The Language LinF for Fractal Specification

Fernando M. Q. Pereira Leonardo T. Rolla Cristiano G. Rezende Rodrigo L. Carceroni

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627, Pampulha – Belo Horizonte, MG, CEP 31270-010, Brazil

{*fernandm, leorolla, rezende, carceron*}*@dcc.ufmg.br*

Abstract

This article presents the language LinF for L-System specification. L-Systems are formal string rewriting systems introduced in 1968 by the botanist Aristid Lindenmayer that are used to model fractal images. LinF allows the definition of three-dimensional fractals and stochastic fractals. Together with the LinF definition this paper presents the implementation of cFLC, an OpenGL-based system that generates fractal images from LinF specifications. Results obtained through the LinF formalism show the ease of use and generality of the developed tools with respect to the existing literature.

1 Introduction

The Euclidean geometry, traditionally used to represent smooth surfaces and regular shapes that can be described with polynomial equations, fails to represent realistically some forms like mountains and clouds because they have irregular or fragmented features [10]. Natural objects can be described more satisfactorily with fractal-geometry methods, where iteration of functions or feedback processing is used to model shapes.

According to Benoit Mandelbrot [18], a fractal is by definition a set for which the Hausdorff-Besicovitch dimension exceeds the topological dimension. For a more detailed discussion about fractal dimension, see [7]. Fractal objects have two basic characteristics: infinite detail at every point and a certain self-similarity degree between the object parts and its overall features.

A large class of fractal structures can be generated by a formalism called L-System, which is based in string rewriting. This formalism, introduced by the biologist Aristid Lindenmayer in 1968 [16], can be used to model not only certain types of fractals but also an enormous variety of evolving systems such as fungi colonies [34] and musical compositions [30].

Although the basic ideas of rewriting systems date from the beginning of the last century, they were not much used until the 1950s, when Noam Chomsky [3] applied them to describe the syntax of natural languages. A key difference between Lindenmayer's and Chomsky's uses of rewriting systems is the way in which the rules are applied. While in Chomsky's formalism the application is sequential, in Lindenmayer's the rules are applied simultaneously.

This article presents a language suitable for L-System specification called LinF (acronym for the Portuguese translation of *Fractal Language*). While LinF incorporates the main characteristics of the notation first introduced by Lindenmayer, it has additional aspects that allow the definition of stochastic fractals in a three-dimensional space. A graphical system for displaying fractals defined by LinF has been implemented and results obtained with it will be presented.

The major contribution of LinF relative to other fractal specification languages found in the literature is that it borrows concepts from the differential geometry of curves to specify 3D rotations in an elegant and compact way, using an intrinsic local coordinate system for each element of the fractal, which makes the construction of self-similar 3D objects considerably more concise and intuitive than in existing tools. A secondary contribution of LinF, which can not be regarded as a conceptual innovation as the one above – but nonetheless makes the specification of complex fractal scenes easier – is that it allows randomness to be added to fractal objects in a way that is slightly more general than those used in most existing tools. For a detailed discussion on these contributions, we refer the reader to Section 5.

2 A short explanation about L-Systems

L-Systems have been used to describe a wide range of growing systems such as plants [29, 17, 22] and other living beings [12], human organs [13, 5], feathers [2], terrain [20, 28, 22] and cities [23]. Graphical representations of L-Systems were first published in 1974 by Frijters and Lindenmayer [8], and by Hogeweg and Hes-

per [11]. The potential of L-Systems for producing realistic images of plants was demonstrated in 1978 by Smith [33]. In 1979 Szilard and Quinton [35] showed that L-Systems can generate fractal curves. In 1983 Siromoney and Subramanian [32] used L-Systems to generate space-filling curves. In 1982 Dekking [6] found the dimension of some curves generated by L-Systems. In 1986 Prusinkiewicz [25] produced more examples of fractals and plants using L-Systems. Also, he obtained three-dimensional versions of L-Systems.

A number of variations to L-Systems have been introduced, including *Stochastic L-Systems*, where irregularity is obtained by assigning probabilities to the production rules. Another important improvement was the introduction of *Parametric L-Systems*. In this case, any rule can be associated with a small set of parameters that are used to change the interpretation of the rule along successive iterations.

