Simulating plant growth

by Marco Grubert

"What a complex matter in its summation, but what a simple one in its graduated steps, the shaping of a tree is."
- Ward

Introduction

The study of plant morphology and plant growth has interested researchers for millennia – not only for survival reasons, but also because of the desire to understand nature and to appreciate the beauty we perceive in natural forms. In the course of history, there has occurred first an abstraction to plant parts, also known as modules [1] (leaves, blossoms, buds, etc.), and later a reduction to cells. However, a universal system for describing changes in plant morphology at the cellular or modular level has yet to be devised.

Computer science in the 1960s was enthusiastic about the formal language theory defined by Church, Backus, and Naur. Out of this enthusiasm for string-rewriting grew classical computer languages like Algol-60. It was also at this time that a biologist, Aristid Lindenmayer, first presented a model for cellular growth using string-rewriting mechanisms [11]. Differing from the Chomsky system of grammars, this formalism, known as Lindenmayer-System or L-System, makes use of parallel replacements. Originally, L-Systems focused on the topological relationships of single cells and larger plant parts. Methods for precise geometric descriptions, necessary for visualization, were formulated later. One of these methods is inspired by the cursor movement commands provided by the LOGO programming language [9]. LOGO, a tutorial language used in high school education, has built-in graphics commands, that make the visualization of state changes easy. The popularity of this approach is derived from its lucid presentation in ground-breaking work published in 1990 by Prusinkiewicz and Lindenmayer [15]. The notation these authors summarized is still being used within most L-System applications.

Many branches of science have contributed to L-Systems research; their specific input can be summarized as follows:

- *Botany:* In botany plants are described as arrangements of functional modules such as buds, flowers,or leaves. Individual plants of the same species are composed of modules that share common traits (e.g. size, color). L-Systems take these modules as their basic units, which change over the course of (simulated) time.
- *Mathematics:* The theory of formal languages deals with grammars consisting of an alphabet of basic symbols and rules (productions) for describing the synthesis of words from symbols. L-Systems are an interesting application of formal language theory. In addition, L-Systems also have connections to fractals.
- *Computer graphics:* In computer graphics, visual models are often described using scene graphs [5]. These graphs consist of various primitives (e.g. lines, triangles, and cylinders) and transformations. Described by displacements and rotations, transformations define the arrangement of all contained primitives. L-System strings are parsed to create scene graphs, which can then be used to display 2D or 3D models.

In this article, I will present a series of L-Systems of varied complexity (complexity governs the expressive power of a system thus the variety of plants that can be created). The article begins with a presentation of deterministic, context-free L-Systems (DOL-Systems). Next, it discusses the interpretation and visualization of the results of L-Systems, strings of characters, as fractals with complex branching structure. Next, the extension of DOL-Systems to context-sensitive and stochastic Lindenmayer-Systems is described. Finally, current applications and alternative models are discussed.

A knowledge of formal languages is advantageous but not necessary to understand this article. Likewise, some subjects (e.g. NURBS, emergence) are mentioned briefly without exhaustive explanation. These subjects are not essential for comprehending L-Systems and explanations are deferred to referenced material.

Deterministic context-free L-Systems (DOL-Systems)

Formally,a simple L-System is a tuple `G=<V,w,P>` consisting of:

```
V : an alphabet,
w∈V + : a non-empty starting word (or
axiom)
P : a set of productions P⊂V x V *
```

Depending on the context, letters of the alphabet could represent cells or modules, e.g., `V={ Leaf, Bud }`. Please note that letters of the alphabet V need not be single characters, but can be arbitrary strings. To avoid any misunderstandings, I will use the term symbol to denote elements of the set `V`.

Productions describe the replacement of symbols by other symbols. This replacement operates on the current word, which initially is the axiom. After all applicable productions have been applied (i.e., all symbols have been replaced), the original word has been transformed into a new word, which will be the current word for the next replacement step. A replacement step (an iteration) can be seen as a discrete time interval, during which development takes place. Usually, productions are written in the form $s \rightarrow X$. This particular production rule specifies that all occurrences of symbol s will be replaced by string X.

In L-Systems, as opposed to Chomsky grammars, it does not make sense to divide the alphabet into terminals (symbols which cannot be replaced) and non-terminals (symbols that can be replaced). It is possible for productions to replace symbols by the empty word λ, which effectively removes that module. It is generally agreed upon that identity replacements need not be specified. Therefore, for any symbol $s \in V$ there exists an implicit production $s \rightarrow s$.

