Evolutionary Plantographics

Alon Gal
Gady Mahal
Moshe Sipper*
Department of Computer Science
Ben Gurion University
P.O. Box 653
Beer Sheva 84105
Israel
{gal_alon,mahalg,sipper}@cs.bgu.ac.il

Abstract This letter describes an evolutionary system for creating lifelike three-dimensional plants and flowers, our main goal being the facilitation of producing realistic plant imagery. With these two goals in mind—ease of generation and realism—we designed the plant genotype and the genotype-to-phenotype mapping. Diversity in our system comes about through two distinct processes—evolution and randomization—allowing the creation not only of single plants but of entire gardens and forests. Thus, we are able to readily produce natural-looking artificial scenes.

Keywords
Computer graphics, evolutionary algorithms, robot shaping, evolutionary plantographics

1 Evolutionary Algorithms + Computer Graphics

The marriage of evolution and computer graphics, ministered by Sims [9] over a decade ago, has blossomed into a small but flourishing endeavor, the idea of using evolutionary algorithms to create graphical images having proven viable. As in many other fields, evolution is quite a powerful computational metaphor [12].

Sims [9] used symbolic Lisp expressions to represent graphical images, where an expression is an equation (the genotype) that calculates a color for each pixel coordinate (x, y), thus producing an image (the phenotype). A population of such genotypes is randomly created, and their phenotypic images are shown to the user, who performs the (unnatural) selection according to her tastes. The genotypes are then combined and mutated—as in a standard genetic algorithm—creating ever lovelier images, adapted to the user's aesthetic sense. Selection is thus performed by the user upon the population of phenotypes, while genetic operators are applied to the genotypes.

A major accomplishment of Sims was the adoption of a suitable genomic representation, which—through the genotype-to-phenotype mapping—is well adapted to evolutionary experimentation. Sims [9] cited several other plant-generation algorithms (the best-known being L systems [7], with which Ochoa [6] evolved two-dimensional plants), though his brand—on which our work is based—still seems to us the most promising for evolutionary experimentation.

Sims later went on to evolve three-dimensional virtual creatures [10, 11], followed by Komosinski and Ulatowski, who introduced the Framsticks model [3]. These works are concerned not solely with graphics, but also (and perhaps more so) with the question of morphological evolution and robot-body development. More recently, Lipson and Pollack [4] evolved virtual robots, of which individuals possessing the highest fitness were implemented in the real world by an automated construction. See Taylor and Massey [13] for a recent review of developments in the field of evolution of robotic morphology (another good source is Hornby and Pollack [2]). As we are dealing with flora rather than fauna, this line of research is less relevant than that described in the previous paragraph.

∗ Corresponding author.

This letter describes an evolutionary system for creating lifelike three-dimensional plants, our main goal being the facilitation of producing realistic plant imagery (hence, evolutionary plantographics). The system is fully detailed herein, and can be obtained at moshesipper.com/plantmaker. The next section describes the phenotypic selection performed by the user, followed by Section 3 describing the genotypic representation and operators. Section 4 presents our results in the form of a gallery. We end with a discussion in Section 5.

2 Down the Garden Phenotype

The user creates the initial plant population by choosing (or drawing) values for every plant's genome (the structure of which will be discussed in the next section). The genome can be manipulated in two ways: (1) indirectly (the software enables the user to examine a plant's genome through an interface allowing the viewing and modification of plant properties), and (2) directly (the genome is ultimately a string of numbers stored in a file, which the user can freely access and modify). Initialization can also be done at random, or by choosing previously evolved plants, should no direct genomic intervention be desired (though our experience has shown that some genotypic tinkering usually produces a more pleasing initial population). Figure 1 shows a sample plant population.

The process of evolving plants involves the user's selecting phenotypes to her liking, from the assortment presented on the screen. The next generation of images is then created through crossover and mutation: The user can choose two or more plants to cross over (Figure 2), with the added possibility of determining the dominant plant during mating (the more dominant a parent, the more genes it will contribute to the offspring). The user can also choose a single plant to undergo mutation (Figure 3), the mutation probability being user-controllable. Every plant created through these two operators is added to the population, and immediately made available for the user's future use. One can also save a plant's genome, thus allowing its display at some future time.

The main program window shows the nine dominant plants. Following an evolutionary step (recombination or mutation) the system replaces the least dominant plant(s) with the offspring. Note that population size is essentially unlimited, the program maintains in volatile memory all evolved plants of the current session; thus plants removed may still be accessed by the user.

3 Pulling Down the Genes

The genomic makeup of a plant (along with the genotype-to-phenotype mapping), described in this section, has been designed for both *evolvability* [5]—the *possibility* of producing a striking array of variegated plants—and *realism.* Finding a viable, evolvable representation is the designer's hardest task in systems like ours (with no a priori "natural" problem representation); one needs to strike a balance between oversimplicity and overcomplexity: The genome must give rise to a phenotypic space that is not too simple and therefore boring, yet not so complex that pleasing forms are very rare and hard to evolve.

A plant is composed of branches and leaves arranged three-dimensionally. For simplicity, both branches and leaves are represented by the same object, called a *branch*, consisting of connected cylindrical and conic shapes. Depending upon the various branch parameters (described below), the rendered object can assume the appearance of a branch or a leaf.