The basic idea is to represent fractal shapes as strings and to define a set of rules that will determine how the strings will evolve in order to generate more complex figures. Every L-System contains two essential parts: an *axiom* and a set of *rules*. The axiom is the initial string that defines a valid fractal. Often the axiom is the smallest fractal string. Each rule in the set of rules is defined by two strings known as the *left side* and the *right side*. During an application of the rules on the string description of the fractal, every left-side symbol in the string is substituted by its corresponding right counterpart in the rule. All rules are applied on the string in parallel during an expansion iteration.

Every symbol in the string that describes a fractal has a particular meaning. For instance, it can be defined that the character `f` will cause the drawing of a line along a specified direction, with pre-determined length and color. This kind of string interpretation is called the turtle interpretation [1], after Szilard and Quinton. For instance, in Figure 1 a simple L-System is shown along with the sequence of pictures it generates. In that L-System, the character `f` causes the drawing of a line segment, and the characters '$-$' and '$+$' rotate drawing direction by 60 degrees clockwise and counterclockwise, respectively.

3 The LinF specification

LinF is a simple language used to describe 3D L-Systems to be interpreted according to the turtle method. In order to model complex living bodies, LinF provides to the user a set of primitives for defining stochastic L-Systems. A program in LinF is formed by six blocks, none of which is mandatory. However, it is not possible to generate meaningful fractals if some of them are missing. These six blocks are called *lines*, *blanks*, *angles*, *axiom*, *rules* and *colors*. There is no order in which the blocks should appear in a LinF program. A short description of each block is given next:

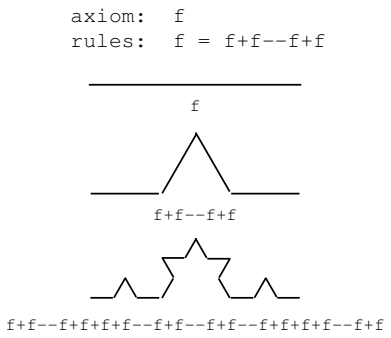

Figure 1. A simple L-System and its three initial renderings.

lines Defines which symbols will represent lines in the production rules. When evaluating the production rules, the lines should be drawn.

blanks Defines which symbols will represent blanks or transparent lines in the context of a particular LinF program.

angles In this section we define the rotation angles that can appear in the fractal description. A rotation angle is constituted by two elements. The first component of a rotation specifies the angle of torsion and the second one specifies the curvature. The rotation scheme is fully explained in Section 3.3.

axiom Defines the axiom of the L-System. An axiom is the starting point from which the fractal will be generated. In Figure 1 the string that represents the fractal, in its most basic form, is given by the symbol `f`, which is the L-System axiom.

rules This module of a LinF program describes every production rule that forms the fractal. A production rule is formed by two parts: the left side and the right side. The left side is simply an identifier, while the right side is an arbitrary sequence of expressions. As in Chomsky's formalism, the grammar rules in LinF are constituted by *terminal* and *non-terminal* symbols. Every symbol that appears in the left side of a production rule is considered a non-terminal symbol. Some of the non-terminal symbols do not have any meaning for the fractal representation and are used just to facilitate the grammar specification.

colors In this section the fractal designer can specify a list of colors that will be used to draw the fractal. The list will be stored in order of specification within the LinF file. The first lines of the fractal will have the first color in the list. When the '$\{$' symbol is encountered in the fractal string, the next color becomes the current one. If the symbol '$\}$' is encountered, the previous color

becomes active again. If a number of '{' exceeding the number of colors stored in the color list is found, then the last color recorded becomes the current color.

3.1 The LinF interpreter

Before understanding how fractals are represented with the LinF formalism, it is worthwhile to comprehend the nature of a basic LinF interpreter. The LinF interpreter is a program that generates fractal figures based on the LinF description of the fractal structure. In this section we will describe the structure of cFLC (*c-Based Fractal Language Compiler*), a simple LinF interpreter developed in C, whose schematic view can be seem in Figure 2:

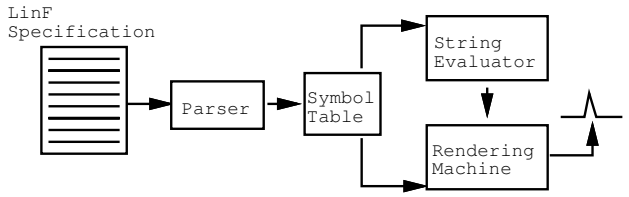

Figure 2. Schematic view of cFLC [24].

The LinF interpreter can be divided into three main constituents: *the parser*, *the evaluator* and *the rendering machine*. The parser is responsible for recognizing the correctness of a LinF file and retrieving from it the meaning of the user-defined symbols, for instance, the name of lines, blanks and angles.

