Digital Creativity, Vol. 1(18), 2007

Swarm grammars: growing dynamic structures in 3D agent spaces

Christian Jacob and Sebastian von Mammen

University of Calgary, Calgary, Canada

cjacob@ucalgary.ca, s.vonmammen@ucalgary.ca

Abstract

We present a new way of dynamically growing and breeding structures in 3D space through swarming agents. Different agent types and the way they evolve over time is specified by a swarm grammar similar to Lindenmayer systems. We expand common L-system string interpretation from a single turtle to a multitude of turtles which behave like a swarm. By describing swarm agents within the framework of formal grammars, we build a bridge from symbolic production systems (rewrite systems) to three-dimensional real-time construction procedures that are executed by reactive and interacting agents which move in simulated physical 3D spaces.

We introduce constructor agents, their formal representation in swarm grammars and demonstrate by examples how (1) the swarm rules, (2) the agent parameters and (3) the environment can influence the actual construction and growth processes that are initiated and directed by the swarms.

In order to facilitate exploration of a large variety of swarm grammars, we apply interactive evolutionary design methods to create swarm grammar sculptures and 3D structures.

Keywords: swarms, swarm intelligence, swarm grammars, design of 3D structures, generative design, rewriting systems, Lindenmayer systems, agent-based design, multi-agent system.

1. Introduction

Looking at life around us, we are immersed in a natural world of massively parallel, decentralized biological 'information processing' systems; a world that exhibits fascinating emergent properties in many ways due to developmental processes, growth, and self-organization. In fact, our very own bodies are the result of emergent patterns, as the development of any multi-cellular organism is determined by localized interactions among an enormous number of cells – carefully orchestrated by enzymes, signaling proteins and other molecular 'agents.' What is particularly striking about these highly distributed developmental processes is that a centralized control agency is completely missing. This is also the case for many other biological systems, such as termites which build their nests without an architect that draws a plan, or brain cells evolving into a complex 'mind machine' without an explicit blueprint of a network layout.

Obviously, being able to understand, build and harness the emergent properties of such systems would be highly beneficial for helping us to create a new generation of design and manufacturing techniques. Designers of complex systems could utilize their adaptability and robustness. Such systems would construct themselves, through self-organization. However, system designers and programmers are facing an enormous challenge. How can we actually build highly distributed systems of which we have only limited understanding? We have to invent new ways of building, maintaining, and controlling such systems.

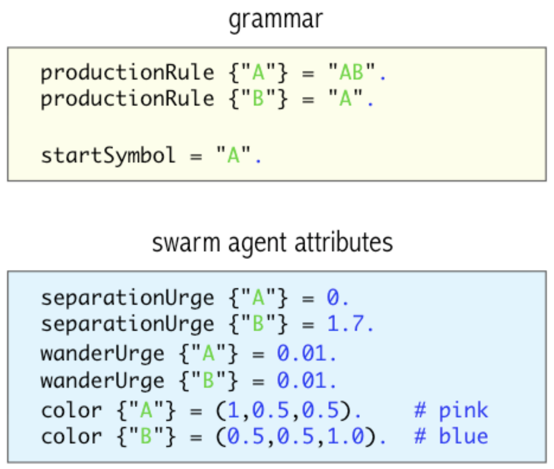
The *Swarm Grammars* we are going to present here provide a first step towards a new methodology for the creation and design of 3D forms and shapes. With swarm grammars (SGs) we capture growth processes that result from the interactions of swarming agents while they create branching structures in 3-dimensional space.

Generative representations of design patterns for 3D forms, such as Lindenmayer systems (L-systems), have been used very successfully to model growth processes. Originally, L-systems were developed to capture growth in bacterial and yeast cells (Lindenmayer 1968; Rozenberg & Lindenmayer 1986). Soon L-systems were investigated in the context of formal languages (Rozenberg & Salomaa 1980; Rozenberg et al. 1986). Capturing the developmental processes that lead to branching patterns in plants became another major area of study involving L-system grammars (Prusinkiewicz & Hanan 1989; Prusinkiewicz & Lindenmayer 1990; Hanan 1992; Prusinkiewicz 2004). Other models of branching structures in dendritic growth of neurons (Hamilton 1994) and in arteries (Zanis 2001) have used L-systems as well.

More recently, generative approaches using L-systems have explored architectural designs (Coates 1999; Hemberg 2001; Jackson 2001), designs for modular robots (Hornby et al. 2001; Hornby & Pollack 2001a), efficiently encoded physical designs (Hornby & Pollack 2001b), evolvable hardware (Haddow 2001) and solutions in computational mechanics (Alber et al. 2002).

In L-systems, a formal grammar specifies rules that capture the step-by-step growth process by rewriting a string of symbols, which are subsequently translated into graphical objects through a turtle interpretation. A turtle is a virtual drawing device that is navigated in 3D space following the symbolic commands of the string. In swarm grammars we substitute the turtle interpretation by a *swarm interpretation*. Instead of a single turtle following the path described by an L-system, a swarm of ‘turtle agents’ interpret the grammar rules. This simple expansion from one interpreting turtle to a swarm reveals new dimensions in performance, dynamics and complexity of the resulting structures. The swarm agents are not only controlled by the grammar rules, but have the potential to interact among each other and with their environment. In fact, collision resolution among branching structures can be accounted for quite easily through parallel swarm-based turtle interpretation. This does not only lead to more interesting designs emerging from the swarm’s dynamics, but also engages the designer in an interactive dialog with the creative process, by introducing alternate swarms or other static and dynamic environmental components that can influence a swarm’s developmental processes.

Describing the swarm grammar approach in more detail, we proceed in the following manner: In Section 2 we define swarm grammar systems and their associated building agents. Examples of building processes implicitly described by swarm grammars are illustrated in Section 3. Here we also show which effects the rewrite rules and the agent parameters have on the actual swarm-driven building process. In Section 4 we show how swarm grammar agents encounter other entities within their environment and how these interactions influence the building dynamics and the resulting compositions. We describe the exploration of new swarm grammar rules and agent parameters through an evolutionary system in Section 5. A brief comparison to L-systems—with respect to parallel turtle interpretation, in particular—is presented in Section 6. A short outline of future expansion possibilities of swarm grammar systems in Section 7 concludes this contribution.



```
productionRule {"A"} = "AB".
productionRule {"B"} = "A".

startSymbol = "A".
```

```
separationUrge {"A"} = 0.
separationUrge {"B"} = 1.7.
wanderUrge {"A"} = 0.01.
wanderUrge {"B"} = 0.01.
color {"A"} = (1,0.5,0.5).    # pink
color {"B"} = (0.5,0.5,1.0).  # blue
```

Figure 1. Example of a swarm grammar system with two rewrite rules, a start symbol, and a set of attributes for agent types *A* and *B* (see Section 2.2 for more details).