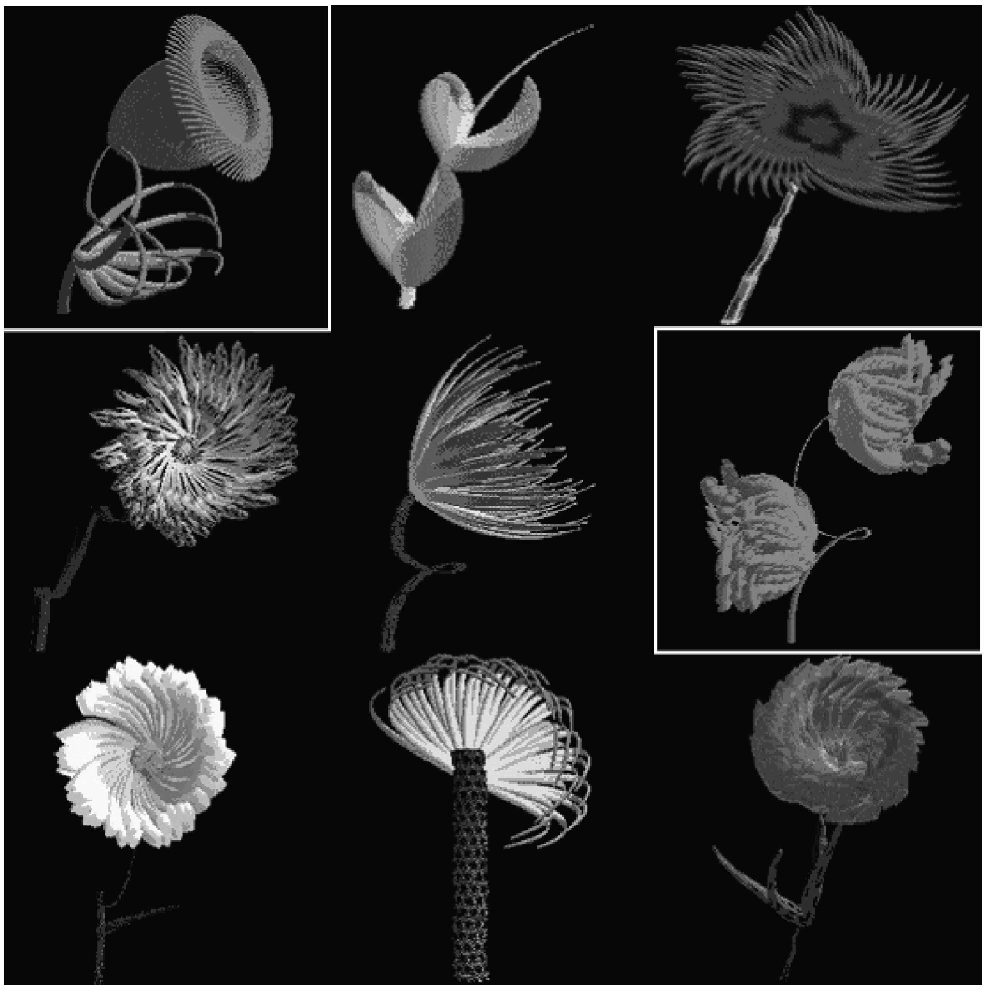


Figure 1. A sample plant population. The user can select plants to cross over and mutate, thus adding new offspring to the growing population.

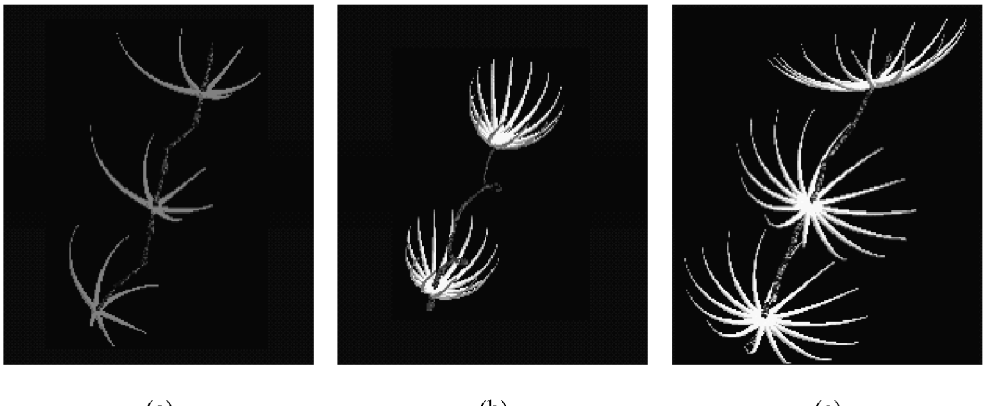


Figure 2. Crossover: Example of two parents and their offspring. (a) Parent *A*. (b) Parent *B*. (c) Offspring of *A* and *B*.

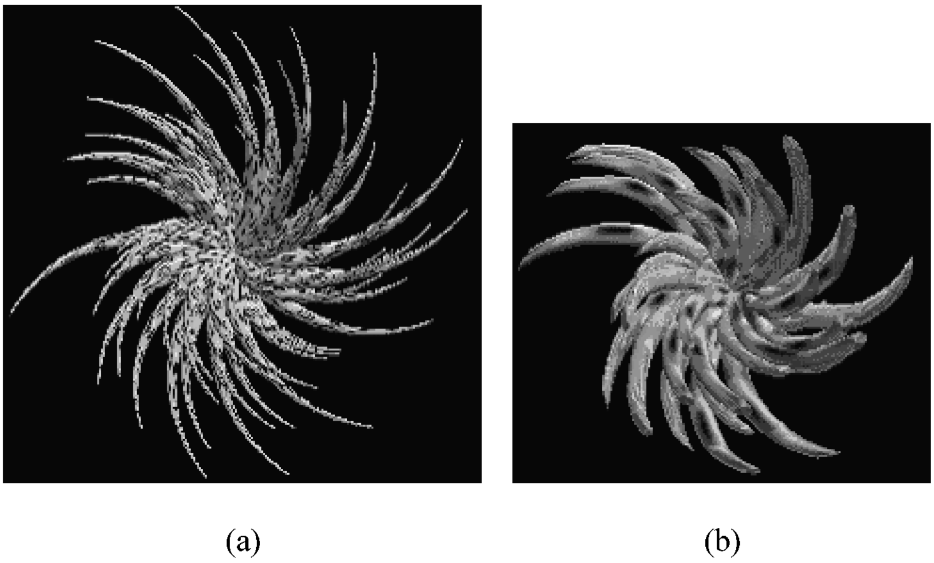


Figure 3. Mutation: Example of parent and mutated offspring. (a) Before mutation. (b) After mutation.