The next component of cFLC is the evaluator. This program is responsible for applying the production rules over the axiom of the fractal and then over the successive strings obtained by the application of the rules. A production rule is formed by two parts. The left-side symbol and the right-side sequence of symbols. To apply a rule over a string traditionally means to substitute every occurrence of its left side that appears on the string for its right side.

In addition to the simple scheme for defining rules, LinF permits the binding of some rules to a probabilistic chance of occurrence. This means that during the string evaluation, some symbols that match the left side of a production rule may not be expanded. This pattern of evaluation allows the creation of stochastic L-Systems that are useful to model real entities such as rivers and plants.

The last constituent of cFLC is called the rendering machine. The rendering machine is responsible for displaying the fractal that is represented by a string. In order to implement this part of cFLC, the OpenGL [21] library was used. This machine has always a *state* that is represented by the tuple (position, frame, length, color). During the rendering process, the state is continuously being updated while the segments that constitute the fractal are being drawn according to the state elements. The meaning of each one of the four state elements is described below:

position This element specifies a point in 3D space. The next line to be drawn will be displayed as a segment starting at this point.

frame This element is composed by a set of three orthonormal vectors $\{t, n, b\}$, whose meaning is explained in Section 3.3. The next line to be drawn will be exhibited in the direction of t.

length This element defines the length of the next segment to be drawn.

color This element specifies the color of the next segment to be drawn.

The LinF interpreter keeps processing the fractal string in a circular fashion in which firstly the string is expanded by the evaluator and then it is read by the rendering machine, which uses it to display a graphical representation of the fractal in a window. The string that represents a fractal contains symbols that force updates in the rendering machine state during the evaluation process. The semantics of each such symbol is explained in the Section 3.2.

3.2 Representation of fractals in LinF

LinF, as an L-System-based formalism, allows fractals to be described by strings that contain finite sequences of symbols. These symbols can be terminals, nonterminals, numbers and reserved words. LinF has nine reserved symbols that are explained below:

[Saves the current machine state in a stack.

] Restores the machine state to the state stored in the top of the stack and pops the stack.

@ Multiplies the current line length by the number previously seen. This symbol is always preceded by a floating point number that specifies the amount used to multiply the current length.

? This symbol does not have any utility during the drawing process. It is useful only during the process of rule application. It means that the next symbol has a chance of not being expanded by the rule in which it appears on the left side. It should always be preceded by a floating point number in the interval $(0, 1)$, which specifies the probability of expansion.

+ Updates the state with a rotation given by an angle specified. This angle will be given by the identifier that follows a sequence of + signs. Note that more than a + sign can be encountered together. It is possible to declare a default rotation in the angle section. In this case, if a sequence of + is not followed by any angle identifier, the default rotation must be assumed.

– This symbol has the same meaning of +, except that it causes a rotation by minus the specified angle.

{ This symbol forces a change in the color state. When a '{' is found, the current color is replaced by the next color in the list of colors. If the current color is already the last element of the list, then no modification is carried on.

} This symbol restores the color state to the previous color.

! This symbol causes an inversion in the interpretation of the symbols + and –.

3.3 Rotations in 3D

The concept of *rotation* for three-dimensional lines is not trivial and devising a representation for it is not straightforward. For instance, consider the quotient group $(\mathbb{R}/(2\pi), +)$, the set of angles with the sum operation. This group is isomorphic to the group $(\mathrm{Rot}(\mathbb{R}^2), \circ)$, the set of rotations in $\mathbb{R}^2$ with the composition operation. In other words, the set $\mathbb{R} \bmod 2\pi$ may be thought as equivalent to the set of rotations in $\mathbb{R}^2$, and summing elements of the first set is analogous to composing elements of the second. Thus, in the two-dimensional case, we have one degree of freedom to specify any rotation and there is no need to specify the "rotation direction". All curves are assumed to rotate around the z-axis.

In order to specify objects in three dimensions, however, this "direction" is needed, and we want to specify transformations on it in a way that depends only on local properties of the segment being drawn, never on the *absolute reference frame*. Thus we propose a scheme analog to the *Frenet formulas* [36] from differential geometry, except that our scheme is discrete.

