A Depth First Search Strategy for L-systems' Turtle Interpretation

Manuel Menezes de Oliveira Neto
Vítor César Benvenuti

FURB - Regional University of Blumenau
Rua Antônio da Veiga, 140
Caixa Postal 1507
89.010-971 Blumenau, SC, Brasil
furb@ibm.ufsc.br

Abstract. L-systems have become one of the most popular approaches for plants and trees modeling. Although they have been widely explored by many researchers, they are almost always treated at grammar-level, while few attention has been given to implementation issues. The only known published algorithmic description for L-systems' turtle interpretation is based on a two steps assembly and interpretation process over a string. This paper proposes a recursive more natural interpretation algorithm that operates in parallel to the process of translation of grammar's symbols, benefiting from its recursive nature. The proposed algorithm dispenses the traditional string assembly. It also has proved to be faster and more robust than the string-based one.

Introduction

The most far reaching goal for image synthesis may be the creation of a visual experience *identical* to that which would be experienced in viewing the real environment [Cohen-Wallace (1993)]. A fundamental difficulty in achieving total visual realism is the complexity of the real world [Foley et al. (1990)].

Natural phenomena resist geometric modeling. For instance, "the shape of a leaf of a tree may be modeled with polygons and its stem may be modeled with a spline tube, but to place explicitly every limb, branch, twig and leaf of a tree would be impossible time consuming and cumbersome" [Foley et al. (1990)]. So, not only the modeling of natural phenomena but of many complex objects require a simpler not geometric approach. One possible solution is the use of large class of objects which can be adjusted parametrically [Foley et al. (1990)]. Watt [Watt-Watt (1992)] presents two motivations to the use of procedural modeling: animation of objects can be easily achieved by use of time-varying parameters; the second one is its low-cost visual complexity. Fractal-based terrain generation, Fourier synthesis, three-dimensional texture mapping, particle systems and grammar-based modeling are examples of these techniques. Although some of these are very popular, all of them possess a specificity [Watt-Watt (1992)]. For instance, grammar-based modeling is better suited to model plants and trees.

Many procedural methods for generating images of plants and trees have been proposed in the last few years [Reeves-Blau (1985)], [Prusinkiewicz et al. (1988)], [Reffye et al (1988)], [Barnsley et al. (1988)], [Viennot et al. (1989)]. The most important ones are iterated function systems (IFS for short) [Barnsley (1988)] and grammar-based systems [Prusinkiewicz-Lindenmayer (1990)].

This paper rewies these two techniques and presents a detailed description of L-Systems: a grammar-based system used in computer graphics for realistic visualization of plant structures and developmental processes. Then, it discuss the algorithm presented by Saupe [Peitgen-Saupe (1988)] for L-systems' turtle interpretation and propouses a more concise and elegant one based on depth first search strategy [Tremblay-Sorenson (1984)]. This is a more natural approach that operates simultaneously to the process of translation of grammar's symbols, benefiting from the recursive nature of the translation process.

Iterated Function Systems

A planar iterated function system is a finite set of contractive affine mappings $T = \{T_1, T_2, \ldots, T_n\}$ which map the plane R^2 into itself. It also requires a set of probabilities $\{p_1, p_2, \ldots, p_n\}$, where $p_i > 0$ and $\sum_{i=1}^{n} p_i = 1$. The set defined by T is the smallest nonempty set A, closed in the topological sense, such that the image y of any point $x \in A$ under any of the mappings $T_i \in T$ also belongs to A [Barnsley (1988)].

IFS algorithms include the ability to produce complicate images and textures from small databases and are potentially suitable to parallel implementations. Another important feature is that the Collage algorithm, based on Collage Theorem [Barnsley (1988)], provides a means of interactive geometric modeling. In the two-dimensional case, the input to the

algorithm is a target image which can be obtained digitalizing an image of the object to be modeled [Barnsley et al. (1988)]. Figure 1 was generated with an iterated function system.

Figure 1: A fern obtained with IFS. (After [Barnsley (1988)].