2. Swarm Grammar System

In this section we describe the two key parts of a swarm grammar system: (1) a set of rewrite rules, which determine the composition of agent types over time, and (2) a set of agent specifications, which define agent type specific parameters that govern the agents' interactions.

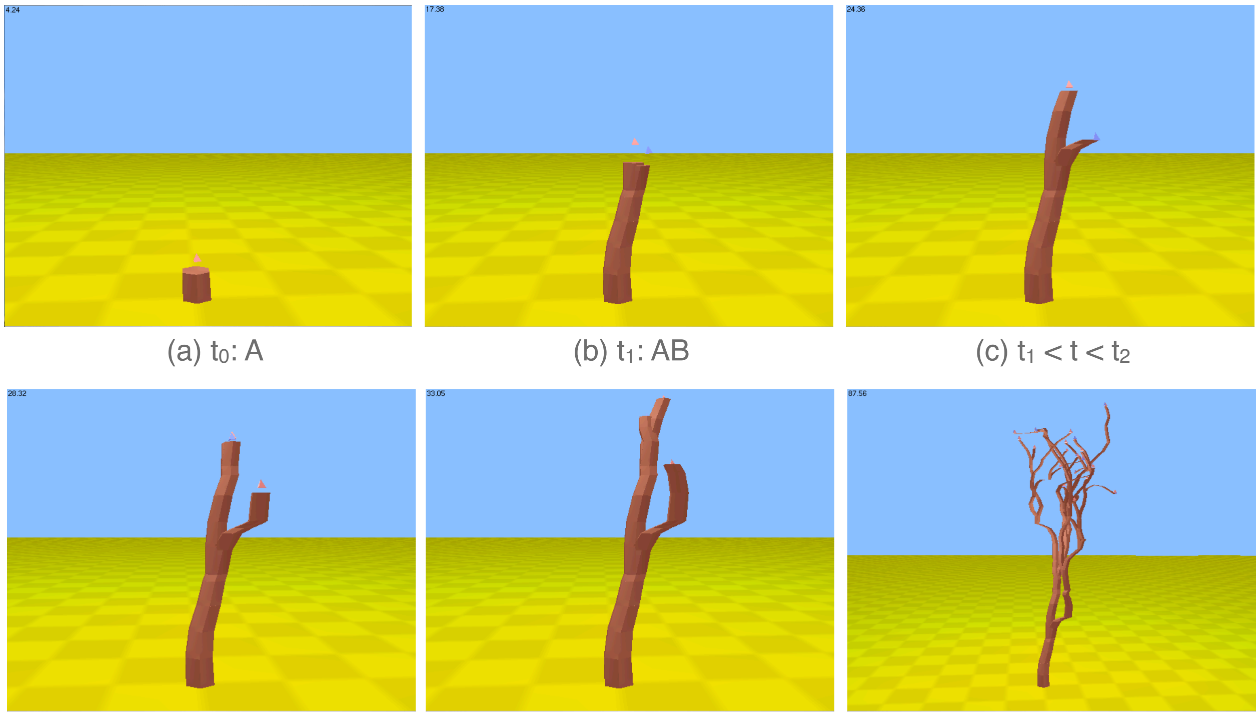
2.1. The Swarm Grammar

A swarm grammar system *SG* = (*SL*, Δ) consists of a rewrite system *SL* = (*α*, *P*) and a set of agents Δ = {a_1, a_2, ..., a_n}. The rewrite system *SL* is an L-system with axiom *α* and production rules *P* (Jacob 2001). In the simplest form of context-free 0L-systems, each rule has the form $p \rightarrow s$, where *p* is a single symbol over an alphabet Ω, and *s* is either the empty symbol (λ) or a word over Ω. Each agent a_i is characterized by a set of attributes, which can include its geometrical shape, color, mass, vision range, radius of perception and other parameters such as separation or cohesion urges that determine its behavior while encountering its environment. Figure 1 gives an example of such a swarm grammar with two types of agents. The rewriting process begins with start symbol A. In the first iteration of applying any matching rules, only the first rule is applicable, hence A is rewritten into AB. At the next iteration, both rules apply: A is rewritten into AB, and B is rewritten into A. The resulting string is ABA. Further rewriting will result in the following word sequence:

t_0:	A
t_1:	AB
t_2:	ABA
t_3:	ABAAB
t_4:	ABAABABA
	...

Here each t_i represents a decision point[1] where an agent triggers the application of the next *S*L-system iteration with the string describing the current composition of the

[1] In the following examples a decision point coincides with the iteration number of the SL-system.



(a) t_0: A (b) t_1: AB (c) $t_1 < t < t_2$

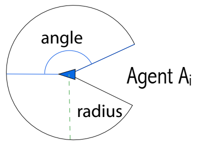
(d) t_2: ABA (e) t_3: ABAAB (f) $t > t_3$

Figure 2. Step-by-step illustration of swarm interpretation in 3D space (see text for details).

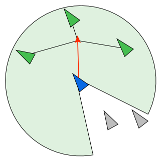
swarm. In the example above we have five type-A and three type-B swarm agents after decision point t_4. Figure 2 shows the first steps of the swarm interpretation in 3D space. The single type-A agent starts its vertical ascent, building a cylindrical shape on its way. At decision point t_1 agent A is replaced by a new agent of type A and a type-B agent. A-agents are the only ones that move, whereas B-agents build a bent branch tip and then stop (Fig. 2(c)). At time point t_2 agent A is replaced by agents of type A and B, and the former B-type agent is replaced by an A-agent. Figure 2(f) illustrates the branching structure resulting after a few more iterations.

2.2. The Swarm Agents

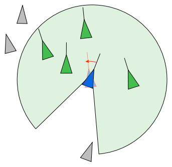
In our demonstrations, a swarm agent is represented as a pyramid pointing in the direction of its velocity vector (Fig. 3). Each agent's awareness of other flock mates is determined by its field of perception, which is defined by a radius and an angle as illustrated in Figure 3(a). An agent will only interact with those agents that are within its field of perception. We call these agents its *neighbors*. Both the radius and angle of the field of vision are part of an agent's attribute set.



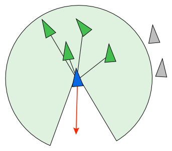
(a) Agent perception



(b) Cohesion



(c) Alignment



(d) Separation

Figure 3. Basic interactions with other agents.

The velocity vector V of an agent is updated according to the following formula:

$$V = c_1\ V_1(d) + c_2\ V_2 + c_3\ V_3 + c_4\ V_4 + c_5\ V_5.$$

Here we follow the simple *boids* model of interaction rules (Reynolds 1987), where an agent changes direction and adjusts its speed according to three influential factors (Fig. 3(b)-(d)):

- *separation* ($V_1(d)$): steer away from the collective of neighbors if the minimum distance is smaller than a *crowding* value d (Kwong 2003).
- *cohesion* (V_2): move toward the average position of local flock mates, and
- *alignment* (V_3): reorientation towards the average direction of its neighbors.