A brief review of differential geometry follows. Given a curve in $\mathbb{R}^3$ we assign, for each point, a set of three orthogonal unit vectors. Any regular parametric curve can be rewritten as $\alpha(s) : \mathbb{R} \to \mathbb{R}^3$ such that $|\alpha'(s)| = 1$. Let $t(s) = \alpha'(s)$, $n(s) = \alpha''(s)/|\alpha''(s)|$, $b(s) = t(s) \times n(s)$. The set $\{t, n, b\}$ is the *Frenet reference frame*, and the vectors are called the *tangent*, *normal* and *binormal* vectors, respectively.

The rate of change of the *frame* can be expressed in terms of the vectors themselves by the *Frenet formulas*:

$$\begin{aligned} t'(s) &= k(s)n(s) \\ n'(s) &= -k(s)t(s) - \tau(s)b(s) \\ b'(s) &= \tau(s)n(s) \end{aligned} \qquad (1)$$

In the Equation (1), $k(s)$ and $\tau(s)$ are, respectively, the curve's curvature and torsion. Vector t points to the direction the curve is *moving*, n points to the direction the curve is *turning* and b completes the orthonormal base with positive orientation.

The *curvature* k represents the rate at which the curve is 'turning', and for a circle, k is equal to the reciprocal of the radius. It makes the tangent vector turn towards the normal vector, whereas the normal vector turns toward the negative of the tangent vector. The *torsion* τ represents the rate of change of the curve's "turning direction", making the binormal vector turn toward the normal vector, whereas the normal vector turns towards the negative of the binormal vector.

The great advantage of this approach is that the relationship expressed in Equation (1) is completely independent of the xyz reference frame, and there is no need to specify a "rotation direction" explicitly. This scheme motivates the definition of rotations within LinF because it makes it simple to specify recursive definitions of self-similar L-Systems. In the LinF formalism, a rotation will be a pair of numbers, which we can identify as torsion and curvature. A 'torsion' of θ will cause

$$\begin{bmatrix} b^T \\ n^T \end{bmatrix} \leftarrow \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} b^T \\ n^T \end{bmatrix}. \qquad (2)$$

Similarly, a 'curvature' of θ will do the same with (t, n) instead of (b, n).

4 Using LinF to generate fractal images

The LinF language can be used to specify a great variety of L-Systems, in two-dimensional and three-dimensional spaces. As explained in Section 3, the L-Systems defined with LinF can be deterministic or stochastic. In this section some examples of images generated by cFLC – the LinF interpreter – will be shown.

4.1 Classic fractals

Two-dimensional, completely deterministic L-Systems are the most basic kind of grammar that can be specified with LinF. In spite of being simple, this class of L-Systems can be used to describe a wide variety of shapes such as plants, tilings and space-filling curves. The classic fractals known as *Koch's snowflake* and *Sierpinski's carpet* are displayed in Figures 3 and 4, along with their LinF grammars.

```
lines f

angles (0, pi/3)

axiom f++f++f

rules f = f-f++f-f
```

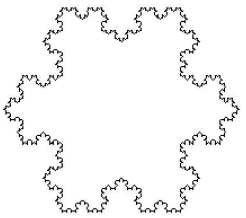

Figure 3. Koch's snowflake in LinF / cFLC.

```
lines f

blanks g

angles (0, pi/2)

axiom f

rules
f = f+f-f-f-g+f+f+f-f;
g = g g g
```

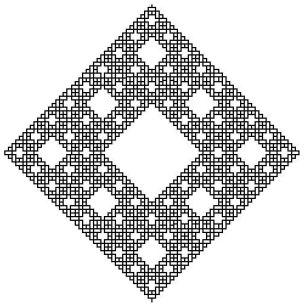

Figure 4. Sierpinski's carpet in LinF / cFLC.

In 1890 Peano defined a curve that is continuous and goes through every point of a square (Figure 5). Peano's curve allows a point to be specified by a single number, its distance from the end of the curve. The definition offractal dimension as the number of variables required to specify a point became untenable. The crisis ended in 1922 when Besicovitch gave the final form to what is now called the Hausdorff-Besicovitch dimension [18].

```
lines f

angles (0, pi/2)

axiom e f

rules
f = f-f+f+f+f-f-f-f+f;
e = .333@ e
```

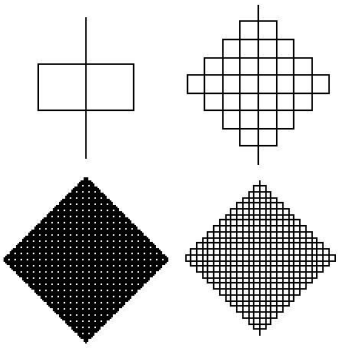

Figure 5. Peano's space-filling curve in LinF and four of its renderings in cFLC.