The following is a simple example demonstrating replacement. Given the following specification:

```
V = { Branch , Bud , OldBranch }
w = Bud
Productions :
      1: Bud→Branch Bud
      2: Branch→OldBranch
```

Through the use of productions 1 and 2, the sequence of iteration is as follows:

```
Bud => Branch Bud => OldBranch Branch
Bud => OldBranch OldBranch Branch Bud
=> ...
```

The first replacement invokes production one. The second replacement illustrates a typical feature of L-Systems, namely parallelism. Both symbols (Branch and Bud) have been replaced in a single step. Branch is replaced by production two, and Bud is replaced by production one. This is biologically motivated; in nature, change occurs at different places at the same time. While it might seem that this is only a minor difference from sequential Chomsky grammars, it actually allows L-Systems to create a wider range of languages than comparable Chomsky grammars [15].

Three steps distinguish L-System applications:

1. Parallel replacement of symbols according to the productions.
2. Interpretation of the current word to create a graph.
3. Visualization of this graph if geometric information is encoded in the current word.

Interpretation of L-System words

Interpretation converts the current word to a rooted graph. To interpret an L-System word it is read/parsed from left to right with the left-most symbol representing the root of the structure. Every module is attached to the module on its left. Only linear arrangements of modules can be created, and therefore, every module is connected to at most one descendant.

However, one of the most obvious features of plants is their branching structure. To model branching, it is necessary to attach more than one module to a single parent. Two additional symbols, square brackets, are defined that mark the beginning and end of a branch

$$V' = V \cup \{ [,] \}$$

During parsing these brackets are interpreted as the push and pop operations of a stack machine [2]. A reference to the current node is pushed, when a "[" is encountered. Subsequent nodes are inserted normally until the matching closing bracket "]" is encountered. When a closing bracket is found, the reference is popped off the stack, making the referenced node a parent for the next branch. The string enclosed in corresponding brackets can be interpreted as a complete branch that gets added to the module defined to the left of this string. By specifying a sequence of these enclosed strings, an arbitrary number of branches can be attached to a single module. Of course, it is also possible to use nested brackets as seen in Figure 1, which depicts the following string as a graph:

```
A [+B] [C [D] E F
```

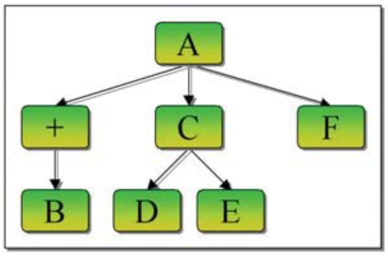


Figure 1 : Graph representation of an L-System word "A [+B] [C [D] E] F"

Visualization of L-System words

So far, we have used L-Systems to create graphs, and in subsequent iterations, we replace nodes in the graph using productions. What is missing, however, is the geometric aspect, which allows the creation of graphical models. Symbols such as the ones chosen above (`"Leaf", "Bud"`) may be easy for humans to interpret, but to a computer, they mean nothing. To remedy this situation, LOGO-style graphics operations are inserted in the L-System as special symbols. LOGO uses the notion of a cursor, also known as turtle, that can be rotated and moved along its current heading. Here are some of the special symbols:

```
F : Move cursor forward by n units and
    draw a line from the last point to
    the current position
f : Move cursor forward by n units,
    but do not draw a line
+ : Turn cursor counter-clockwise by
    alpha degrees
- : Turn cursor clockwise by alpha
    degrees
! : Turn cursor by 180°
```

The stack operations "[" and "]" are extended to push and pop the current cursor position and heading, in addition to the current node. Let the cursor originally be located at a position (0,0) in a Cartesian space, pointing towards the positive y-axis, and set n = 1 unit and alpha = 20°. Then the following L-System will create the (scaled) images depicted in Figure 2.

```
V = { F , + , - , [ , ] }
w = F
1: F F F +[ + F - F - F ] - [ - F + F
   + F ]
```

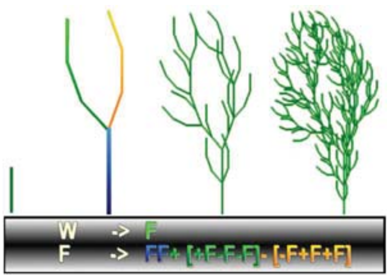


Figure 2 : Iterations 0.3 of a branched L-System. The different branches are color-coded in iteration 1.