Vector V_4 points to the center of the simulated 3D world and V_5 represents a random unit-length vector to add some noise. The weights c_1, ..., c_5 determine how much influence each factor has on the agent. Each of these 'urges' is specified for an agent type as part of a swarm grammar. In Figure 1, for example, *separation* and *wander* urge correspond to weights c_4 and c_5, respectively.

An agent stops applying the SL-system rules when it runs out of energy, which is passed on from one generation of agents to the next. The energy level also influences certain properties of the built 3D structures such as, for example, the radius of the cylinders.

In summary, an SL-grammar repeatedly defines the successors of an agent. Predefined parameters determine when a construction element is built, when a production rule is applied, how much energy is lost through the creation of a construction element, and when the agent runs out of energy and thus is unable to reproduce.

3. SG Agents in Action

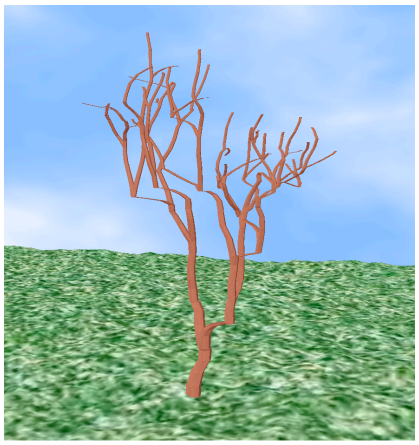
Now let us have a look at the effects that emerge when we modify the set of production rules and the agent parameters that determine their flocking behaviors. The following examples will demonstrate the high degree of interaction dynamics and the resulting variety of outcomes to be expected from swarm grammar systems that build 3D structures.

3.1. Changing the SL-system Rules

We first discuss a small sample of tree-like structures that result from various sets of production rules. In order to illustrate some of the basic effects, we use only a fairly limited number of swarm agents.

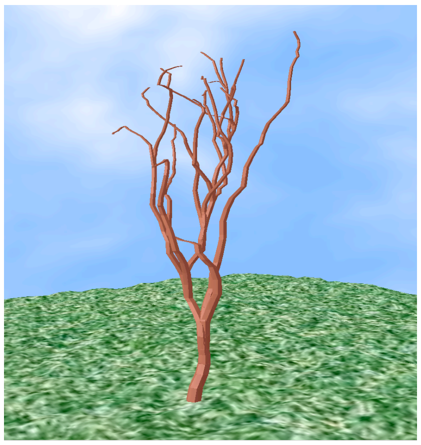
Agent Type	Separation c_1	Random c_5
A	0	0.01
B	1.7	0.01
C	13.7	0

Table 1. Flocking parameters of agent types *A*, *B*, and *C*. All other parameter weights (c_2, c_3, and c_4) are set to zero.



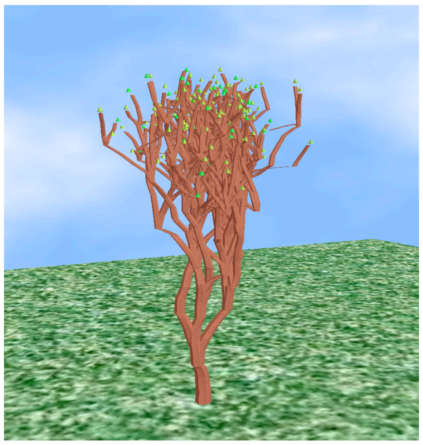
(a) Agents: 87

$SL_a = (A, \{A \rightarrow AB, B \rightarrow A\})$



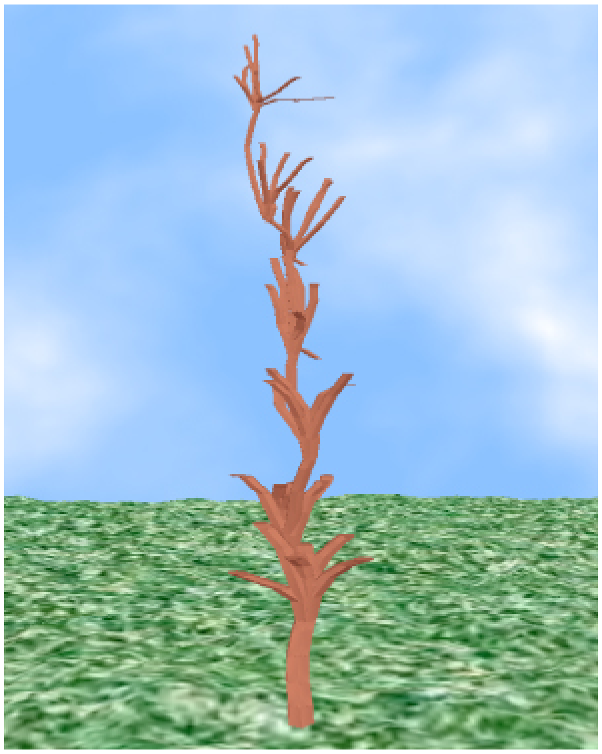
(b) Agents: 64

$SL_b = (A, \{A \rightarrow BAB\})$



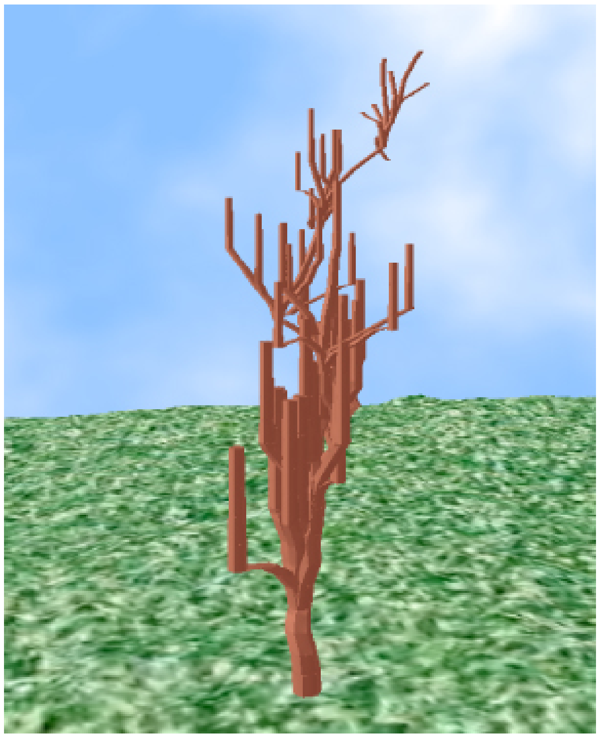
(c) Agents: 407

$SL_c = (A, \{A \rightarrow ABA, B \rightarrow A\})$



(d) Agents: 50

$SL_d = (A, \{A \rightarrow BBBABBB, B \rightarrow \lambda\})$



(e) Agents: 86

$SL_e = (A, \{A \rightarrow BBBABBB, B \rightarrow C, C \rightarrow \lambda\})$

Figure 4. Examples of branching structures created from agent interactions governed by different swarm grammars.