L-Systems were devised originally to represent the growth of living beings such as plants and fungi colonies [16]. Very accurate simulations using simple sets of rules have been obtained since the presentation of the formalism in 1968 [27, 31]. A short example of plant simulation by means of L-System is given in Figure 6.

The brackets in the LinF specification are used in order to create the branches in the plant. Ordinarily, branches tend to be older near the base, that is, the extremities tend to have smaller sizes than the parts of the plant close to its base. This natural behavior was simulated using length reductors while drawing the branches in Figure 6. As the string representation of the fractal is expanded, more and more chains of reductor factors are encountered, contributing to make a very realistic organism.

```
lines f, g

angles (0, pi/25)

axiom f


rules
f = g [+++++++++ .5@ f] -g [--------- .4@ f] .6@ f
```

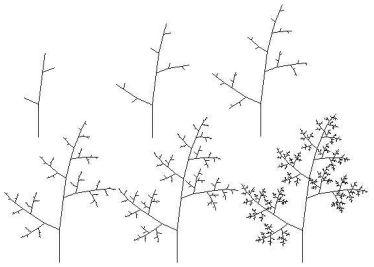

Figure 6. 2D plant simulation in LinF / cFLC.

4.2 Three-dimensional objects

In order to model even more realistic shapes LinF allows the specification of three-dimensional figures. As explained in Section 3, drawing in three-dimensional space can be accomplished by defining rotations with non-zero torsion components. In order to provide a better visualization of the generated images, cFLC permits that any visualization point be specified by the user while displaying the fractals.

In Figure 7 there are three different views of the same rendered fractal. In each iteration of the string evaluator, any stem of the plant evolves to a new branch containing six secondary stems that arise from its main axis. In order to generate non-collinear sequences of branches, the angles t and c are combined in a way that every new subdivision of a stem is $\pi/3$ radians distant from the nearest branches.

```
lines f, g

angles
a: (pi/3, 0)
b: (0, .3)

axiom
5@ f

rules
f = .5@ g h;
h = {i+a i+a i+a
     i+a i+a i};
i = [+b f]

colors
0: (000,000,000) 1: (040,040,040) 2: (080,080,080)
3: (120,120,120) 4: (160,160,160) 5: (190,190,190)
6: (210,210,210) 7: (220,220,220) 8: (230,230,230)
```

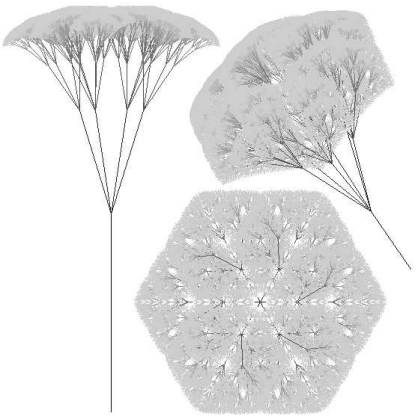

Figure 7. Three views of a single rendering of a LinF-specified 3D plant.

In the plant shown in Figure 8 there are ten main branches. Six of them perform a 120-degree rotation relative to the trunk of the tree. The other four branches have a more accentuated inclination in relation to the trunk: 135 degrees. Every branch in the tree, after the next iteration of the string iterator, will evolve to a new smaller tree that follows the same pattern described above. The color table is

omitted in this and in all the following figures due to space restrictions.

```
lines f

angles
a: (pi/3, 0) b: (0, pi/3)
c: (pi/2, 0) d: (0, pi/4)

axiom 2@ f

rules
f = f [g g g g g g] [h h h h] i;
g = +a [{+b .5@ f}];
h = +c [{+d .5@ f}];
i = [{.5@ f}]
```

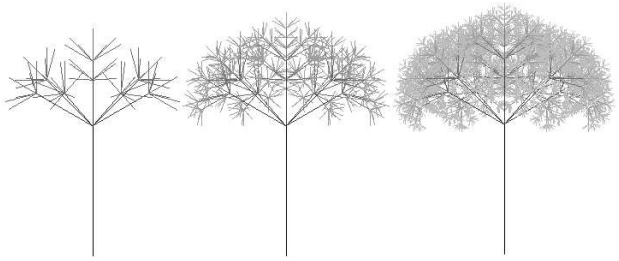

Figure 8. 3D fractal specification in LinF and three renderings with cFLC.