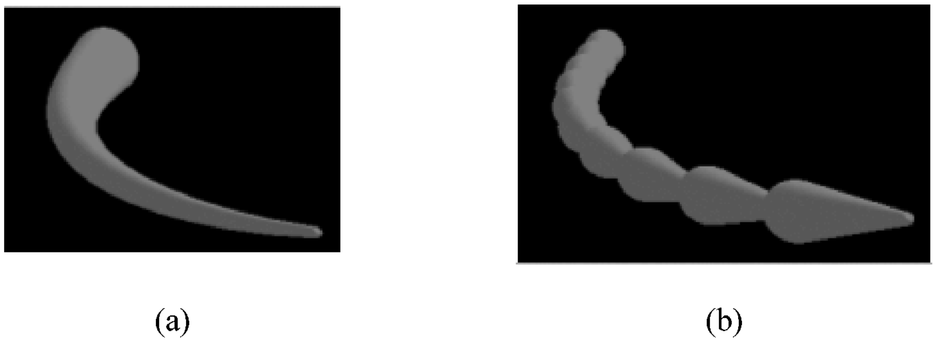


Figure 4. Two segment patterns defined by variable 1 (see Table 1). (a) Straight. (b) Triangular.

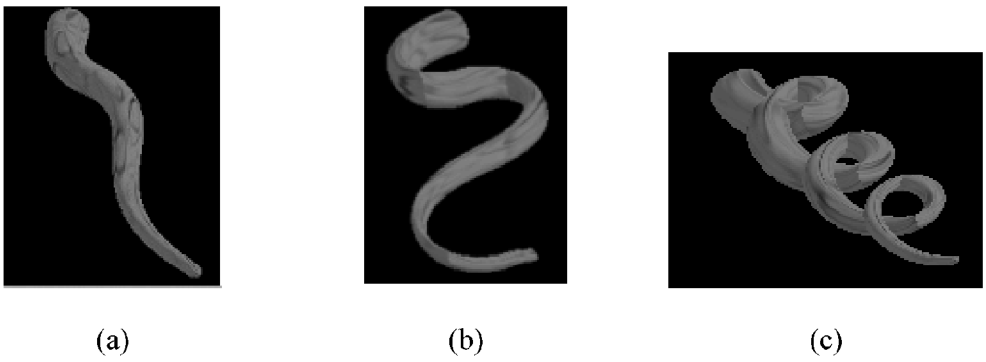


Figure 5. Three branch shapes defined by variable 2 (see Table 1). (a) Circular with {min $x = 0$, max $x = 20$, min $y = 0$, max $y = 50$}. (b) Circular with {min $x = 0$, max $x = 20$, min $y = 0$, max $y = 150$}. (c) Curled with number of curls = 3.

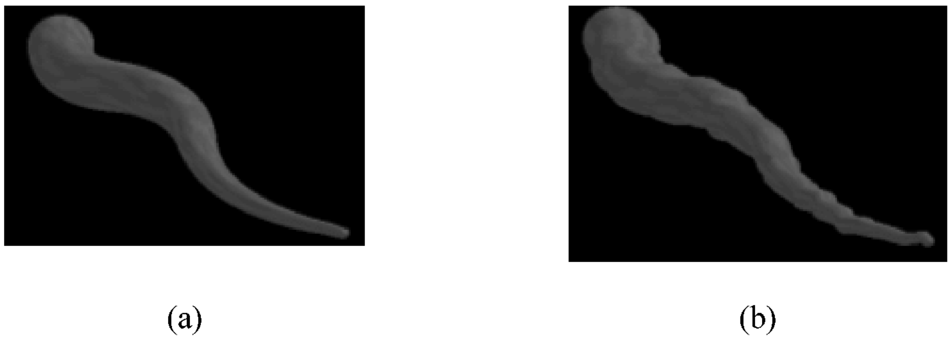


Figure 6. The effect of noise. (a) Without noise. (b) With noise.

A plant's genome contains a fixed number of variables that ultimately determine its appearance. The genome is divided into two main parts:

- *Branch segments*, which contain the variables that determine the form of the branches.
- *Composition segments*, which contain the variables that determine the arrangement of the branches in the plant (position and direction of every branch).

The genome consists of several branch segments, one for each branch in the plant. This segment, which affords the creation of highly complex branches and leaves, is an array of 23 variables that determines the branch's appearance: curled or straight, smooth or rough, length, overall thickness, angle between each cylinder and its successor, and so on. See Table 1 for a complete specification.

The composition segment is an array of 19 variables that determine the position and direction of every branch in the plant; it is divided into two segments:

- *Fan segments*, defining branches that originate from the same point in the plant (Figure 7).
- *Inflorescence segments*, defining fan segments that originate from the same point in the stem (Figure 8).