Consider three types of swarm agents—*A*, *B*, and *C*—with parameters as in Table 1, which describe the weights of their separation urge (c_1) and random movement (c_5). The remaining behavior parameters (c_2, c_3, c_4, d) are set to zero. Initially, all agents are oriented upwards, hence will move towards the top (increasing their *y* coordinate).

The interpretation of swarm grammar $SL_a = (\alpha = A, P = \{A \rightarrow AB, B \rightarrow A\})$ results in a tree-like structure with sparse branches, which makes it easy to analyze (Fig. 4(a)). The 'natural' look of the overall tree can be attributed to the small degree of random movements of both types of agents. *A*-type agents move upwards with no urge to separate, whereas any *B*-agent moves away from agents of type *A*, due to its urge for separation ($c_1 = 1.7$). Hence the arrangement of the branches is mainly a consequence of the agents' interactions.

With the even simpler grammar SL_c, the style of the tree looks similar to the structure from SL_a (Fig. 4(b)), where *B*-agents only place stationary building blocks and then stop.

A different branching pattern is shown in Figure 4(c), where a slightly larger number of *A*-agents is generated at each decision point by adding an extra *A*-type agent compared to SL_a. This leads to bursting agent reproductions, a more expansive growth of the branches, and the formation of a denser canopy. The small green objects at the branch tips represent the swarm agents that are still to finish their next building step.

However, an increased number of generated agents does not always mean that the complexity of the emerging structures increases as well. The SL-system in Figure 4(d) produces a large number of agents, but the outcome is quite simple, as type-*B* agents only get the chance to establish a short side branch and are removed before the next building step.

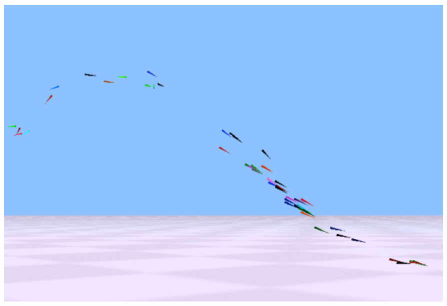
In Figure 4(e), a third agent type, *C*, is added, which has a very high separation urge with no random component added (Table 1). As *C*-agents are also oriented vertically at their time of creation, they are responsible for the vertical branch endings.

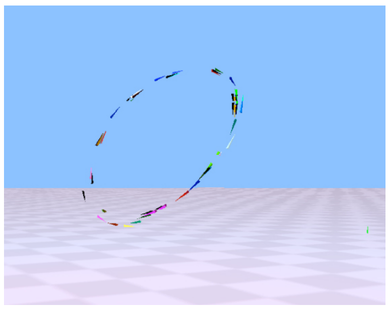
3.2. Changing the Agent Parameters

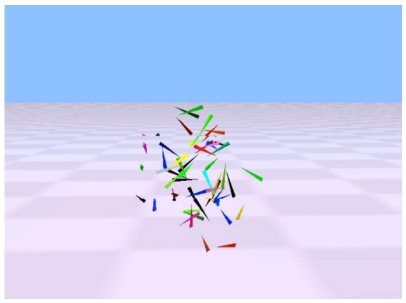
Instead of changing the SL-system rules, we are now going to modify the agents' flocking parameters and look at the consequences with regard to the generated 3D structures. We start from a swarm grammar with a single rule that enables forked branching:

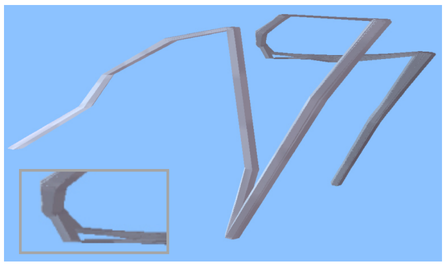
$$SG_{simple} = (\alpha = A, P = \{A \rightarrow AA\}, \Delta).$$

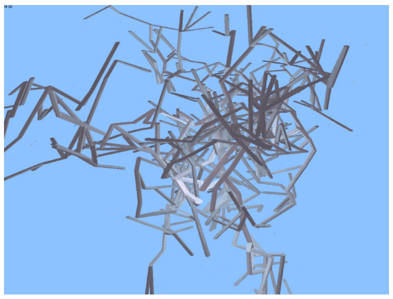
At each iteration step, one type-*A* agent reproduces into two *A*-agents. As there is only one type of agents, they all share the same flocking parameters listed in Table 2. These settings were reported by Kwong (2003) who investigated swarm interaction patterns and their evolution in more detail. Kwong discovered a range of parameter settings, where the agents displayed formations such as figure eights, rings and other choreographed arrangements (see also Kwong & Jacob, 2004). Figures 5(a), (b), and (c) show snapshots of a line formation, a ring formation, and a loose cluster emerging from the parameter sets (1), (2), and (3) in Table 2, respectively. Here the additional parameter *crowding* is introduced. If the distance to a neighbor is within *crowding* range, the separation urge is in effect. This allows an agent to influence only a subset of its actual neighbors. The parameters a_{max} and v_{max} denote the agents' maximum allowed acceleration and velocity, respec-


(a)


(b)


(c)


(d)


(e)

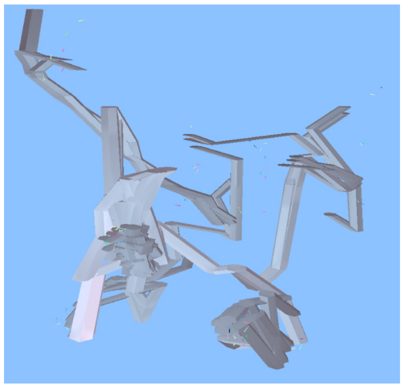

(f)

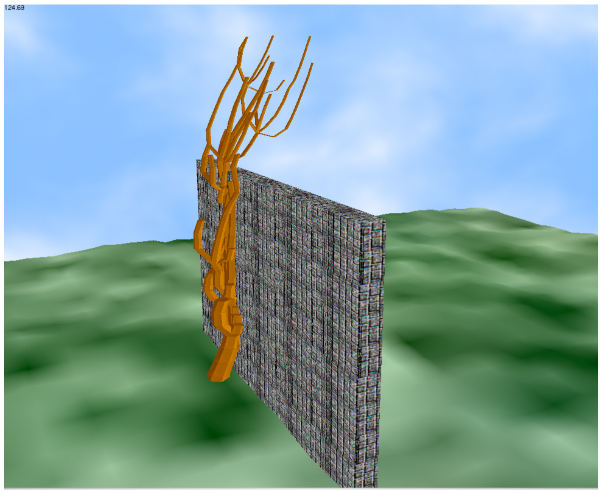
Figure 5. Choreographed swarms are employed for building 3D sculptures. Top: snapshots of choreographed flocking behaviors; (a) line formation, (b) ring formation, (c) loose cluster formation. Bottom: the structures built by the corresponding swarm grammar agents. Lighter (darker) colors of building blocks indicate earlier (later) addition during the building process (flocking parameters according to Table 2). Videos of these choreographed swarms are available at: http://www.swarm-design.org/SwarmGrammars/movies/.

tively. No agent can fly faster or accelerate faster than specified by these limits.