4.3 Stochastic fractals

The L-Systems examined in Section 4.1 and Section 4.2 are formed by deterministic rules, hence the fractals generated by them will always have the same structure. However that is not the best approach when modeling living bodies such as plants or bacteria. Natural processes, in general, are defined by very complex rules. Simulating this kind of process by means of probabilities is, thus, essential for realism.

In order to achieve stochastic behavior in the fractal generation process, LinF permits the association of probabilities with symbols. When a symbol that is the left-hand-side of some rule is preceded by a sequence given by a number and the `?` character, a probability of expansion equal to this number is associated with that symbol. The specification shown in Figure 9 is a simple example of stochastic L-System. If that grammar is used to fill a virtual garden with vegetables, all plants will have different appearances, although they are defined in the same way. The different instances of the images, in this example, are obtained by varying the probability parameter associated with the production rule `i = [.5? j]`.

In addition to associating probabilities with the application of rules, LinF allows the definition of angles as intervals, instead of fixed values. That is a powerful resource for creating statistically self-similar fractals. This technique can be used for simulating the microscopic nature of gases. Gas molecules in a container continually collide with

```
lines f, g

angles
a: (pi/3, 0)
b: (0, .3)

axiom 5@ f

rules
f = .5@ g h;
h = {i+a i+a i+a
     i+a i+a i};
i = [.5? j];
j = +b f
```

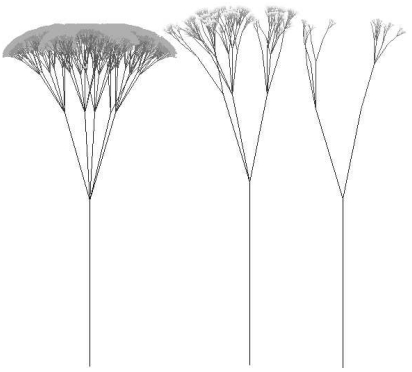

Figure 9. Stochastic 3D plant simulation in LinF. The different images are obtained by varying the probability parameter associated with the production rule i = [.5? j]

one another and the container's walls, describing a pattern known as Brownian motion. This motion pattern, for a single molecule, can be simulated with the grammar shown in Figure 10.

```
lines f

angles (0, [0 .. 2*pi])

axiom f

rules f = f+f
```

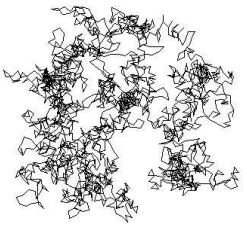

Figure 10. Brownian motion simulation.

It is possible to combine the stochastic tools provided by LinF in order to describe a wide range of real entities. In Figure 11 there is an example of modeling of feathers with L-Systems. The probabilistic rule is used to vary the size of the feather's barbs and the definition of an angle as an interval is used to model the imperfections in any of the feather's lateral "branches".

Once models of fractal objects have been created, it is possible to render a complete scene by placing several transformed instances of these models together. Figure 12 illustrates two different instances of a simple fractal tree.

5 A comparison between LinF and other L-System-based fractal generators

Since the publication of the Lindenmayer's formalism [16], several implementations of L-System interpreters have been developed – for instance, CPFG [19], Fractint [9], LParser [14] and Lin's system [15]. The great majority of those implementations – including all the systems listed above – are based in the language for L-System specification described by Prwsinkiewicz in his famous book "The

```
lines f, g

angles
a: (0, pi)
b: (0, pi/2)
c: (0, .003)
   (0, [0 .. .012])

axiom f

rules
f = [.5@ h] +a [i];
h = +g+g+g+g+g+g+g+g+g
    +g+g+g+g+g+g+g+g;
i = i-c [+b 1.3@ j]
    [-b 1.3@ j] g;
j = -k-g {-g {l}};
k = -k;
l = -g {-g {m}};
m = -g {-g {n}};
n = -g {-g {o}};
o = -g {-g {p}};
p = -g {-g .5? q};
q = -q
```

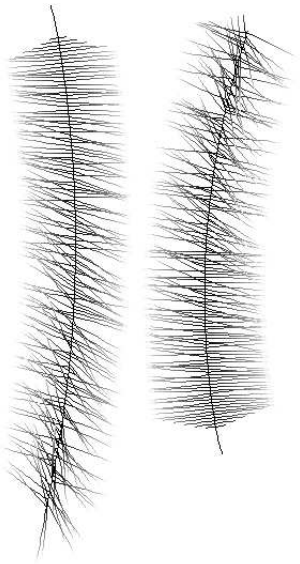

Figure 11. Two realizations of a stochastic L-System for modeling feathers.

```
lines f

angles
a: ([.9 .. 1.1], 0)
b: (0, [.75 .. 1.15])
c: ([1.5 .. 1.65], 0)
d: (0, [.65 .. .95])

axiom 2@ f

rules
f = f [g g g g g g]
    [h h h h] i;
g = +a [{+b .5@ f}];
h = +c [{+d .5@ f}];
i = [{.5@ f}]
```

Figure 12. Two realizations of stochastic L-System for modeling trees.

