Evolution and Growth of Virtual Plants

Marc Ebner

Universität Würzburg, Lehrstuhl für Informatik II
Am Hubland, 97074 Würzburg, Germany
ebner@informatik.uni-wuerzburg.de
http://www2.informatik.uni-wuerzburg.de/staff/ebner/welcome.html

Abstract. According to the Red Queen hypothesis, an evolving population may be improving some trait, even though its fitness remains constant. We have created such a scenario with a population of coevolving plants. Plants are modeled using Lindenmayer systems and rendered with OpenGL. The plants consist of branches and leaves. Their reproductive success depends on their ability to catch sunlight as well as their structural complexity. All plants are evaluated inside the same environment, which means that one plant is able to cover other plants leaves. Leaves which are placed in the shadow of other plants do not catch any sunlight. The shape of the plant also determines the area where offspring can be placed. Offspring can only be placed in the vicinity of a plant. A number of experiments were performed in different environments. The Red Queen effect was seen in all cases.

1 Motivation

The fitness of coevolving species may remain at the same level over time even though each individual may be continually improving some specific trait. This has been called the Red Queen hypothesis [20], after the Red Queen chess piece in Lewis Carroll's "Through the Looking Glass". The queen has to keep running just to stay in the same place. The classic example of the Red Queen effect is the chase between cheetahs and gazelles. If the population of prey becomes faster, some of the predators are unable to catch their prey. The slow ones will not be able to reproduce. After some generations, mutations might lead to predators which are a little faster than their ancestors. These individuals will again be able to catch their prey easily and the status quo is maintained. In the course of evolution both populations will become better at achieving their task. We have visualized the Red Queen effect in a population of artificial plants.

Plants are usually represented as Lindenmayer systems or L-systems for short [19]. Prusinkiewicz and Lindenmayer [19] have shown how complex, photorealistically looking plants can be created from a relatively small number of rules. See Deussen et al. [3] for an introduction to the realistic modeling and rendering of plant ecosystems. Computer-simulated evolution of virtual plants was pioneered by Niklas [16]. He performed an adaptive walk through a space of branching patterns. Jacob [5–9] used a variant of genetic programming to evolve

context-free and context-sensitive L-Systems representing plants. He used a combination of the number of blossoms, the number of leaves and the volume of the plant as a fitness function. Two-dimensional plant morphologies were evolved by Ochoa [17]. Kokai et al. [11, 12] also evolved L-Systems. They tried to generate rules which, when executed and viewed, look identical to a given image. Mock [15] developed a system where the user could take the role of a virtual gardener selecting plants for reproduction. He also worked with an explicit fitness function that rewarded plants which are short but wide. Artificial models for natural plant evolution were developed by Kim [10].

In contrast to most previous work on artificial plant evolution we allow interactions between the plant and its environment. All plants are placed in the same environment and evaluated together. Plants need to catch as much virtual sunlight as possible in order to reproduce successfully. Because all plants are evaluated inside the same environment, one plant may shadow the leaves of other plants or it may even shadow its own lower leaves. If a plant reproduces twice into the next generation, it will have to compete with a copy of itself. We will see that after a short number of generations, maximum fitness starts to fluctuate around a constant level. This contrasts sharply with a more continuous evolution when plants are evaluated in isolation. However, even though fitness is no longer rising, plants are still evolving and adapting to their environment. At this point an evolutionary arms race [2] sets in. The plants need to grow higher and higher in an attempt to gain more sunlight.

2 Plant Representation

Plants are modeled as deterministic, context-free L-Systems. A L-System is basically a string rewrite system where the individual letters of a word are transformed according to a set of rules. The set of rules is applied to all letters in parallel. A L-System is defined by an alphabet V, a starting word ω and a set of rules P [19]. The starting word ω is a non-empty string $\omega \in V^+$. This word is transformed using the set of rules. The set of rules is defined as a subset of $V \times V^+$. Each rule consists of a predecessor a and a successor χ. Thus, a single rule can be written as $a \rightarrow \chi$. The letter a will be replaced by the string χ wherever it appears in the word. In case no successor is defined for a predecessor a, it is assumed that the rule $a \rightarrow a$ also belongs to the set of rules P. This rule leaves the letter a unchanged. By replacing all predecessors with their successors, a new word is derived from the original one. This process is repeated 5 times (for the experiments described in this paper). The final word is interpreted as a sequence of commands for a three-dimensional drawing device.