A fan is represented by 15 variables that determine the branch prototype, how many branches are in the arrangement, the general direction of the branches (all branches at the same angle, angle increase from one branch to the next, etc.), the length of the branches (each branch has a length specified in the genome, but the fan can enlarge adjacent branches), the color and texture of branches, thickness (again, the branch has its own genetically specified thickness, but the fan can increase or decrease it in certain patterns), and so on. See Table 2 for a complete specification of the fan segment.

An inflorescence is represented by four variables that determine its position on the stem (that is, from which point on the stem all the fans are to develop), which fans are in the inflorescence, and the position of each fan (at what range within the 360 degrees around the stem every fan will be located). See Table 3 for a complete specification of the inflorescence segment.

Having described the constituents of a plant's genome, the entire genome is now quite simple to put together: It consists of several branch segments, fan segments, and inflorescence segments, the precise number of segments being under user control

Table 1. Branch segment specification includes the 23 variables listed.

No.	Size (bits)	Range or parameters	Role	Description
1	2	{00 = straight, 01 = segmented, 10 = triangular, 11 = diamond}	Segment Pattern	A value that represents the general pattern of the branch segments: straight, segmented, triangular, diamond-shaped (see Figure 4).
2	2	{00 = straight, 01 = curled, 10 = curved}	Branch shape	This value defines the general shape of the branch: straight, curved, or curled. It determines the angle between each segment and its successor: Straight—Angle 0 between all segments. Curled—The angles between all segments are computed to create a curled look with the number of curls defined by variable 7. Curved—The angles between all segments are computed to create a wavy look with the number of waves defined by variable 7. Every wave starts with the angles specified by variables 3 and 5, and ends in those specified by variables 4 and 6. See Figure 5.
3	9	{0–360}		Minimum *x* angle.
4	9	{0–360}		Maximum *x* angle.
5	9	{0–360}		Minimum *y* angle.
6	9	{0–360}		Maximum *y* angle.
7	4	{1–16}		Number of waves or curls.
8	1	{0 = constant, 1 = random}	Number of segments	A Boolean value that determines if the number of segments in the branch is random or constant. If random, it will be generated in the range defined by variables 9 and 10; if constant, it will be their average.
9	7	{1–128}		Minimum number of segments.
10	7	{1–128}		Maximum number of segments.
11	4	{00 = constant, 01 = thinner, 10 = thicker, 11 = random}	Thickness attributes	General thickness attribute: constant thickness throughout the branch, thinner toward the apex, thicker toward the apex, or random. If the thickness is constant it will be the average of the max and min values below. If random, it will be a random number within the range {min, max} below. If thinner (thicker) it will go from max to min (min to max). *Note*: A branch is composed of connected cones; every cone has its base thickness and apex thickness.

Table 1 continued.

No.	Size (bits)	Range or parameters	Role	Description
12	8	{1–250}		Min base thickness.
13	8	{1–250}		Max base thickness.
14	8	{1–250}		Min apex thickness.
15	8	{1–250}		Max apex thickness.
16	1	{0 = constant, 1 = random}	Length attributes	A Boolean value that determines if the length of the branch is random or constant.
17	5	{1–30}		Min length.
18	5	{1–30}		Max length.
19	1	{0 = constant, 1 = random}	Noise attributes	A Boolean value that determines if noise is applied to the branch (see Figure 6). If so, a number in the range {min, max} below is added to every x and y angle between each two segments; this effect renders the branch's appearance more realistic (see text).
20	9	{0–360}		Min x angle.
21	9	{0–360}		Max x angle.
22	9	{0–360}		Min y angle.
23	9	{0–360}		Max y angle.

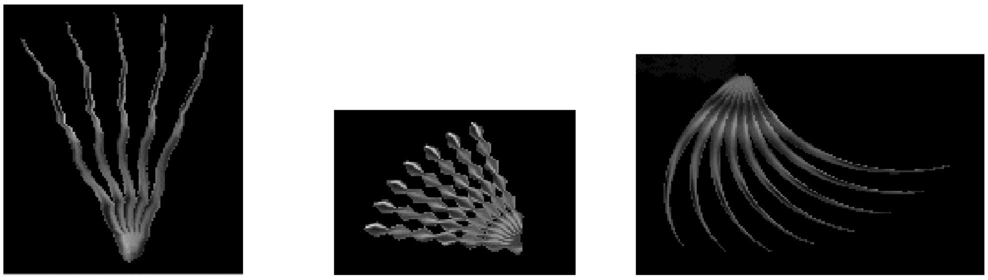


Figure 7. Three examples of fans.

(Figure 9a). Every fan has a variable that determines which branch to use, and each inflorescence has a variable that determines which fans to use (indicated by arrows in Figure 9a). The first branch to the left is the stem of the plant, and the others are its branches and leaves.

Note that two fans can point to the same branch and two inflorescences can point to the same fan, thus affording the possibility that some fans and branches may be unlinked "orphans." Such segments do not affect the plant's appearance, but are available for genotypic manipulation by the evolutionary process.

There are three forms of crossover, among which the user may select:

(1) *Low-level* (Figure 9b): All branches and fans are copied from one parent, all inflorescences copied from the other parent.

(2) *Medium-level* (Figure 9c): Some branches (chosen at random) are copied from one parent, other branches from the other parent.

Table 2. Fan segment specification includes the 15 variables listed.