Algorithmic Beauty of Plants" [26]. Although LinF is also based on the notation set up by Prusinkiewicz, it departs from the other systems in two fundamental aspects. Firstly, in the way that three-dimensional rotations are specified. Secondly, in the way in which randomness is added to grammar rules. In addition to this, LinF allows the user to define several symbols representing lines, blank spaces and angles, a facility that to the best of our knowledge has not been incorporated in existing systems.

An orientation in the 3D space can be represented by three vectors: h, l and u. These vectors have unit length, are perpendicular to each other and thus satisfy the equation $h \times l = u$. In the interpretation originally devised by Przemyslaw Prusinkiewicz [27] rotations were expressed by the equation $[h', l', u'] = [h, l, u]\, R$, where R is a 3×3 rotation matrix [10]. Because there are three possible different references for rotations, the matrix R can have three different forms, depending on the rotation direction. The consequence of this approach is the use of at least six symbols to define rotations, two of them for each possible direction, meaning the positive and negative orientation. On the other hand, the approach adopted in LinF to represent changes in the drawing direction has been borrowed from differential geometry and is more concise. Such system, which has been described in Section 3.3, allows the definition of several distinct rotation angles and demands just two predefined symbols ($+$ and $-$) in the specification of rotations.

Another difference between LinF and those other implementations is the approach used to describe stochastic L-Systems is that LinF permits the user to define angles as intervals of probable values. Among the programs cited in this section, just LParser [14] allows a similar technique although its approach is not so general.

There are some examples of L-System interpreters that are general enough to simulate the LinF's semantics regarding symbol definitions, three-dimensional rotations and stochastic approach; however, such systems yield really complex definitions, if compared with LinF. For instance, RTEvol [4], a system that allows L-System evaluation, permits that C like functions defined by the user be used to modify the way in which the grammar symbols are interpreted, but these specifications demands some knowledge about the C programming language.

6 Final considerations

This paper presented LinF, a programming language that can be used in order to define L-Systems. LinF allows the specification of self-similar fractals and statistically self-similar fractals in two-dimensional and three-dimensional spaces. The paper also introduced cFLC, an interpreter of LinF written in C, which uses OpenGL to perform image generation. This tool can be downloaded at `http://www.dcc.ufmg.br/~carceron/linf`.

Although the L-System formalism has been deeply analyzed in previous works and is already a consolidated area in mathematics and computer science, there is not a universally-accepted standard for three-dimensional L-System specification. Because LinF utilizes concepts of differential geometry for rotation specification it is a suitable alternative for constructing tools that aim at simplifying the process of the fractal specification, generation and visualization in the three-dimensional space.

In addition, LinF gives its user great flexibility in the use of probabilities for fractal generation. Angles can be de-

fined as intervals of admissible values and the expansion of some symbols during the evolution of the string that represents a fractal can be conditioned to a pre-defined probability. Taken together, these features allow the specification of a wide range of realistic fractals more easily and efficiently than in existing systems, as evidenced by the various examples displayed throughout this paper.

Acknowledgements

This research has been supported by CNPq, by Fapemig and by PRPq-UFMG (Fundo Fundep RD).

References

[1] H. Abelson and A. A. Siessa. *Turtle Geometry*. MIT Press, 1981.

[2] Y. Chen, Y. Xu, B. Guo, and H.-Y. Shum. Modeling and rendering of realistic feathers. In *Proc. SIGGRAPH*, pages 630–636, 2002.

[3] N. Chomsky. *The logical structure of linguistic theory*. Indiana University Linguistics Club, 1955.

[4] N. D. Cuong. Ray traced evolution - user's manual, 1997. http://www.stud.tu-ilmenau.de/∼juhu/GX/RTEvol/ – accessed July 2003.

[5] S. Czanner, R. Durikovic, and H. Inoue. Growth simulation of human embryo brain. In *Proc. Spring Conference on Computer Graphics*, 2001.