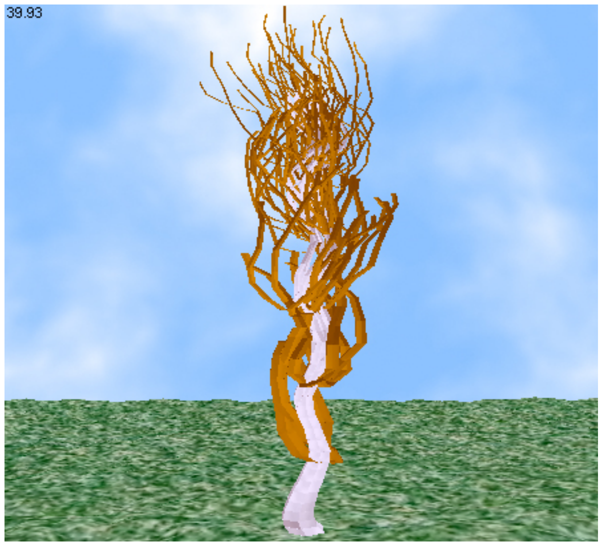
Table 2. Flocking parameter settings that lead to the following behaviors: (1) large ring formation, (2) line formation, (3) a loose stationary cluster, and (4) a figure eight.

	(1)	(2)	(3)	(4)
Separation (c_1)	1	5	5	2
Cohesion (c_2)	10	8	0	3
Alignment (c_3)	5	7	2	7
World Ctr (c_4)	14	8	7	6
Random (c_5)	1	5	6	3
Crowding	0.14	0.14	0.23	0.01
a_{max}	39	38	40	40
v_{max}	9	13	6	6

The bottom images in Figure 5 show the structures that result from using the same types of agents to interpret swarm grammar SG_{simple} as described above. The building blocks of the depicted structures bear different colors (or grey levels) so that their composition over time is visualized. Lighter-colored building blocks are built earlier. The structure in Figure 5(d), for example, was built from left to right, with intermittent changes of the swarm's direction. This construction does not seem to involve any



(a)



(b)

Figure 6. Swarm grammar agents interacting with objects in their environment: (a) a static wall constricts agents from reaching a goal point behind it; (b) agents tend towards a goal point that orbits above the construction center. Videos of these environmental interactions are available at: http://www.swarm-design.org/SwarmGrammars/movies/.

branching due to agent separation urges. The smooth bands originate from the agents' almost perfect flight coordination while constructing very similar, almost parallel fibers. Looking a little closer, however, reveals a small gap at a U-turn slightly off the center at the top right of the image (see Fig. 5(d) inset).

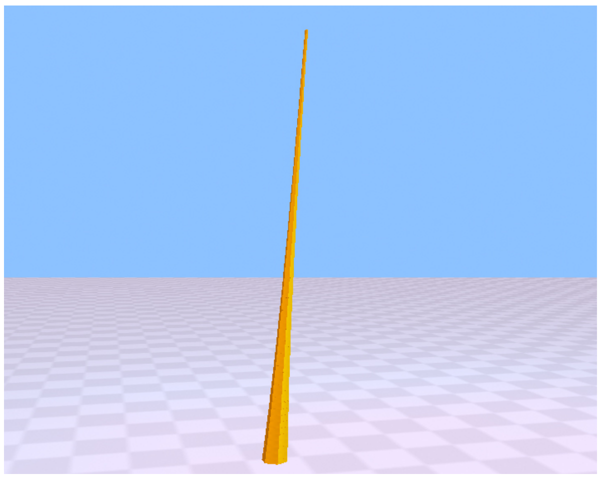
The structure in Figure 5(e) evolves spherically from a center point. The large ring flocking behavior of the swarm contributes to a spiky and impulsive character of this growing 'sculpture'.

Our third example of combining choreographic swarms with swarm grammars involves flocking behavior where the agents form loose, temporary clusters, then disperse and regroup to form new clusters at a different location. This behavior is induced by the parameters in Table 2(3). The formation of one of these clusters is depicted in Figure 5(c). Looking at the corresponding structure built by the swarm grammar agents, the sites of cluster formation are clearly identifiable as 'knots'. Since the flocking parameters allow for a rather dynamic flight, single agents can leave one cluster and join another one at a different location.

4. Interactions with the Environment

In this section we present three different kinds of interaction with both static and dynamic elements within the environment. Table 3 lists the parameters for the six types of agents we are going to employ.

	D	E	F, G, H	I
Separation	0	10	80	33
Cohesion	0	0	0	10
Alignment	0	0	10	11
World Ctr	10	1	1	5
Random	10	2	4	0
Crowding	0	10	10	1


(a)

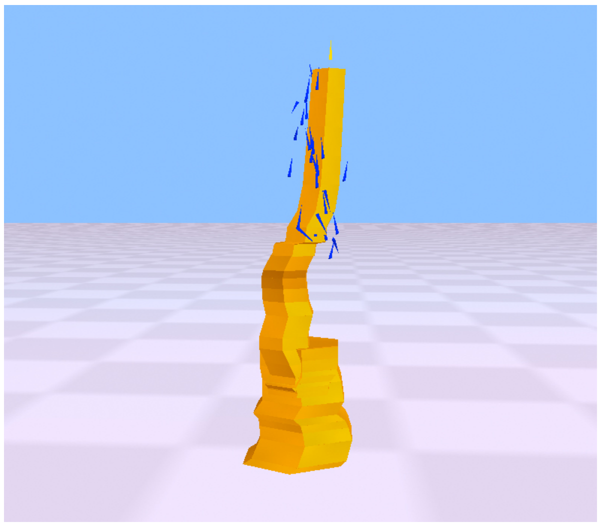

(b)

Figure 7. Constructing swarms interacting with another flock of agents. (a) Resulting construction with no other swarm present; (b) the same swarm's movements are influenced by another (non-constructing) swarm of agents shown in blue. Videos of these swarm constructions are available at: http://www.swarm-design.org/SwarmGrammars/movies/.

	D	E	F, G, H	I
a_{max}	30	30	10	27
v_{max}	2	5	4	2

Table 3. Parameter settings for agent types *D*, *E*, *F*, *G*, *H*, and *I*.

4.1. Swarm—Object Interaction

Figure 6(a) shows an example of agents interacting with non-moving objects in their environment. Agents of types *F*, *G* and *H* tend to move towards the world center, which, in this case, is located beyond the wall and far up in the sky (like a sun). Whenever a swarm agent tries to penetrate the wall, it bounces back as its velocity vector's *x*- and *z*-coordinates are reversed. This implements a simple collision detection with static objects. As soon as the swarm structure has outgrown the wall, the agents are no more prevented from moving towards their destination.