Parametric L-Systems

In the above example, it is necessary to define step size and rotation angle outside the L-System specification. If the model uses different angles or step sizes, then it would be necessary to find a common denominator and combine several transformation symbols that create the desired rotation or translation. For example if a structure with angles of 10° and 40° is to be modeled, it would make sense to set alpha = 10° and use the following word (where each + represents a rotation of 10°):

```
 + F ++++ F
```

A better approach is to use parameters that specify these attributes:

```
+(10) F(1) +(40) F(1)
```

Adding parameters to modules not only makes sense for predefined geometric symbols, but for user-defined symbols as well. The meaning of such a user-defined parameter is, of course, up to the L-System designer. Parameters have been used to store the age, size, and intracellular hormone levels of plant parts. To make use of this additional information stored in user-definable parameters, the definition of productions needs to be changed as well. A simple parametric L-System defined in the following manner:

```
V = { Flower }
w = Flower ( 2.0 )
1: Flower ( Timer ) : Timer > 0.0
   →Flower ( Timer-1.0 )
2: Flower ( Timer ) : Timer <= 0.0→ λ
```

On the left side is the symbol name, as well as a formal parameter called Timer. If this formal parameter is found elsewhere in the same production, it will be substituted by the actual value of the module currently under consideration. This kind of parameter substitution has led researchers to interpret productions as function definitions similar to those found in C, Pascal, etc.

The middle part of the production must be a Boolean expression and may contain comparison operators. The Boolean and comparison operators follow C syntax. If the expression evaluates to true, then the module gets replaced by the right side of the production. As can be seen in production 1, it is possible to perform calculations on the right side. For a starting word of `w = Flower( 2.0 )`, the following words will be created by the above productions :

```
Flower( 2.0 ) => Flower( 1.0 ) =>
Flower( 0.0 ) => λ
```

Because no geometric symbols were used, this sequence cannot be readily visualized. If the translation symbol "F", as defined in the visualization section above, had been used instead of "Flower" an animation could be created, in which a line is shortened by one unit per time-step and finally disappears. This is a very simple example, but I hope it is obvious that parametric L-Systems provide a powerful mechanism for creating animations of change in geometry as well as topology.

Context-sensitive L-Systems (IL-Systems)

With context-free L-Systems (OL-Systems), information is exchanged between the replaced module and its replacing module. In contrast, living systems rely heavily on information exchange between neighboring modules. To capture this aspect, context-sensitive L-Systems (IL-Systems) have been introduced. Again, the productions are changed slightly to include constraints on the context of the module to be replaced.

```
V = { A , B , C }
w = A( 5.0 ) B( 0.0 ) C ( 0.0 )
1: A(amount) < B > C : true→B( amount)
2: B(amount) < C : amount > 2.5
→C( amount-2.0 )
```

The productions look a little different than the ones discussed so far. The symbol to be replaced is on the left-hand side enclosed by one or two brackets. These brackets delimit the left and the right context, respectively. The productions will only be applied when the symbol is encountered in the specified context and when the precondition can be evaluated successfully. The second production contains just a single bracket, causing only the left context to be checked for matching strings; the right context will be ignored.

Iterations for the above example:

```
A(5.0) B(0.0) C (0.0) => A(5.0) B(5.0)
C (0.0) => A(5.0) B(5.0) C(3.0) => ...
```

In this example, a value is passed from left to right. In the first iteration, production 1 can be applied because the axiom contains a symbol B, which has symbol A as its left and symbol C as its right context. B is replaced by a new B which takes the value originally stored in A. Note that no production is defined that operates on A, therefore its value remains constant throughout the derivation process. Also in iteration 1, it appears as if the second production could be applied as well; there is a symbol B to the left of symbol C as demanded by production 2. However, the original value stored in B was 0.0, which is below the threshold defined in that production's precondition ("amount > 2.5"). Therefore C can not be replaced. In the second iteration B undergoes an identity replacement because of production 1. This time, production 2 can also be applied. The value of C is calculated based on the value stored in the symbol to the left of it. The second production can be interpreted as a transfer of part of the amount from B to C if it is above a certain threshold (2.5).

This example is not sufficient for modeling realistic information exchange. However, it shows, that it is possible to use the information stored in neighboring modules for decision-making and computation.

