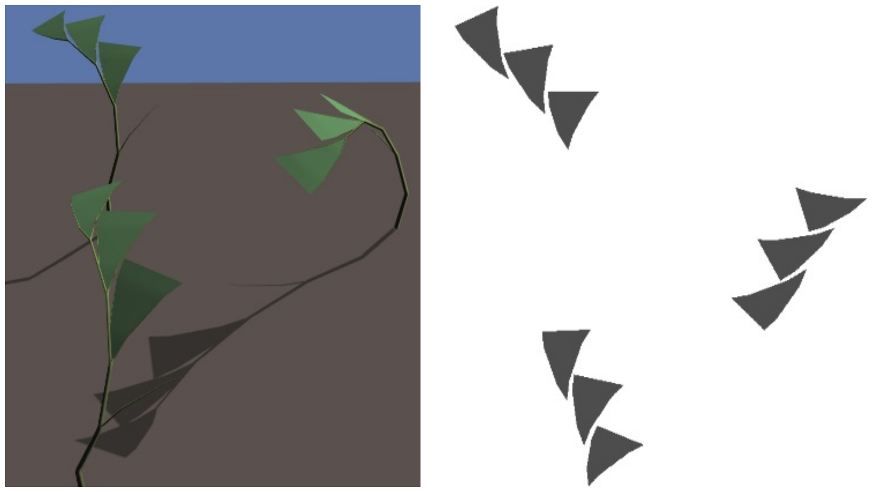


Fig. 18: ***Left:*** *A 3D rendering of three agents.* ***Right:*** *The corresponding exposure projection taken from approximately the same angle. Note that the phenotype has evolved such that the leaves do not occlude each other, while still being compact.*

References

1. Alsweis, M., Deussen, O.: Modeling and visualization of symmetric and asymmetric plant competition. In: Proceedings of the First Eurographics Conference on Natural Phenomena. pp. 83–88. NPH'05, Eurographics Association (2005)
2. Beyer, R., Etard, O., Cournède, P.H., Laurent-Gengoux, P.: Modeling spatial competition for light in plant populations with the porous medium equation. Journal of Mathematical Biology **70**, 533–547 (2015)
3. Bornhofen, S., Lattaud, C.: Competition and evolution in virtual plant communities: a new modeling approach. Natural Computing **8**(2), 349–385 (Jun 2009)
4. Burt, T.: Interactive Evolution by Duplication and Diversification of L-systems. Master's thesis, University of Calgary (2013)
5. Chelle, M., Andrieu, B.: Modelling the light environment of virtual crop canopies. In: Functional-Structural Plant Modelling in Crop Production. vol. 22, pp. 75–89. Springer (2007)
6. Cournède, P., d. Reffye, P.: A generalized Poisson model to estimate inter-plant competition for light. In: 2006 2nd Int. Symp. on Plant Growth Modeling and Applications. pp. 11–15 (2006)
7. Cournède, P.H., Mathieu, A., Houllier, F., Barthélémy, D., de Reffye, P.: Computing competition for light in the GREENLAB model of plant growth: A contribution to the study of the effects of density on resource acquisition and architectural development. Annals of Botany **101**(8), 1207–1219 (2007)
8. Ech, R., Prusinkiewicz, P.: Visual models of plants interacting with their environment. Computer Graphics (SIGGRAPH 1996) (1996)
9. Fitch, B.G., Parslow, P., Lundqvist, K.Ø.: Evolving complete L-systems: Using genetic algorithms for the generation of realistic plants. In: Lewis, P.R., Headleand, C.J., Battle, S., Ritsos, P.D. (eds.) Artificial Life and Intelligent Agents. pp. 16–23. Springer (2018)

10. Fourcaud, T., Zhang, X., Stokes, A., Lambers, H., Körner, C.: Plant Growth Modelling and Applications: The Increasing Importance of Plant Architecture in Growth Models. Annals of Botany **101**(8), 1053–1063 (04 2008)
11. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
12. Hua, J., Kang, M.: Functional tree models reacting to the environment. In: ACM SIGGRAPH 2011 Posters. SIGGRAPH 11 (2011)
13. Jacob, C.: Evolving evolution programs: Genetic programming and L-systems. In: Proc. of the 1st Annual Conf. on Genetic Programming. pp. 107–115. MIT Press (1996)
14. Köhler, F.E.: Lily of the valley illustration. In: Hermann Adolph Köhler's Medicinal Plants, edited by Gustav Pabst (1887)
15. Lindenmayer, A.: Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. Journal of Theoretical Biology **18**(3), 300–315 (1968)
16. Lintermann, B., Deussen, O.: A modelling method and user interface for creating plants. Computer Graphics Forum **17**(1), 73–82 (1998)
17. MacKenzie, C., Prusinkiewicz, P.: Artificial evolution of plant forms. In: Proceedings of the Fifth Annual Western Computer Graphics Symposium. pp. 1–9 (1993)
18. Makowski, M., Hädrich, T., Scheffczyk, J., Michels, D.L., Pirk, S., Pałubicki, W.: Synthetic silviculture: Multi-scale modeling of plant ecosystems. ACM Trans. Graph. **38**(4) (Jul 2019)
19. Mathieu, A., Cournède, P., Letort, V., Barthélémy, D., de Reffye, P.: A dynamic model of plant growth with interactions between development and functional mechanisms to study plant structural plasticity related to trophic competition. Annals of Botany **103** (2009)
20. Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (Jan 1998)
21. Ochoa, G.: On genetic algorithms and Lindenmayer systems. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.P. (eds.) Parallel Problem Solving from Nature. pp. 335–344. Springer (1998)
22. Palubicki, W., Horel, K., Longay, S., Runions, A., Lane, B., Měch, R., Prusinkiewicz, P.: Self-organizing tree models for image synthesis. ACM Trans. Graph. **28**(3) (2009)
23. Pirk, S., Stava, O., Kratt, J., Said, M.A.M., Neubert, B., Měch, R., Beneš, B., Deussen, O.: Plastic trees: Interactive self-adapting botanical tree models. ACM Trans. Graph. **31**(4) (2012)
24. Prusinkiewicz, P.: A look at the visual modeling of plants using L-systems. In: Hofestädt, R., Lengauer, T., Löffler, M., Schomburg, D. (eds.) Bioinformatics. pp. 11–29. Springer (1997)
25. Risi, S., Stoy, K., Faíña, A., Veenstra, F.: Generating artificial plant morphologies for function and aesthetics through evolving L-systems. The 2019 Conf. on Artificial Life (28), 692–699 (2016)
26. Sarlikioti, V., de Visser, P.H.B., Marcelis, L.F.M.: Exploring the spatial distribution of light interception and photosynthesis of canopies by means of a functional-structural plant model. Annals of Botany **107**(5), 875–883 (2011)
27. Talle, J.J.V.: LGen software source code, https://github.com/jobtalle/LGen