No.	Size (bits)	Range or Parameters	Role	Description
1	Ceiling (log n)	{1–n}	General properties	A value that determines which type of branch to use from the n branch prototypes found in the genome (Figure 9). $n = 5$ was used in this letter.
2	7	{1–72}		The number of branches that will be in the fan.
3	3	{000 = constant, 001 = increasing, 010 = decreasing, 011 = alternating, 100 = random}	Length properties	A value that determines the general property of the branches' length from the following options: Constant—all branches have the same length. The length of every branch in the fan will be multiplied by the max length factor (variable 5). Increasing—the length of every branch is multiplied by a larger factor than its predecessor, in the range (min length factor, max length factor). Thus, the first branch in the fan will be the shortest, and every subsequent branch will be longer than its predecessor. Decreasing—exactly the opposite of the increasing option. The length of every branch is multiplied by a smaller factor than its predecessor, in the range (min length factor, max length factor). Alternating—the length increases and decreases alternately. The length of the branches is multiplied by increasing factors in the range (min length factor, max length factor). When the length of a branch is multiplied by max length factor, the length of its successors will be multiplied by decreasing factors in the range (max length factor, min length factor). When the length of a branch is multiplied by min length factor, the length of its successors will be multiplied by increasing factors in the range (min length factor, max length factor). Random—the length of every branch is multiplied by a random factor in the range (min length factor, max length factor).

Table 2 continued.

No.	Size (bits)	Range or Parameters	Role	Description
4	4	{1–30} 1 = factor of 0.1 2 = factor of 0.2 ⋮ 30 = factor of 3		Min length factor.
5	4	{1–30} Same as above		Max length factor.
6	3	{000 = constant, 001 = increasing, 010 = decreasing, 011 = alternating, 100 = random}	Angle properties	Similarly to variable 3, a value that determines the general property of the upward angles of the branches, from the following options: constant (i.e., all branches slant upward at the same angle), increasing, decreasing, increasing and decreasing alternately, and random. Constant—all branches slanting upward at an angle defined by variable 7 (min angle). Increasing—the upward-slanting angle of each branch is higher than the upward-slanting angle of its predecessor, in the range (min angle, max angle). Thus, the first branch in the fan slants by min angle, and the last branch slants by max angle. Alternating—the upward-slanting angles of the branches increase and decrease alternately in the range (min angle, max angle). Random—the upward-slanting angle of the branches is a random number in the range (min angle, max angle).
7	9	{0–360}		Min angle.
8	9	{0–360}		Max angle.
9	3	{000 = constant, 001 = increasing, 010 = decreasing, 011 = alternating, 100 = random}	Thickness properties	Similarly to variable 3, a value that determines the general property of the branches' thickness from the following options: constant (i.e., all branches have the same thickness), increasing, decreasing, increasing and decreasing alternately, random.
10	6	{1–50}: 1 = factor of 0.1 2 = factor of 0.2 ⋮ 50 = factor of 5		Min thickness factor.

Table 2 continued.

No.	Size (bits)	Range or Parameters	Role	Description
11	6	{1–50}: same as for variable 10		Max thickness factor.
12	3	{000 = constant, 001 = increasing, 010 = decreasing, 011 = alternating, 100 = random}	Color properties	Similarly to variable 6, a value that determines the general property of the branches' color from the following options: constant (all branches have the same color), increasing, decreasing, increasing and decreasing alternately, random.
13	24	{1–256^3}		Min color.
14	24	{1–256^3}		Max color.
15	7	{0–127}; 0 = no texture, {1–127} = texture id	Texture properties	Branch texture (if any). Each texture has a unique id. Textures are implementation-dependent, to be chosen by the implementer in accordance with her or his tastes.

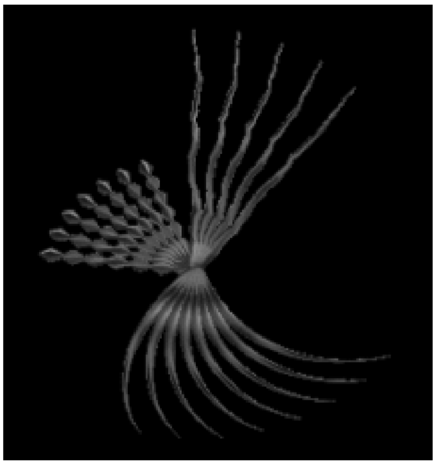


Figure 8. An inflorescence consisting of the fans in Figure 7.

(3) *High-level* (Figure 9d): Each individual gene is randomly selected from one parent or the other.

Mutation is performed by adding random noise to a gene (taking care to remain within the legal boundaries), its *intensity* (probability) being controlled by the user.[1]

One of our main concerns in this work was the attainment of realistic images, toward which end we strove to make our gardens and forests full of similar—but not identical—plants. While originating from the same genome, branches and leaves should not be

1 In the program there is a *Depth* button for selecting the intensity of evolution. The system translates the chosen value to the proper level of evolution, depending on the process selected: low-, medium-, or high-level crossover, or mutation (with a higher *Depth* value implying a higher mutation probability).

Table 3. Inflorescence segment specification includes the four variables listed.

No.	Size (bits)	Range or Parameters	Description
1	9	{0–360}	Start angle—the angle around the stem from which to start drawing the inflorescence (i.e., all fans included in the inflorescence).
2	1	{0 = yes, 1 = no}	Is this inflorescence expressed or not (see text)?
3	7	{0–100}: 0 = bottom of stem ⋮ 50 = middle of stem ⋮ 100 = top of stem	The position of the inflorescence on the stem.
4	m	{1–2^m}	Which fans are included in the inflorescence, from the m fan types in the plant genome (Figure 9); for example, if $m = 5$ (as used in this paper): 00001 = only fifth fan included 10100 = only first and third fans included

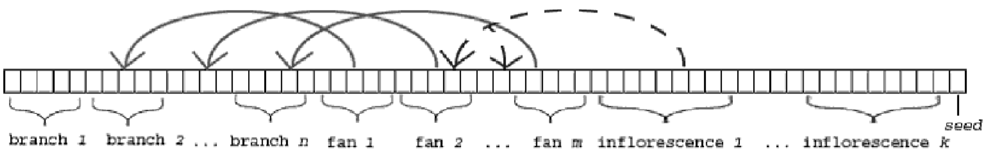
clonelike in appearance, and the arrangement of the branches should not be fully symmetrical.