Our alphabet consists of the symbols:

$$V = \{\texttt{f}, \texttt{l}, \texttt{+}, \texttt{-}, \texttt{<}, \texttt{>}, \texttt{/}, \texttt{\textbackslash}, \texttt{[}, \texttt{]}, \texttt{A}, ..., \texttt{Z}\}. \quad (1)$$

The letter `f` produces a branch segment. It draws a cylinder and moves the drawing device forward by a distance equal to the length of the cylinder. The letter `l` produces a leaf. Here, the position of the drawing device does not change.

initial word: `f`
rules:

`f` → `f\\<l[Afl]\\\-f`
`A` → `B-\f`
`B` → `CA`
`C` → `>`

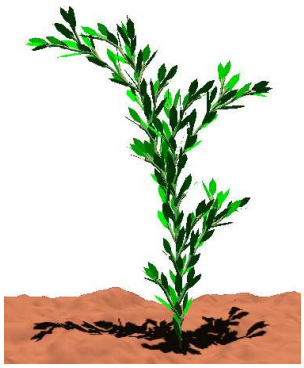

Fig. 1. Sample grammar of an evolved plant. The plant was evolved using coevolution on a flat landscape.

All leaves of the plant have the same shape and size. These are the building blocks which are used to build the plant. The symbols `+`, `-`, `<`, `>`, `/`, and `\` can be used to change the orientation of the drawing device. The orientation can by changed in discrete steps of 22.5 degrees in both directions around any of the three axis of the coordinate system. Thus, the sequence `f+f` creates a bent branch segment. Branching structures can be created with the symbols `[` and `]`. Whenever the symbol `[` is encountered during the interpretation of the drawing commands, the current state, i.e. the three-dimensional position and orientation, of the drawing device is pushed on a stack. Whenever the symbol `]` is encountered the topmost state is popped from the stack. This restores the position and orientation of the drawing device to a previously saved state. Thus, the sequence `f[-f]+f` creates a Y-shaped branching structure. Additional symbols `A` through `Z` can be used to define substructures. These symbols only play a role during development. When interpreting the final word, they cause no operation. A sample grammar of an evolved plant is shown in Figure 1.

3 Plant Evaluation

All individuals of a population are evaluated at the same time. Each individual has a specific position (x, y) and orientation α inside an rectangular evaluation area. Fitness is a function of the plant's ability to collect sunlight as well as the plant's structural complexity. We assume that the sun is positioned directly above the plant. The leaves of the plant may be used to collect virtual sunlight. Figure 2 shows a plant viewed from above. The plant was rendered with OpenGL using orthographic projection into a 512×512 image. Shading calculations were turned off. Cylinders were drawn in black and leaves in green. OpenGL uses a technique called Z-buffer for hidden surface removal. Thus, the number of green pixels in the image is a measure of the plant's ability to collect sunlight. Use of the Z-Buffer to estimate the amount of sunlight hitting a plant was proposed by Beneš [1].

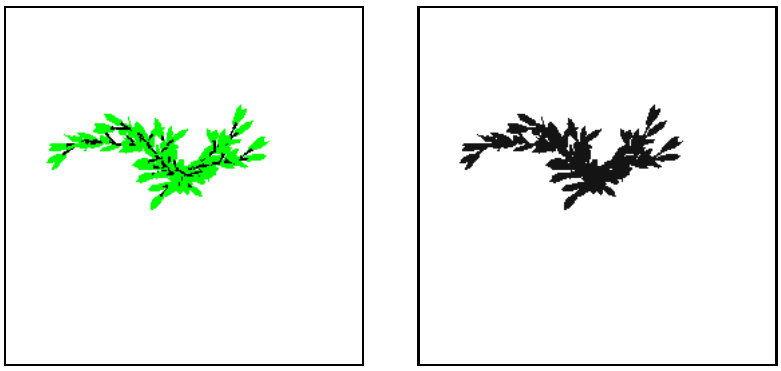

Fig. 2. The image on the left shows an image of the plant shown in Figure 1. The plant is viewed from above, and rendered using orthographic projection. Leaves are drawn with green color. The number of green pixels is used to calculate the plant's fitness. The image on the right shows the shadow of the plant. Offspring may grow anywhere on this footprint of the plant. Thus a plant's offspring are always placed in the parent's vicinity.