As soon as the world center becomes dynamic, its movement pattern is reflected in the construction of those swarm agents that tend towards it. In Figure 6(b) the world center orbits far up in the sky and around the *y*-axis of the simulation. Both agent types, *D* and *E*, are attracted towards the moving world centre. Consequently, the structure they build reflects an upward, twisted growth pattern. In order to better recognize the constructors, *D*-type agents are assigned a very light and agents of type *E* a darker color. As *D*-agents do not feel the urge to separate from their neighbors, they almost perfectly drive upwards around the *y*-axis. The constructions from agents of type *E* outgrow the ones from the *D*-type since *E*-agents are allowed a greater maximum velocity (compare Table 3).

4.2. Swarm—Swarm Interaction

In the previous examples, the swarm grammar agents were interacting with either static or dynamic objects. Now, consider a second swarm that is not part of a swarm grammar, but exhibits flocking behavior within the environment. Both swarms influence each other as soon as some of their individuals enter the field of vision of the other swarm agents.

These swarm-swarm interactions are hard to capture in a screenshot. However, the swarm grammar agents witness the exertion of influence from the other swarm by leaving a trace in the 3D construction space.

We look at another simplistic swarm grammar:

$$SG_{straight\text{-}up} = (\alpha = I, P = \{I \to I\}, \Delta).$$

Figure 7(a) shows the structure that is built by this swarm grammar, with no elements interacting with the swarm agents. The movements of the type-*I* agents are not driven by any randomness, so that any deviation from the presented structure has to be seen as the result of other external factors. The agent parameter settings are listed in Table 3.

Figure 7(b) displays a scene where the interaction between both flocking and swarm grammar agents is still in progress. The blue pyramidal shapes represent (non-building) agents that organize their flight in a figure eight formation (parameters according to Table 3 and taken from Jacob & Kwong 2004). As a result of the interactions between the building swarm and the flocking swarm, a completely different structure emerges. When one observes this construction during run time, the influence of the swarm grammar agent on the other swarm is fascinating to watch: as long as the swarm grammar agent is present, there is a very high probability of the other flock-mates to interact with it, as the figure eight formation usually occurs around the world center.

5. Swarm Grammar Evolution

We use an extension of *Inspirica* (Kwong 2003), one of our evolutionary design tools, to explore the potential of generating swarm grammar systems that exhibit intriguing constructions. As illustrated in Figure 8, a collection of swarm builder simulations is simultaneously presented to the user. Each window shows the interpretation of different swarm grammar rules and with different agent parameters. All windows display the construction process as it occurs. All designs are true objects in 3D space, hence can be rotated, zoomed and inspected in various ways. After assessment of the presented structures, the swarm designer assigns fitness values between 0 and 10 to each solution.

The rewrite rules and agent parameters are represented as symbolic expressions, so that genetic programming (GP) can be used to evolve both the set of rules as well as any agent attributes (Jacob 2001). For the examples we present here, only context-free rules with a maximum string length of three ($|s| = 3$) are applied. We allow at most five rules per SG-genotype. GP mutation and crossover are the only genetic operators.

As this is our first swarm grammar prototype, the results presented here are still simplistic, but they already reveal the potential of form generation through SG systems. Figures 9 and 10 show selected examples of such evolved structures. As developmental rewrite systems are usually rather sensitive to changes in the genotypes—which can result in vastly different growth structures and developmental processes—we have limited our grammars to only three symbols. In the *Evol-*

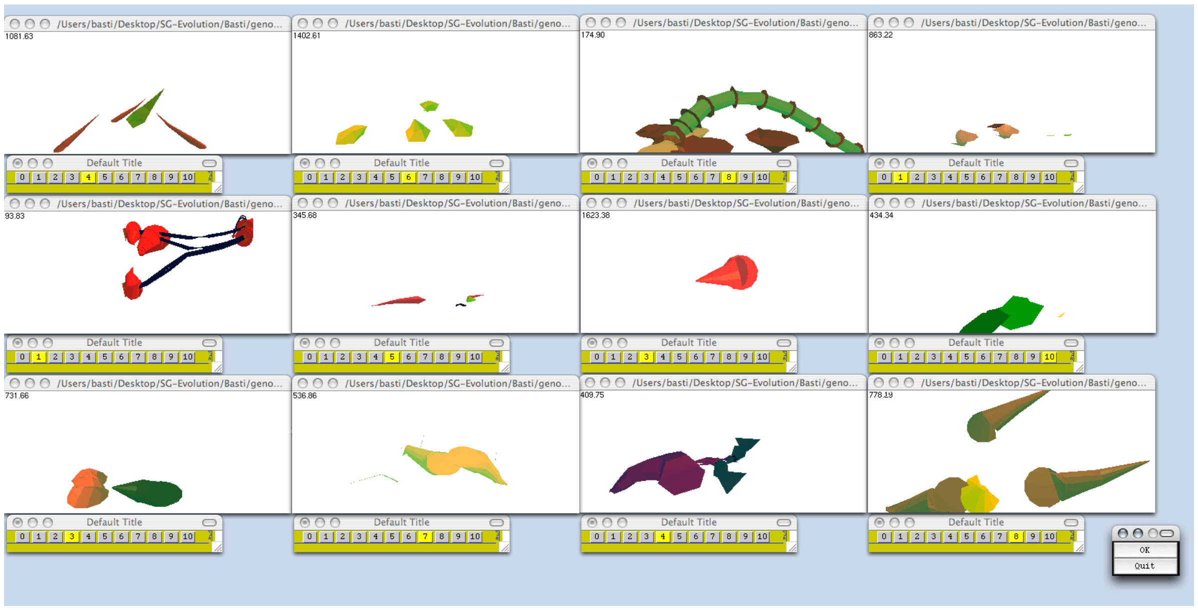


Figure 8. The interactive evolutionary design interface used to explore swarm grammar rules, agent parameters, and their corresponding 3D structures.

vica system (Jacob 2001) we have used filters on typed genetic operators to limit variability on L-system genotypes, which can be applied to swarm grammars as well. However, further investigations will be necessary to explore different encodings and genotype-phenotype mappings for swarm grammars.

6. Discussion

The interpretation of an expanded L-system string by a single turtle has always been one of the major constraints of L-systems. Whereas the rewrite rules are applied in parallel on a single string (i.e., any matching rule is applied), the interpretation of the string by a single turtle serializes the actual drawing or creation process of the 3-dimensional structures. Simulating the growing branches of a tree, for example, this creates major issues as the branch tips are not created in a parallel fashion. Hence, detection of branch collisions and their resolution has to be dealt with after collisions have already occurred (Mech & Prusinkiewicz 1996). Within the swarm grammar approach, these problems do not arise any more, as the swarm agents act as independent, interacting units which resolve collisions on their own. Hence, swarm grammars combine the ease of specification of a grammar system with the interpretive power of a multitude of building devices (extended 'turtles') in 3D spaces.