Stochastic L-Systems

Lindenmayer-Systems usually are deterministic. Consequently, every time a word is derived using a given system, the resulting words will be the same. Therefore, it can be said that L-Systems describe individual plants, not plant species. This may not always be desired. Imagine visualizing a complete field of flowers. If only one L-System was used to describe every flower, then the results would look unrealistic. On the other hand, creating a single L-System with individual productions for each plant would be a tedious task, and it still could not guarantee similarity between individuals.

What can be done instead is to use probabilistic/stochastic L-Systems. These provide alternative productions with the same left-hand side, but different right–hand sides, where each of these alternatives is assigned a probability. This probability is written on top or in front of the

$$
\begin{array}{lcl}
V & = & \{\ F,\ +,\ -,\ [,\ ]\ \} \\
w & = & F \\
\\
1.\ F & \xrightarrow{0.33} & F\ [+F]\ F\ [-F]\ F \\
2.\ F & \xrightarrow{0.33} & F\ [+F]\ F \\
3.\ F & \xrightarrow{0.34} & F\ [-F]
\end{array}
$$

Figure 3: A stochastic L-System. Please note the production probabilities on top of the arrows.

arrow symbol. If during derivation a situation is encountered where more than one production would match, a random number generator is employed to select one of them based on their assigned probability. The example below and Figure 4 show how different results can be created from the same L-System specification.

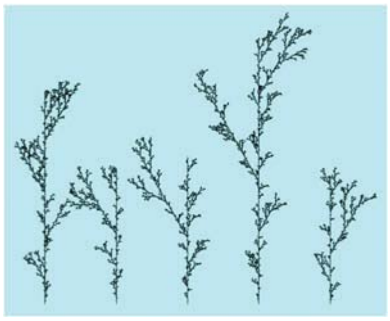


Figure 4 : A sample group created by a stochastic L-System. Even though every individual is different, they seem to belong to the same species.

Deterministic productions can be seen as special cases of probabilistic productions, where the probability equals 1. This observation can be used to create L-System specifications for a plant species: those developmental processes that occur in every individual of the plant are encoded in productions with a probability of one. Processes that might be different for each individual are specified using alternative productions with probabilities less than one.

The results generated through Stochastic L-Systems are different for every derivation process. Sometimes, however, it might be desirable to reproduce a certain derivation. This can be done using a simple trick - as most random number generators are not really random, but rather are based on a seemingly random series of numbers, it is sufficient to note the starting number (also called the seed) and use this starting number to later on reproduce the same results.

Remarks

One phenomenon that never ceases to amaze even the most seasoned L-System designer is known as emergence [10]. This term refers to the creation of complex structures from simple starting data. Similar effects can be observed in the theory of fractals and are of great importance to the research of Artificial Life.

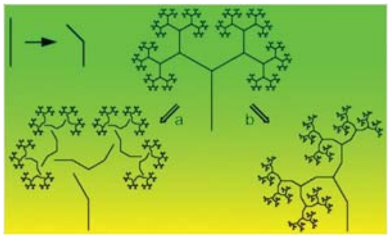


Figure 5 : A comparison of the Koch construction (a) with a rewriting system preserving the branching topology of the modeled structures (b). The same production is applied in both cases, but the rules for incorporating the successor into the structure are different. (From [17])

L-Systems are also closely related to Koch curves, such as the famous snowflake curve. However, there are subtle differences in connectedness. A Koch curve is a set of oriented line segments with no topology information. L-System words, on the other hand, do contain topology information. Consequently, every module is dependent on its parent. This is illustrated in Figure 5.

Using the bracketed string notation introduced above, a graph can be mapped to a number of different strings [18]. Therefore, mapping is not unique. Imagine a parent module A with three children B, C, D. Valid strings for this graph would be A[B][C]D, A[B][C][D], A[C][B]D, and so on. This causes problems with context-sensitive L-Systems when string notation is used for comparisons. Because different words can give rise to the same graphs, it would be necessary to create a large amount of productions for a task that can easily be stated as "Replace every module A which has a module B as its child." One possible solution, incorporated into the system "L-arbor" [8], is to do comparisons based on the actual (sub-) graph and not the string notation.

Current work

As can be seen from the previous discussion, defining a working L-System is simple; however, generating an L-System that closely resembles existing plants is a difficult task. This difficulty is twofold; first, there is the problem of emergence, and second there is the problem of non-intuitive representation.