In a natural environment a plant faces competition from many sources. One prime source of evolution may be the presence of other individuals. We have modeled this by evaluating all individuals of the population at the same time. All plants are rendered in a single image using orthographic projection. Each plant is assigned a unique green color to draw the plant's leaves. Cylinders are again drawn in black. A sample population of 7 plants is shown in Figure 3. The two images on the left show the plant population and the image on the right shows the image which is used to determine the fitness values of the plants. If a plant grows very tall it may shadow other plants. Thus, if coevolution is used, there is an incentive for the plants to grow higher and higher. Note that this type of coevolution does not use two separate populations. We only work with a single population. All individuals of the population are evaluated in a single environment.

The amount of light received is only one part of the fitness calculation. The second part is due to the structural complexity of the plant. In our model a branch segment costs 1 point and a leaf costs 3 points. This basic cost is multiplied with a factor which depends on the distance to the root of the plant. Branches and leaves which are far away from the root are more costly than branches and leaves which are close to the root. The structural complexity of a plant is defined as

$$\text{complexity} = \sum_{b \in B} \text{cost}_{\text{branch}} \cdot \text{factor}^{\text{height}(b)} + \sum_{l \in L} \text{cost}_{\text{leaf}} \cdot \text{factor}^{\text{height}(l)} \qquad (2)$$

where B is the set of branch segments, L is the set of leaves and height is the number of branch segments between the current position and the root of the

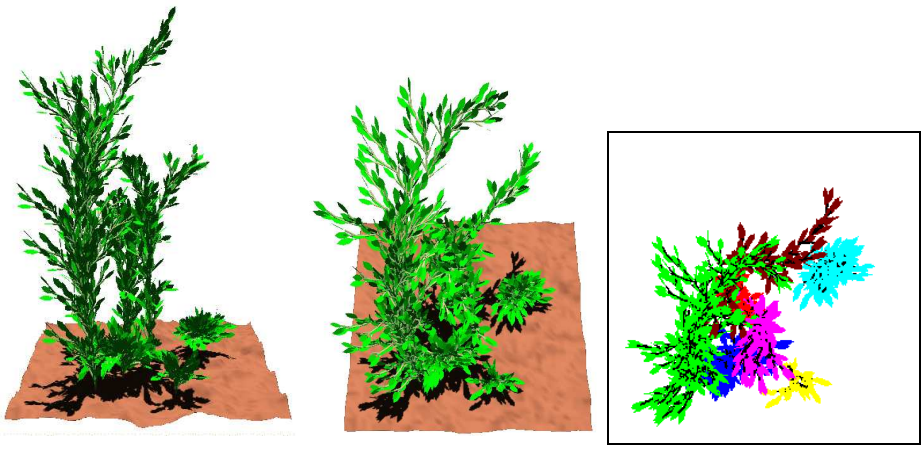

Fig. 3. The two images on the left show 7 different evolved plants. Large plants shadow smaller plants. The image on the right shows the image used to evaluate the plant's fitness values. Each plant has a unique color which is used to render the plant's leaves.

plant. The following parameters were used during the experiments: factor = 1.1, $\text{cost}_{\text{branch}} = 1$, and $\text{cost}_{\text{leaf}} = 3$. Fitness is calculated by subtracting the structural complexity from the amount of light received:

$$\text{fitness} = 10 \cdot \text{pixels} - \text{complexity} \tag{3}$$

where pixels is the number of pixels covered by the plant's leaves. The number of green pixels is weighted with a factor of 10 which was determined experimentally. If this value is too small, plants will not be able to grow very high as the number of green pixels per plant is fixed when distributed evenly among all plants. On the other hand, if this value is very large, plants have the potential to grow very high. Negative fitness values are not allowed. If this occurs, the fitness is set to zero.