To achieve this realism effect we added noise to our system, by defining the variables in the genome to represent a *range* of values, rather than constants. When processing such a variable, the final parameter's value is assigned a random value generated within the range. The genome thus no longer represents a specific plant but a plant prototype—a group of similar but not identical plants. Every plant will also look more natural, its branches and leaves differing, and the branch positions being asymmetrical. This is similar in spirit to stochastic L systems [7], which also strive to render the final image more realistic.

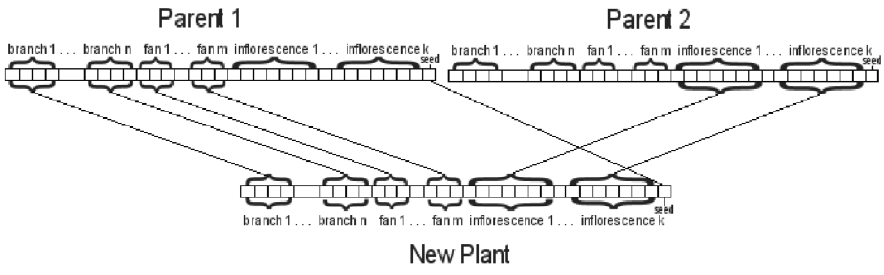
The added noise comes at a cost, though: A stored plant can no longer be recreated precisely, since the genotype-to-phenotype mapping is nondeterministic due to the added noise (Figure 10). To rectify this situation, while retaining the aforementioned advantages offered by randomness, we added another variable to the genome called the *seed*. Each plant has its own unique seed stored within the genome, a numerical value to be used by the pseudorandom-number generator when (re)creating the plant. Since the generator generates the same sequence of random numbers upon being given the same seed as input, we are able to maintain both reproducibility and randomness. A genome now represents a plant prototype—or family—with the added seed determining the unique plant to actually develop. This feature is highly useful when creating a garden of similar-looking trees. Moreover, in order to recreate the garden one need only store the (single) genome prototype along with all the seeds.

4 A Garden Variety

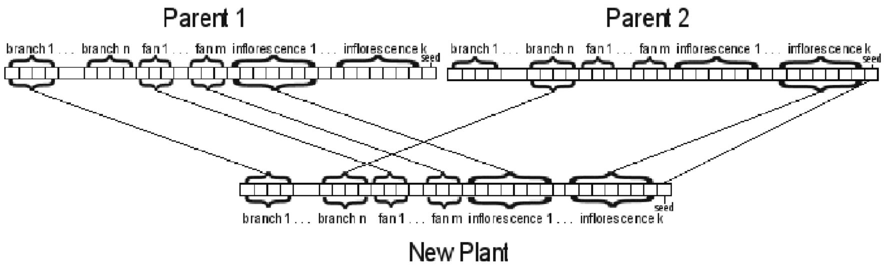
The plants depicted in Figure 11 were evolved using the system described above. A larger gallery (in full Technicolor), as well as the software itself, is available online at moshesipper.com/plantmaker.



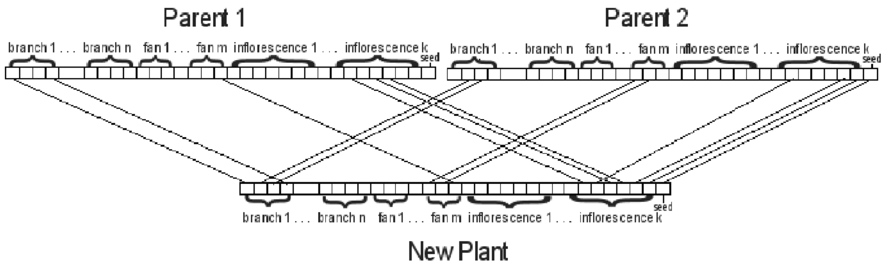
(a)


Parent 1
Parent 2
branch 1 ... branch n fan 1 ... fan m inflorescence 1 ... inflorescence k
seed
New Plant

(b)


Parent 1
Parent 2
branch 1 ... branch n fan 1 ... fan m inflorescence 1 ... inflorescence k
seed
New Plant

(c)


Parent 1
Parent 2
branch 1 ... branch n fan 1 ... fan m inflorescence 1 ... inflorescence k
seed
New Plant

(d)

Figure 9. (a) Plant genome. Arrows indicate branch used by fan (solid arrow), and fan(s) used by inflorescence (dashed arrow). The variable seed is used by the pseudorandom-number generator during plant development (see text). (b) Low-level crossover. All branches and fans are copied from one parent, all inflorescences copied from the other parent. (c) Medium-level crossover. Some branches/fans/inflorescences (chosen at random) copied from one parent, others from second parent. (d) High-level crossover. Each individual gene randomly selected from one parent or the other. For simplicity crossover is shown for two parents only.