When modeling a given plant, one knows the desired outcome and the intermediate steps, and so one can try to create matching L-Systems. However, emergence (related to database-amplification [17]) prevents easily estimating the outcomes of these. In terms of formal languages, the problem can be described as finding a language that creates a number of words in a certain order.

Because 3D structures are described using a 1D string notation, it is often difficult to see what a string creates. To overcome this problem of non-intuitive representation, a few suggestions have been made. The "L-arbor" integrated development environment [8], for example, allows specifying the right-hand side of productions using a 3D editor, in order for complex syntax to be avoided. Similarly, a graphical editor is provided for leaf specification using NURBS [5]. NURBS (Non-Uniform Rational B-Splines) are a method for defining curves by specifying a small number of control points, which are then interpolated to form a smooth curve or surface. Moreover, only visual aspects of an L-System can be defined using this approach. In practice, however, one can often find L-System specifications where a module is initially invisible and only later on gets converted to a visible module. Editing productions in graph notation instead of string notation is a similar approach that allows specifying non-visual parts of a production in an easier way.

This article has focused on L-Systems. However, there are other plant generation mechanisms. De Reffye [3] presented an influential biologically inspired model for procedural plant generation. In his approach, probabilities for certain plant events (bud ramification, mortality, growth direction, etc.) are assigned to groups of branches. Therefore, by specifying a small number of parameters, a wide variety of different species can be modeled. A recent enhancement to this modeling paradigm using visual editing and growth control has been implemented by Lintermann and Deussen [12]. In general, the ease of design found in procedural plant generation is achieved at the cost of having full control over the model, since the only attributes that can be simulated are those provided by the simulation framework. On the other hand, L-Systems are much more general and can be used to produce mathematical models as well. More precisely, the recursive nature, which is essential for fractal models, is lacking in the procedural approach.

Plants do not exist in a void, but in an environment that can support or hinder development. Tendrils, for example, are dependent on obstructions in order to grow upwards. In general, living organisms are embedded in an ecosystem, which affects organisms and is affected by them. Query modules, a general-purpose mechanism for interacting with other processes, have been introduced in [13]. These modules exchange information with user-definable programs via parametric symbols. An L-System word is passed to the called application, which reads the necessary data from it and returns data by replacing some of these parameters. Query modules have also been used to retrieve absolute coordinates and orientation. Here, three parameters are used to retrieve a direction or position vector from the L-System simulator. Production preconditions then check for specific values and act accordingly (e.g. by avoiding an obstacle). Visually impressive results of a topiary garden are reproduced in Figure 6. While it is nice that these enhancements work in the formalism of L-Systems, defining complex environments using equations in preconditions is out of the question. For this reason, I have used a slightly different approach for "L-arbor": a standard 3D polygon file is loaded and used as bounding volume. Should any collisions between model and bounding volume occur then a single parameter is set which can be queried using preconditions. Greene has pointed out the computational advantages of operating in voxel space for quickly checking for obstructions as well as determining direct lighting [5].

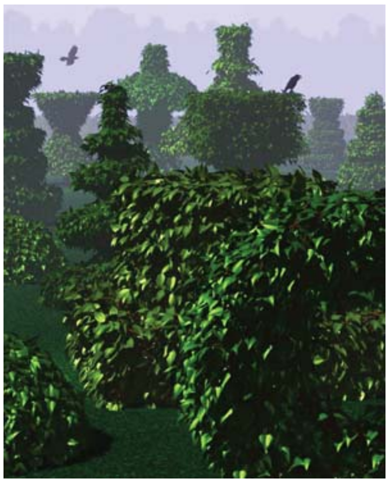


Figure 6 : A synthetic model of the topiary garden at Levens Hall, England (from [17])

Finally, when animations are constructed, another problem occurs that requires our attention. L-Systems work with discrete time steps. While these time steps can be chosen to be arbitrarily small, once they are set, it is hard to change them. More importantly, depending on the magnification, during animations it is possible to see new modules suddenly popping up or greatly changing their appearance. A solution described in [16] has been suggested by Prusinkiewicz et al. They propose that instead of discrete changes in parameters, differential equations can be used for parameter changes. Geometrical changes then take place in continuous time, while topological changes are discrete events that work like ordinary L-Systems. This way, a combined discrete-continuous specification is created, which when properly designed eliminates the visual artifacts just mentioned [4]. These differential L-Systems (dL-Systems) have been implemented in "cpfg" [14] and "L-arbor" [8]. There are some subtle points that need to be considered when designing dL-Systems, but they are beyond our scope. The interested reader is referred to [16] and [7] for more information.