4 Evolution of Artificial Plants

We use an evolutionary algorithm, a variant of genetic programming [13, 14] to automatically generate new populations of plants [4]. To create the individuals of the next generation we first chose a genetic operator at random. Depending on the type of genetic operator one or two individuals are selected from the original population. Parents are selected using tournament selection. This selection process does not depend on the spatial location of the plants. The following genetic operators were used: permutation, mutation, insertion, add-branch, deletion, one-point-crossover, sub-tree-crossover, delete branch, add-rule, delete-rule. For a description of these operators see Ebner et al. [4]. Offspring are basically

created by making small changes to the selected individual (mutation) or by combining segments of two selected individuals (crossover). Offspring are inserted into the next generation until the generation is filled. Whenever an offspring is created, we have to determine the position where it will grow. This position is determined by rendering the plant using OpenGL with orthographic projection. Cylinders and leaves are both drawn in black. A sample footprint of an evolved plant is shown in Figure 2. From this footprint, we randomly select a position to place the offspring. The orientation of the plant is chosen randomly. Thus, offspring always grow in the vicinity of their parents.

5 Experiments

We ran experiments on three different landscapes. The landscapes were generated using Perlin noise [18]. The first landscape is essentially flat with small height differences. Individuals are initially placed randomly within a small circular area in the center of the evaluation area. A population of 200 individuals with tournament selection and a tournament size of 7 was used. The first generation contains only individuals with the single rule $\texttt{f} \rightarrow \texttt{f}$. The starting word is `f`. Therefore, all individuals of the first generation have a fitness of zero. Each genetic operator was applied with a probability of 0.1 to generate an offspring. Results for the flat landscape are shown in Figure 4. The second experiment was carried out on a landscape with a large vertical slope. Results for this experiment are shown in Figure 5. The third experiment was carried out on a landscape where one fourth of the landscape was elevated. Results for this experiment are shown in Figure 6.

In all three cases, maximum fitness stops increasing after generation 50. However, the evolutionary process has not come to a halt yet. Plants are still adapting to their environment and changing their shape. Note, that we did not impose a maximum height for the plants. However, the number of green pixels per plant is fixed when distributed evenly among plants. Thus, this also limits the maximum height attainable. Maximum complexity can be no larger than the first term of the fitness function. Plants evolve to the point where no further increase in structural complexity is possible. Plants all over the environment converged to very similar looking plants. The best individuals of generation 250 are shown in Figure 7.

6 Conclusion

Due to the Red Queen effect, evolution may occur even if fitness seems to remain constant. We have shown this for a population of artificial plants. The plants adapt to their environment even though maximum fitness starts to fluctuate around a constant level. Plants were modeled using Lindenmayer systems and evaluated using OpenGL. They need to catch virtual sunlight in order to reproduce. OpenGL proved to be very effective to evaluate the amount of sunlight each plant receives. A similar mechanism was used to determine the position