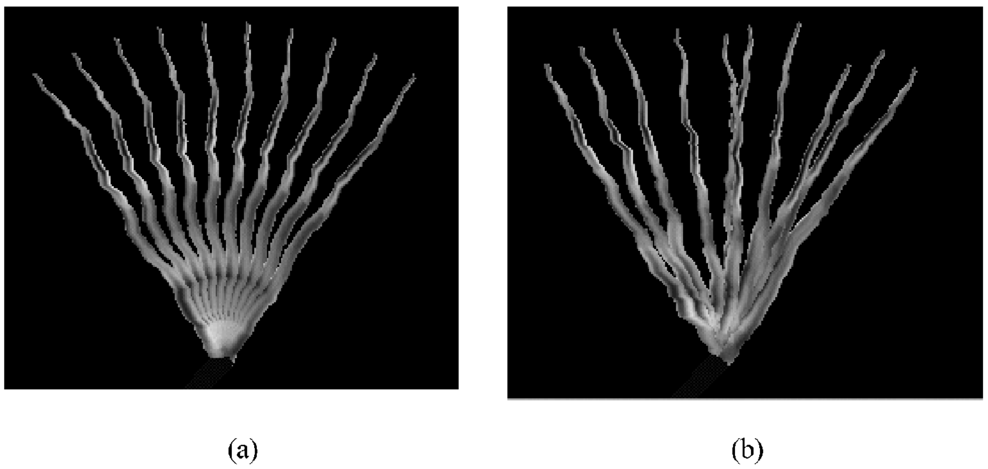


Figure 10. Adding noise to the plant-generating process. (a) Fan before noise was applied has a somewhat artificial look. (b) Fan after noise was applied is more asymmetrical and thus less artificial in appearance.

5 Discussion

In designing our system the main goal has been the facilitation of fashioning realistic plant imagery (Section 1). We aimed at combining the ease of production of systems such as Dawkins' [1], Rooke's [8], and Sims' [9] with the realistic look of L systems. Our experience with the system points to its ease of use (as any reader who downloads the software can judge), while the realism of the few sample images provided here is submitted to the user's judgment. As we concentrated specifically on plants and flowers, our model is simpler than that of Sims [9] and is more readily reproducible.[2]

As noted above, segments may remain unlinked in the genome, and thus be unexpressed visually. This engenders a lifelike scenario wherein the offspring may inherit unexpressed traits from its parents, only to be expressed in its own offspring—the grandchildren. The existence of unexpressed segments in the genome also increases the diversity of the population, while at the same time rendering it less clonelike in appearance.

The addition of a seed variable to the genome creates both a more random—and thus lifelike—appearance, while affording perfect repeatability. This feature facilitates the creation of a forest from the same prototype, with similar but nonidentical plants. Diversity thus comes about through two distinct processes—evolution and randomization.

We believe our system presents a small step forward toward the creation of *natural artificial worlds*.

References

1. Dawkins, R. (1986). *The blind watchmaker*. New York: W. W. Norton & Company.

2. Hornby, G. S., & Pollack, J. B. (2002). Generating high-level components with a generative representation for body-brain evolution. *Artificial Life, 8*, 223–446.

3. Komosinski, M., & Ulatowski, Sz. (1999). Framsticks: Towards a simulation of a nature-like world, creatures and evolution. In D. Floreano, J.-D. Nicoud, & F. Mondada (Eds.),

2 Note that Sims provides very little detail regarding the evolution of 3D plants. Our comparison is with respect to his general evolutionary art system [9].