Organizing sets of swarm agents through deterministic, context-free grammars has enabled us to transfer the notion of connectivity – which is inherent in rewriting systems – onto structures that are created by coordinated movements among swarm agents. The underlying grammar has a profound effect on the resulting topology of the built structures, whereas the employed swarms and their characteristics largely determine the dynamic composition process.

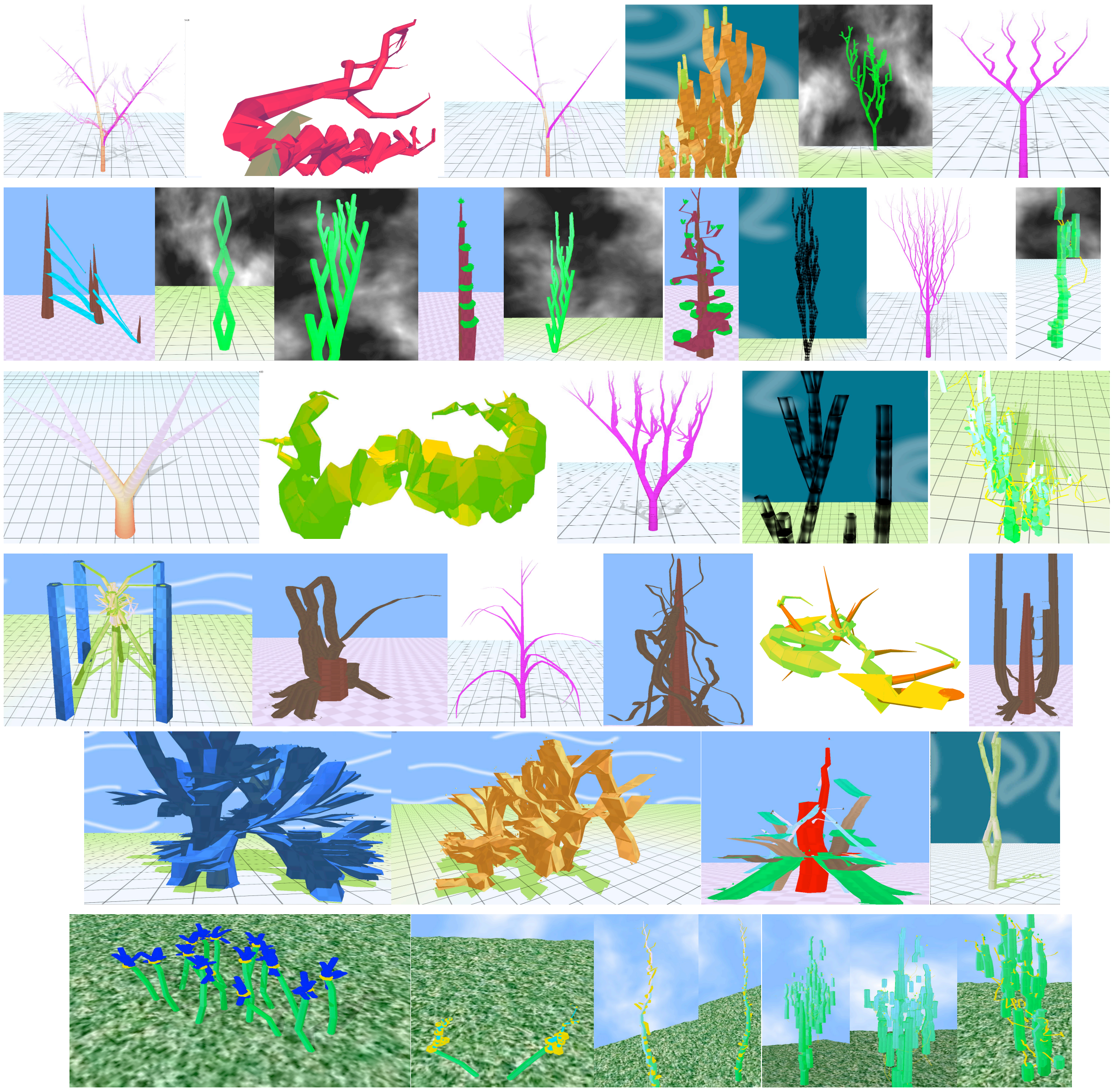


Figure 9. Collage of evolved designs generated from swarm grammar systems.

7. Swarm Grammars: What Next?

There is a wide range of possibilities to extend the proposed swarm grammar approach. Here are a few of these expansions we are currently beginning to explore. As SG-systems are natural extensions of Lindenmayer grammars, all variants of L-systems are applicable as well, such as: context-sensitive production rules, non-deterministic or probabilistic rules, map SG-systems, and table SG-systems. Prusinkiewicz & Lindenmayer (1991) give a good overview of these L-system variants. How far these extended SG-systems will expand the variety of conceivable designs remains to be seen.

Similar arguments apply to the agent side of SG-systems. Not only can interaction parameters be changed, but one may define agents with specific (simulated) physical properties (limited speed, mass, vision, etc), or constrain their interaction spaces (e.g., termites that build nests, but cannot fly). Evolutionary design systems—such as *Evolvica* (Jacob 2001) and *Inspirica* (Kwong 2003)—will certainly help us to unleash the still largely hidden powers of generative, dynamic design through swarm grammar systems.

Software

Sample code of our swarm grammar systems and other swarm-based simulations, which our *Evolutionary & Swarm Design Laboratory* is working on, are available at http://www.swarm-design.org.

Acknowledgement

Our swarm grammar systems are implemented in *BREVE*, a simulation package for modeling decentralized, agent-based systems in 3-dimensional space (Spector & Klein 2002). *BREVE* was designed and is still being further developed by Jon Klein, whom we have to thank for his continuous support and for providing such an excellent research tool for our swarm-based investigations.

References

Alber, R., Rudolph, S., & Kröplin, B. (2002). On Formal Languages in Design Generation and Evolution. 5th World Congress on Computational Mechanics (WCCM V), Vienna, Austria.

Coates, P., Broughton, T., & Jackson, H. (1999). Exploring Three-Dimensional Design Worlds using Lindenmayer Systems and Genetic Programming. In P. Bentley (Ed.), Evolutionary Design by Computers. (pp. 323-341). San Francisco, CA, USA: Morgan Kaufmann.

Haddow, P. C., Tufte, G., & van Remortel, P. (2001). Shrinking the Genotype: L-systems for EHW? Evolvable Systems: From Biology to Hardware: 4th International Conference, Tokyo, Japan.

Hamilton, P. (1994). Computing Dendritic Growth. In R. Paton (Ed.), Computing with Biological Metaphors. (pp. 86-102). London Chapman & Hall.

Hanan, J. S. (1992). Parametric L-systems and their application to the modelling and visualization of plants. Ph.D. Thesis. Department of Computer Science, University of Regina.