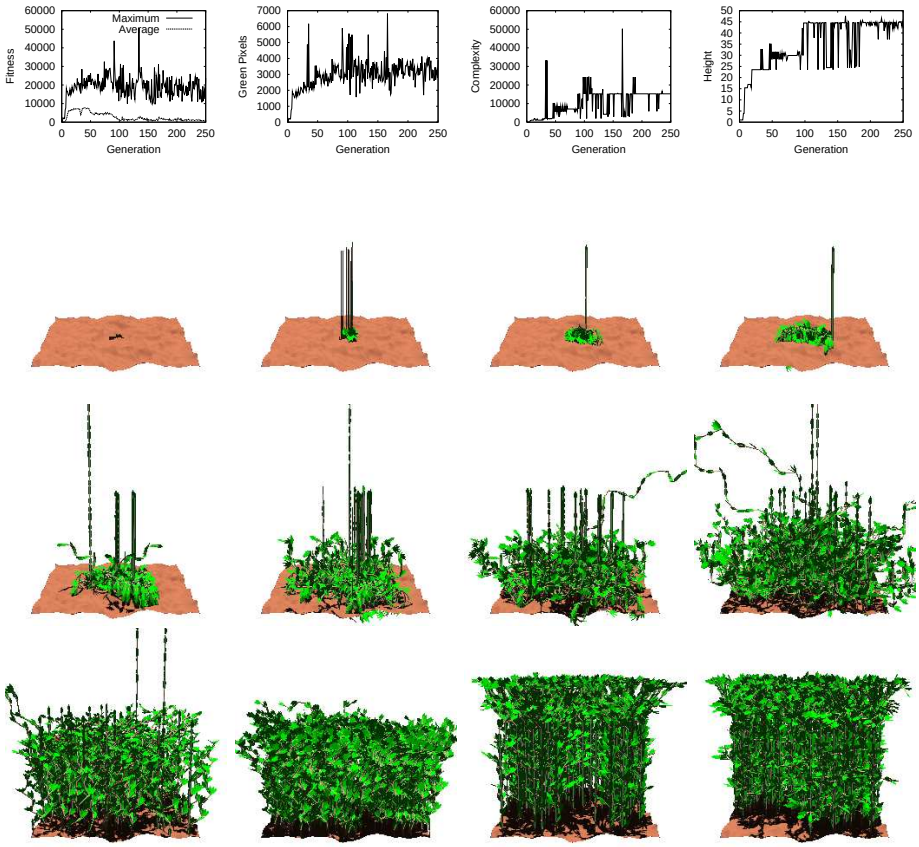

Fig. 4. Results for the experiment on a flat landscape. The first graph shows maximum and average fitness over time. The second graph shows the number of green pixels, the third graph shows plant complexity and the fourth graph shows the height of the best plant of each generation. The images below show the population at generation 0, 4, 6, 7, 8, 9, 10, 14, 32, 64, 128, and 250. The population spreads from a small area in the middle and eventually populates the whole area.

where offspring could be placed. Offspring were always placed in the vicinity of their parent. Three types of environments were used to visualize the Red Queen effect. An essentially flat landscape, a landscape with a large slope and a landscape with two height levels. In all three cases, instances of the plants quickly populated the environment. After all of the environment was populated, an arms race set in, which further shaped the plants. Plants need to outgrow their competitors in order to gain sunlight. This lead to the evolution of higher and higher plants.