[6] F. M. Dekking. Recurrent sets: a fractal formalism. Technical Report 82-32, Technische Hogeschool, Delft, The Netherlands, 1982.

[7] G. A. Edgar. *Measure, Topology, and Fractal Geometry*. Springer-Verlag, 1990.

[8] Frijters and Lindenmayer. A model for the growth and flowering of Aster novaeangliae on the basis of table (1,0)L-systems. *L Systems*, 15:24–52, 1974.

[9] N. Giffin. Fractint. http://spanky.triumf.ca/www/fractint/fractint.html – accessed July 2003.

[10] J. Hearn and M. Baker. *Computer Graphics: C Version*. Prentice Hall, second edition, 1997.

[11] Hogeweg and Hesper. A model study on biomorphological description. *Pattern Recognition*, 6:165–179, 1974.

[12] G. S. Hornby and J. B. Pollack. Evolving L-systems to generate virtual creatures. *Computers and Graphics*, 25(6):1041–1048, 2001.

[13] G. Kókai, Z. Tóth, and R. Ványi. Modelling blood vessels of the eye with parametric L-systems using evolutionary algorithms. In *Proc. Joint European Conf. on Artificial Intelligence in Medicine and Medical Decision Making*, pages 433–443, 1999.

[14] L. Lapré. Lsystems and Lparser. http://home.wanadoo.nl/laurens.lapre/ – accessed July 2003.

[15] T. Lin. Animation of l-system based 3-D plant growing in java. http://www.cs.umbc.edu/∼ebert/693/TLin/ – accessed July 2003.

[16] A. Lindenmayer. Mathematical models for cellular interactions in development, I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[17] B. Lintermann and O. Deussen. Interactive modelling of plants. *IEEE Computer Graphics and Applications*, 19(1), 1999.

[18] B. Mandelbrot. *The Fractal Geometry of Nature*. Freeman, 1977.

[19] R. Mech. CPFG Version 3.4 User's Manual. http://www.cpsc.ucalgary.ca/Research/bmv/lstudio/ – accessed July 2003.

[20] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics*, 23(3):41–50, 1989.

[21] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide: the official guide to learning OpenGL version 1.1*. Addison-Wesley, second edition, 1996.

[22] H. Noser, S. Rudolph, and P. Stucki. Physics-enhanced L-systems. In *WSCG 2001 Conference Proceedings*, 2001.

[23] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proc. SIGGRAPH*, pages 301–308, 2001.

[24] F. M. Q. Pereira. The LinF home page. http://www.dcc.ufmg.br/∼carceron/linf/ – accessed July 2003.

[25] P. Prusinkiewicz. Graphical applications of L-Systems. In *Proc. Graphics Interface*, pages 247–253, 1986.

[26] P. Prusinkiewicz. *The Algorithmic Beauty Of Plants*. Springer-Verlag, 1991.

[27] P. Prusinkiewicz, M. Hammal, R. Mech, and J. Hanan. The artificial life of plants. In *SIGGRAPH*, volume 1, pages 1–38, 1995.

[28] P. Prusinkiewicz and M. Hammel. A fractal model of mountains with rivers. In *Proc. Graphics Interface*, pages 174–180, 1993.

[29] P. Prusinkiewicz, M. S. Hammel, and E. Mjolsness. Animation of plant development. *Computer Graphics*, 27:351–360, 1993.

[30] P. Prusinkiewicz and J. Hanan. Lindemayer systems, fractals, and plants. *Lecture Notes in Biomathematics*, 1980.

[31] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. Development models of herbaceous plants for computer imagery purposes. In *SIGGRAPH*, volume 22, pages 141–150, 1988.

[32] R. Siromoney and K. Subramanian. Space-filling curves and infinite graphs. In *Proc. 2nd. Int. W. Graph Grammars and their Application to Computer Science*, pages 380–391. Springer-Verlag, 1983.

[33] A. R. Smith. Plants, fractals, and formal languages. *Computer Graphics*, 18(3):1–10, 1984.

[34] F. Soddel. *Using Lindenmayer Systems to Model the Growth of Filamentouw Micoorganisms*. PhD thesis, La Trobe University, Bendigo, 1994.

[35] A. L. Szilard and R. E. Quinton. An interpretation for DOL systems by computer graphics. *The Science Terrapin*, 4:8–13, 1979.

[36] K. Teneblat. *Introdução à Geometria Diferencial*. Editora UnB, 1988.