Conclusion

The notion and notation of L-Systems has undergone a number of changes, from a purely theoretical construct for describing cellular growth to a versatile tool for creating realistic models of organic structures. This has been made possible through continuously increasing the power of expression and advances made in the field of formal languages in general. However, it is becoming obvious that the notation used for L-Systems is powerful, but difficult to learn and to master. Furthermore, the limitations of old-fashioned design considerations (e.g., L-Systems use 7-bit ASCII characters) seem to hamper great advancements. For this reason, two different approaches have emerged. The first continues the old tradition and provides support for mathematical modeling and realistic simulations, which require control over individual modules. The other one is concerned with ease-of-use and visual similarity between model and reality only. Is there a way to reunite these approaches, allowing for powerful simulations and ease-of-use at the same time?

From a philosophical point of view, one has to ask whether modeling growth using atomic units (modules or cells) makes sense at all. The inclusion of real-valued parameters and differential equations shows that at least the attributes of plant parts cannot be split into distinct units. Perhaps a completely different model should be employed, one that is based on the notion of a continuous structure with continuous parameters, instead of the current model that represents a discrete structure with continuous parameters.

The greatest challenge for future plant simulations, however, will be to consistently integrate environmental factors, such as surrounding plants, obstructions, water and mineral availability, and lighting conditions. Changes in resource availability influence plant growth, which in turn results in a change of resource availability. Powerful models and powerful hardware will be necessary to effectively simulate these recursive interactions of recursive structures.

References

1 Bell. *Illustrierte Morphologie der Blütenpflanzen.* Ulmer, Stuttgart, 1994.

2 Claus, Schwill. *Schülerduden-Die Informatik.* Dudenverlag, Mannheim, 2nd edition, 1991.

3 De Reffye. Plant models faithful to botanical structure and development. *Proceedings of SIGGRAPH 1988*, in Computer Graphics 22, 4, ACM SIGGRAPH, New York, 1988, pp. 151-158.

4 Fahrland. Combined discrete event – continuous systems simulation. *Simulation*, 14- #2, 1970.

5 Foley, van Dam, Feiner, Hughes. *Computer Graphics: Principles and Practice (2nd Ed.).* Addison-Wesley, 1990.

6 Greene. Voxel space automata. Modeling with stochastic growth processes in voxel space. *Proceedings of SIGGRAPH 1989* in Computer Graphics 23, ACM SIGGRAPH, New York, 1989.

7 Grubert. *Graphische Spezifikation von erweiterten differenziellen Lindenmayer-Systemen*. Master's Thesis, FB Informatik, Technical University of Berlin, 2000.

8 Grubert. L-arbor, 11 May 2001, http://www.geocities.com/grubertm/id34.htm (11 May 2001).

9 Harvey. *Computer Science Logo Style Vol. I (2nd Ed.).* MIT Press, 1997.

10 Krieger. *Einführung in die allgemeine Systemtheorie.* Fink Verlag, Munich, 1996.

11 Lindenmayer. Mathematical models for cellular interaction in development I+II. *Journal of Theoretical Biology,* 1968.

12 Lintermann, Deussen. Interactive Modeling of Plants, *IEEE Computer Graphics and Applications,* January/February 1999.

13 Mêch, Prusinkiewicz. Visual Models of Plants Interacting with Their Environments. *Computer Graphics Proceedings*, Annual Conference Series, 1996.