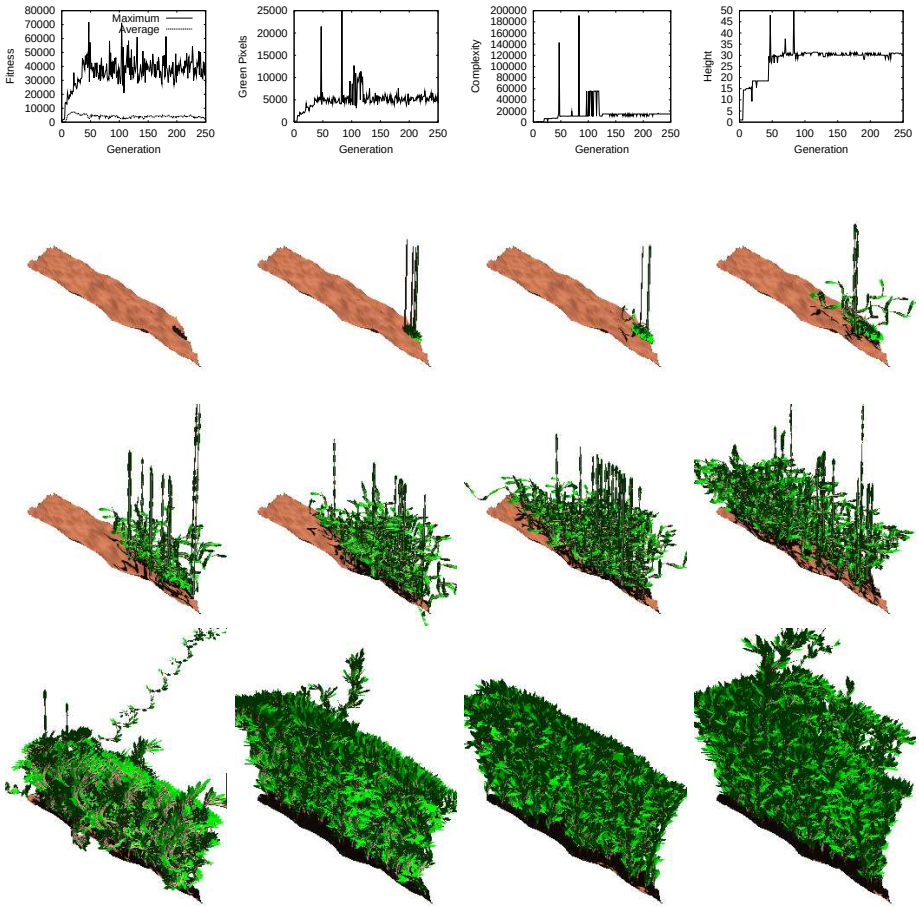

Fig. 5. Results for the experiment on a landscape with a large slope. The images below show the population at generation 0, 4, 6, 7, 8, 9, 10, 14, 32, 64, 128, and 250. The population spreads from a small area on the right side, climbs up the slope and eventually populates the whole area.

References

1. B. Beneš. An efficient estimation of light in simulation of plant development. In R. Boulic and G. Hegron, eds., *Computer Animation and Simulation 96*, pp. 153–165, Springer-Verlag, Berlin, 1996.
2. R. Dawkins and J. R. Krebs. Arms races between and within species. *Proc. R. Soc. Lond. B*, 205:489–511, 1979.
3. O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH '98 Conf. Proceedings, Comp. Graphics, Orlando, FL*, pp. 275–286. ACM Press, 1998.

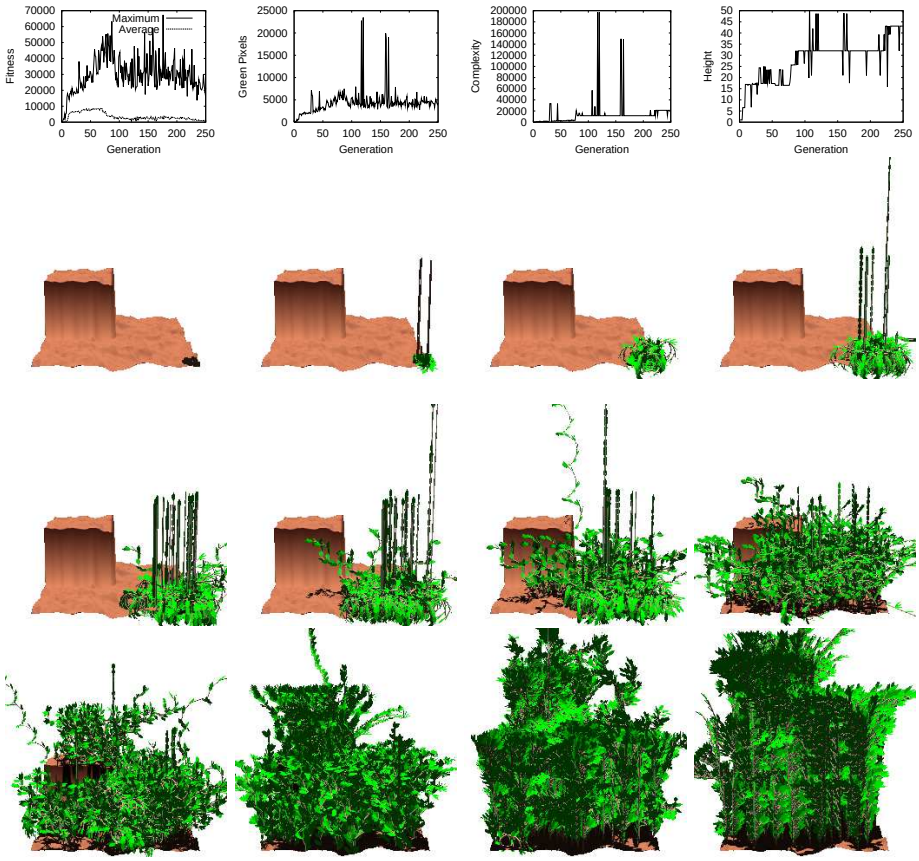

Fig. 6. Results for the experiment on a landscape with two different height levels. The images below show the population at generation 0, 4, 7, 8, 9, 10, 11, 15, 32, 64, 128, and 250. The population spreads from a small area in the lower right corner and eventually populates the whole area.