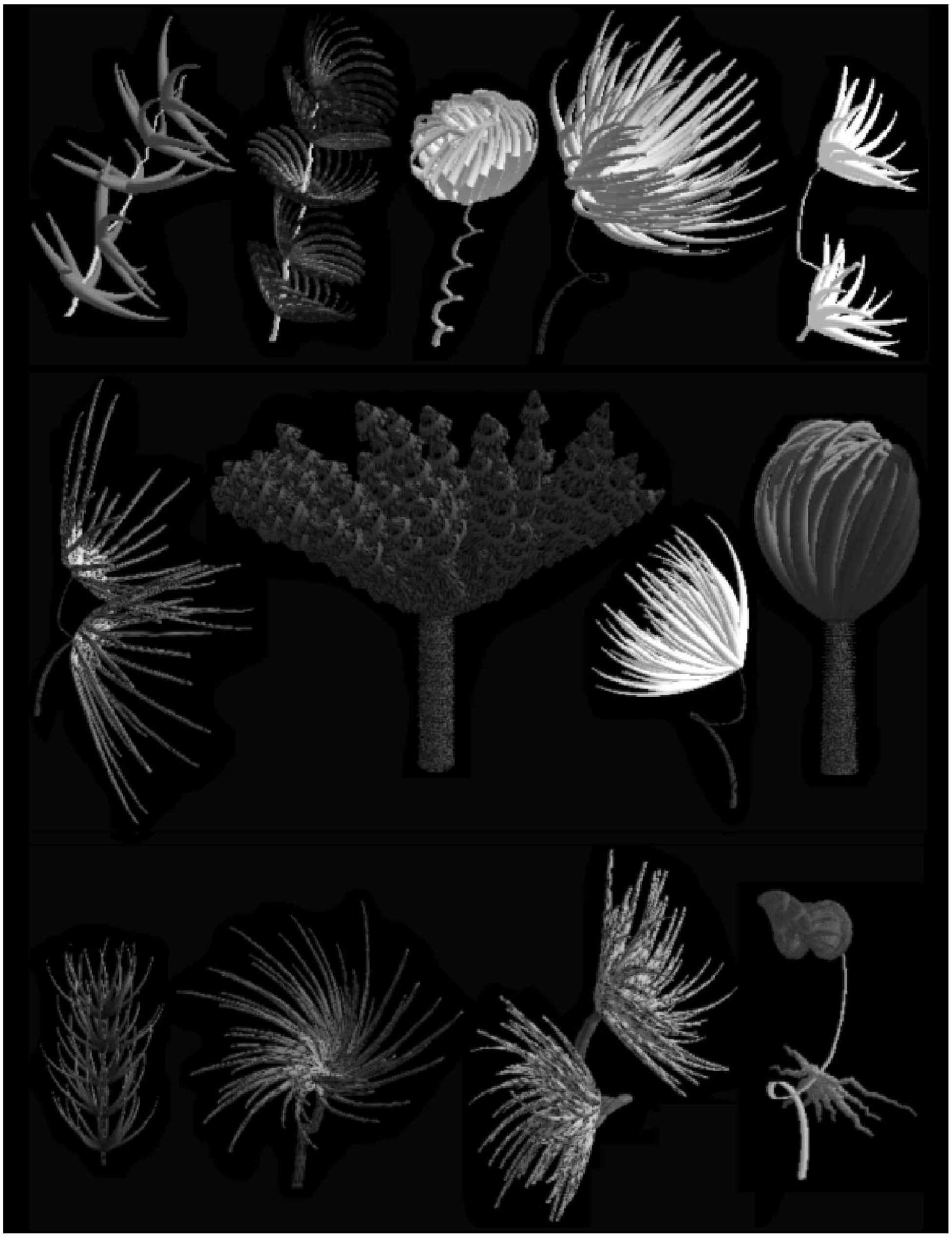


Figure 11a.

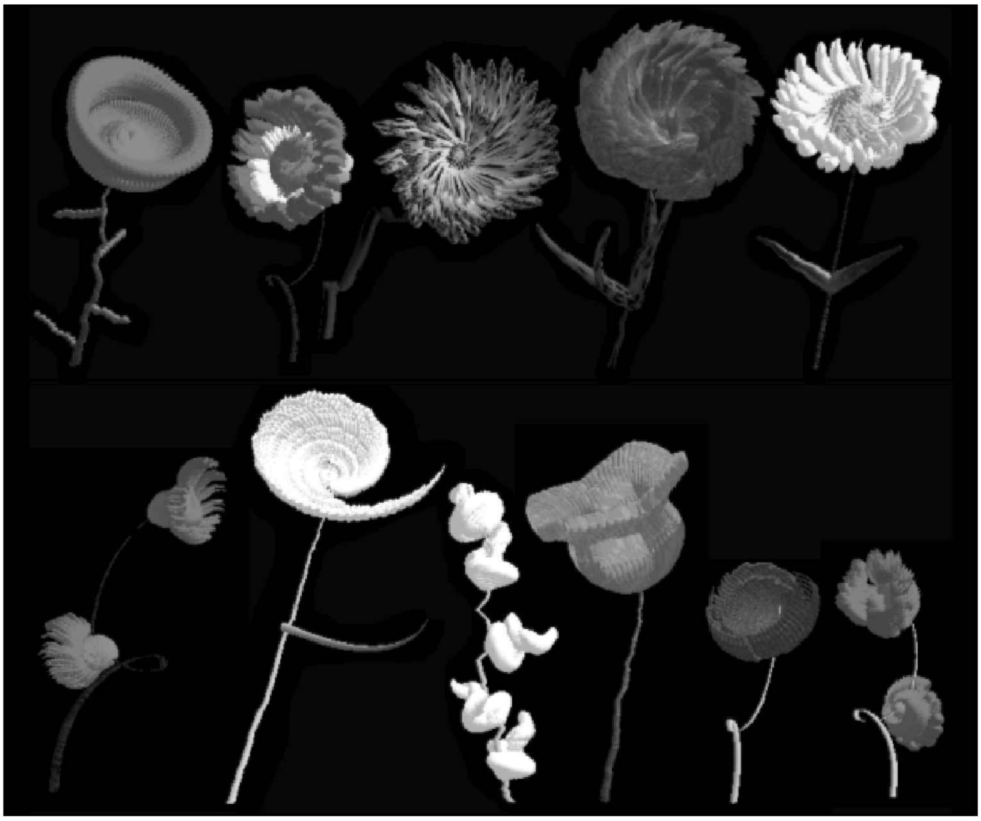


Figure 11b. A garden variety.

Proceedings of 5th European Conference on Artificial Life (ECAL99) (pp. 261–265). Berlin: Springer-Verlag.

4. Lipson, H., & Pollack, J. B. (2000). Automatic design and manufacture of artificial lifeforms. *Nature, 406*, 974–978.
5. Nehaniv, C. L. (2003). *BioSystems: Special Issue on Evolvability* (in press).
6. Ochoa, G. (1998). On genetic algorithms and Lindenmayer systems. In A. Eiben, T. Baeck, M. Schoenauer, & H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature (PPSN V)* (pp. 335–344). Berlin: Springer-Verlag.
7. Prusinkiewicz, P., & Lindenmayer, A. (1990). *The algorithmic beauty of plants*. New York: Springer-Verlag.
8. Rooke, S. (2003). *The evolutionary art of Steven Rooke*. http://www.azstarnet.com/~srooke/
9. Sims, K. (1991). Artificial evolution for computer graphics. *Computer Graphics, 25*, 319–328.
10. Sims, K. (1994). Evolving virtual creatures. *Computer Graphics (Annual Conference Series)*, 43–50.
11. Sims, K. (1994). Evolving 3D morphology and behavior by competition. In R. A. Brooks & P. Maes (Eds.), *Artificial Life IV* (pp. 28–39). Cambridge, MA: MIT Press.
12. Sipper, M. (2002). *Machine nature: The coming age of bio-inspired computing*. New York: McGraw-Hill.
13. Taylor, T., & Massey, C. (2001). Recent developments in the evolution of morphologies and controllers for physically simulated creatures. *Artificial Life, 7*, 77–87.