Hemberg, M. (2001). GENR8 - A Design Tool for Surface Generation. M.Sc. Engineering Physics Thesis. Department of Physical Resource Theory, MIT, Boston, Ma, USA.

Henry, Kwong (2003). Evolutionary Design of Implicit Surfaces and Swarm Dynamics. MSc Thesis. Department of Computer Science, University of Calgary, Calgary, AB, Canada.

Hornby, G., Lipson, H., & Pollack, J. (2001). Evolution of generative design systems for modular physical robots. IEEE International Conference on Robotics and Automation (ICRA), Seoul, Korea.

Hornby, G. & Pollack, J. B. (2001a). Evolving L-systems to generate virtual creatures. Computers & Graphics, 25, 1041-1048.

Hornby, G., & Pollack, J. (2001b). The advantages of generative grammatical encodings for physical design. Congress on Evolutionary Computation, Seoul, South Korea.

Jackson, H. (2001). Toward a Symbiotic Coevolutionary Approach to Architecture. In D. W. Corne, & P. Bentley (Eds.), Creative Evolutionary Systems. (pp. 299-314). San Francisco, CA, USA: Morgan Kaufmann.

Jacob, C. (1994). Genetic L-System Programming. PPSN III, Parallel Problem Solving from Nature.

Jacob, C. (2001). Illustrating Evolutionary Computation with Mathematica. San Francisco, CA, USA: Morgan Kaufmann.

Kitano, H. (1990). Designing Neural Networks Using Genetic Algorithms with Graph Generation System. Complex Systems, 4(4).

Kwong, H., & Jacob, C. (2003). Evolutionary Exploration of Dynamic Swarm Behaviour. Congress on Evolutionary Computation, Canberra, Australia.

Lindenmayer, A. (1968). Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology, 18, 280-315.

Mech, R., & Prusinkiewicz, P. (1996). Visual models of plants interacting with their environment. 23rd Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA.

Prusinkiewicz, P. (2004). Art and science for life: Designing and growing virtual plants with L-systems. Acta Horticulturae, 630, 15-28.

Prusinkiewicz, P., & Hanan, J. (1989). Lindenmayer Systems, Fractals, and Plants (SIAM/SIREV 33(2)). New York: Springer.

Prusinkiewicz, P., & Lindenmayer, A. (1990). The Algorithmic Beauty of Plants. New York, NY, USA: Springer.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. Int. Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, Anaheim, USA.

Rozenberg, G., & Salomaa, A. (1980). The Mathematical Theory of L-Systems. New York, NY, USA: Academic Press.

Rozenberg, G., Salomaa, A., & Lindenmayer, A. (1986). The Book of L. Berlin ; New York: Springer-Verlag.

Spector, L., & Klein, J. (2002). Evolutionary Dynamics Discovered via Visualization in the BREVE Simulation Environment. 8th International Conference on the Simulation and Synthesis of Living Systems, Sydney, Australia.

Zamir, M. (2001). Arterial Branching within the Confines of Fractal L-System Formalism. The Journal of General Physiology, 118(3), 267-276.

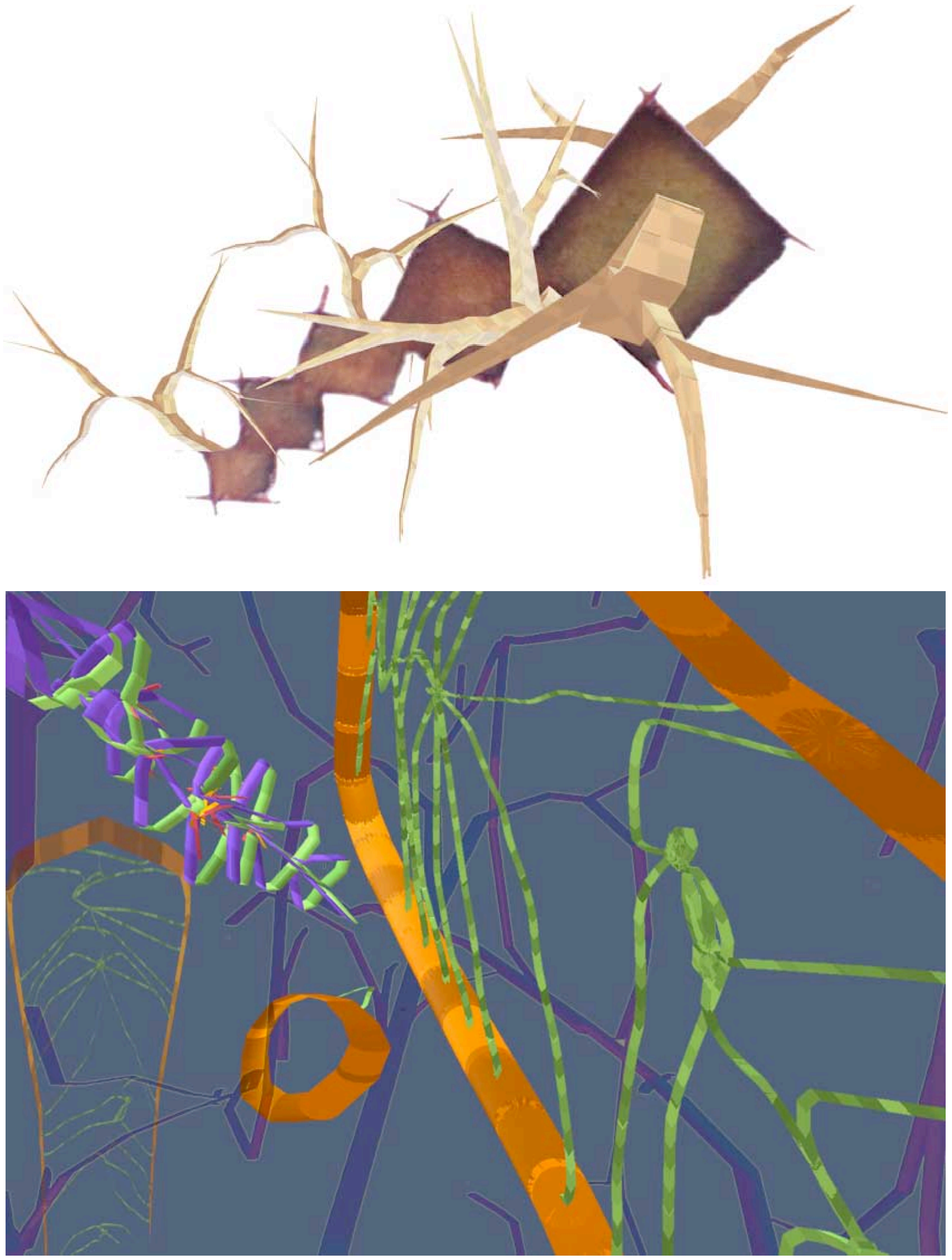


Figure 10. *SwarmGrammar* Art: Each of the two scenes is a composition of sub-structures evolved from different swarm grammars, similar to the ones discussed in the text.