4. M. Ebner, A. Grigore, A. Heffner, and J. Albert. Coevolution produces an arms race among virtual plants. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi, eds., *Genetic Programming: Proc. of the Fifth Europ. Conf., EuroGP 2002, Kinsale, Ireland*, Springer-Verlag, Berlin, 2002.
5. C. Jacob. Genetic L-system programming. In Y. Davudor, H.-P. Schwefel, and R. Männer, eds., *Parallel Problem Solving from Nature – PPSN III. The Third Int. Conf. on Evolutionary Computation. Jerusalem, Israel*, pp. 334–343, Springer-Verlag, Berlin, 1994.
6. C. Jacob. Evolution programs evolved. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds., *Parallel Problem Solving from Nature – PPSN IV. The Fourth Int. Conf. on Evolutionary Computation. Berlin, Germany*, pp. 42–51, Springer-Verlag, Berlin, 1996.

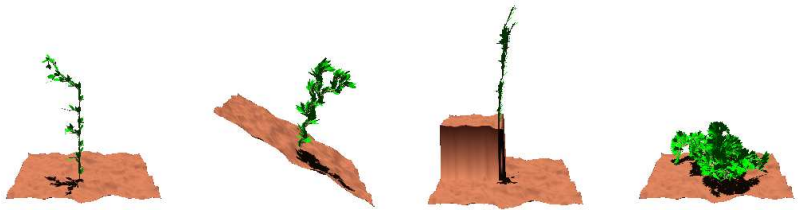

Fig. 7. Best individuals of experiments 1, 2, and 3. The fourth image shows a plant which was evolved by evaluating the individuals in isolation. This mode of evaluation produces a bush-shaped plant. It contrasts sharply with the plants evolved using coevolution.

7. C. Jacob. Evolving evolution programs: Genetic programming and L-systems. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, eds., *Proc. of the First Annual Conf. on Genetic Programming*, pp. 107–115, The MIT Press, Cambridge, MA, 1996.
8. C. Jacob. Evolution and coevolution of developmental programs. *Computer Physics Communications*, pp. 46–50, 1999.
9. C. Jacob. *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann Publishers, San Francisco, CA, 2001.
10. J. T. Kim. Lindevol: Artificial models for natural plant evolution. *Künstliche Intelligenz*, 1:26–32, 2000.
11. G. Kókai, Z. Tóth, and R. Ványi. Application of genetic algorithms with more populations for Lindenmayer systems. In E. Alpaydin and C. Fyfe, eds., *Int. ICSC Symposium on Engineering of Int. Systems EIS '98, University of La Laguna, Tenerife, Spain*, pp. 324–331, ICSC Academic Press, Canada/Switzerland, 1998.
12. G. Kókai, Z. Tóth, and R. Ványi. Evolving artificial trees described by parametric L-systems. In *Proc. of the 1999 IEEE Canadian Conf. on Electrical and Computer Engineering, Shaw Conference Center, Edmonton, Alberta, Canada*, pp. 1722–1727. IEEE Press, 1999.
13. J. R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, MA, 1992.
14. J. R. Koza. *Genetic Programming II. Automatic Discovery of Reusable Programs*. The MIT Press, Cambridge, MA, 1994.
15. K. J. Mock. Wildwood: The evolution of L-system plants for virtual environments. In *Int. Conf. on Evolutionary Computation, Anchorage, Alaska*, pp. 476–480, 1998.
16. K. J. Niklas. Computer-simulated plant evolution. *Scientific American*, 254(3):68–75, 1986.
17. G. Ochoa. On genetic algorithms and Lindenmayer systems. In *Parallel Problem Solving from Nature - PPSN V*, pp. 335–344, Springer-Verlag, Berlin, 1998.
18. K. Perlin. Noise, hypertexture, antialiasing and gesture. In D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, eds., *Texturing and Modeling: A Procedural Approach. 2nd Ed.*, pp. 209–274, AP Professional, Cambridge, 1998.
19. P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, New York, 1990.
20. L. Van Valen. A new evolutionary law. *Evolutionary Theory*, 1:1–30, July 1973.