14 Mêch. *CPFG Version 3.4 User's Manual*. Department of Computer Sciences, University of Calgary, 1998.

15 Prusinkiewicz, Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990.

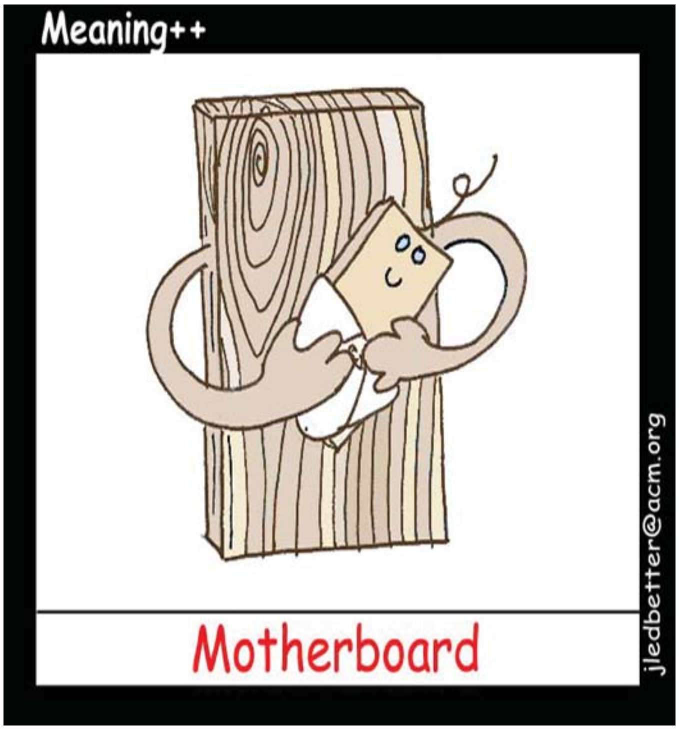
16 Prusinkiewicz, Hammel, Mjolsness. Animation of Plant Development. *Computer Graphics Proceedings,* Annual Conference Series, 1993.

17 Prusinkiewicz, Hammel, Mêch, Hanan. The Artificial Life of Plants. Artificial Life for Graphics, Animation and Virtual Reality, *SIGGRAPH '95 Course Notes,* 1995.

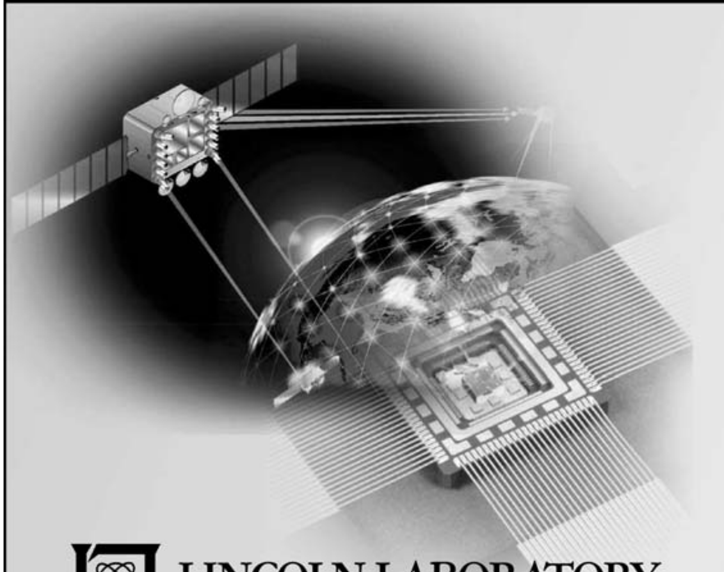
18 Prusinkiewicz, Hanan, Mech. *An L-System based plant modeing language.* Technical report, University of Calgary, 1999.

Biography

Marco Grubert (grubertm@hotmail.com) has recently received an MSc degree from Technical University of Berlin, Germany with Computer Sciences as major and Philosophy as minor subject. His main interests are VR, physical simulations, and AI. Currently he is looking for employment in the real world before returning to academia for his doctoral studies which will be related to 3D graphics in some way.



DISCOVER **the power** to *innovate*



If you are pursuing a degree in:

- **Electrical Engineering**
- **Physics**
- **Mathematics**
- **Computer Science**
- **Mechanical Engineering**
- **Aeronautics/Astronautics**

maybe you have the power to succeed at Lincoln Laboratory.

For more information about our scheduled on-campus recruiting activities, please visit our website or your Career Services Office.

Put some power behind your ideas. At MIT Lincoln Laboratory, your research and ideas will be transferred directly into the development of leading-edge, real-world applications, including air defense, space surveillance, communications and air traffic control systems.

If you are looking for a way to power up your career, forward your scannable resume by e-mail, fax or mail to: resume@ll.mit.edu; Fax: 781-981-2011; MIT Lincoln Laboratory, Human Resources, 244 Wood St., Lexington, MA 02420-9108. Job Code CRF01

An Equal Opportunity Employer, M/F/D/V.
US Citizenship is required.

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

COLLEGE RECRUITMENT PROGRAM

w w w . l l . m i t . e d u