Grammar-Based Systems

Although IFS algorithms are very powerful, their mathematical foundations are unfamiliar to most non matematicians. On the other hand, grammar-based algorithms are very simple and can be easily understood.

Plant structures exhibit some simple and elegant features that contribute to their beauty such as bilateral symmetry of leaves, rotational symmetry of flowers, helical arrangements of scales in pine cones and self-similarity [Prusinkiewicz - Lindenmayer (1990)]. Mandelbrot [Mandelbrot (1982)] describes self-similarity as a property of a fractal object in which each part of a shape is geometrically similar to the whole. In any domain in which the objects being modeled exhibit sufficient regularity, there may be an opportunity to develop a grammar-based model [Foley et al. (1990)].

The main idea of grammar-based systems is the notion of defining complex objects by successively replacing symbols of an initial description (*axiom*) by another ones according to a set of *rewriting* or *production rules*. Although grammar-based definitions are powerful tools and it is possible to recognize a topological relationship between symbols, the grammar itself has no inherent geometric content. So, using a grammar-based model requires both a grammar and a geometric interpretation of the language [Foley et al. (1990)].

Grammars are also useful in many other fields in computer science, such as formal definition of programming languages [Aho-Sehti-Ullman (1986)].

L-Systems

The most popular grammar-based technique for image synthesis is Lindenmayer system (L-system for short). Prusinkiewicz [Prusinkiewicz-Lindenmayer (1990)] gives a formal definition to a context-free L-system (noted OL-system): a string OL-system is an ordered triplet $G = \langle V, w, P \rangle$ where V is the alphabet of the system, $w \in V^+$ is a noncmpty word called the axiom and $P \subset VxV^*$ (V^* is the set of all words over V) is a finite set of production. A production $(a, \chi) \in P$ is written as $a \rightarrow \chi$. The letter a and the word χ are called the predecessor and the successor of this production, respectively. It is assumed that for any letter $a \in V$, there is at least one word $\chi \in V^*$ such that $a \rightarrow \chi$. If no production is explicitly specified for a given predecessor $a \in V$, the identity $a \rightarrow a$ production is assumed to belong to the set of production rules P. An OL-system is deterministic (noted DOL-system) if and only if for each $a \in V$ there is exactly one $\chi \in V^*$ such that $a \rightarrow \chi$.

L-systems can be used to describe the branching topology of modeled plants and their production rules can be applied in parallel to replace all symbols in a given expression simultaneously. Association of geometric features to the symbols of the language makes L-systems detailed enough to allow their use in computer graphics for realistic visualization of plant structures and developmental processes [Prusinkiewicz - Lindenayer (1990)].

Smith [Smith (1984)] demonstrated the potential of L-systems for realistic image synthesis. Szilard and Quinton [Szilard-Quinton (1979)] demonstrated that simple DOL-systems could generate fractal curves. Siromoney and Subramanian [Siromoney-Subramanian (1983)] specified L-systems that generate classical space-filling curves. Prusinkiewicz focused on L-system interpretation based on a LOGO-style turtle [Prusinkiewicz-Lindenmayer (1990)]. His approach uses some predefined symbols. A brief description of

the most useful ones for a two-dimensional space is given below. For a complete description of the symbols used in the turtle LOGO-style see [Prusinkiewicz - Lindenayer (1990)].

F : draws a line segment of length d from current (x,y) coordinates to (x',y'), where $x' = x + d\cos\alpha$, $y' = y + d\cos\alpha$ and α is a resultant rotation angle;

f : moves from (x,y) to (x',y') without drawing;

+ : rotates the drawing direction by δ degrees in counter-clockwise sense;

- : rotates the drawing direction by δ degrees in clockwise sense;

[: saves current position parameters in a stack (it's used to draw ramifications);

] : restores position parameters from the stack;

Two parameters must be informed: the number of recursive interpretations for the axiom and the rotational angle to be apllied at each rotation symbol. In order to clarify these concepts, two simple examples are ilustrated by figures 2 and 3. Both objects were produced with 6 levels of recursion and $\delta = 60°$. Table 1 sumarizes their data.

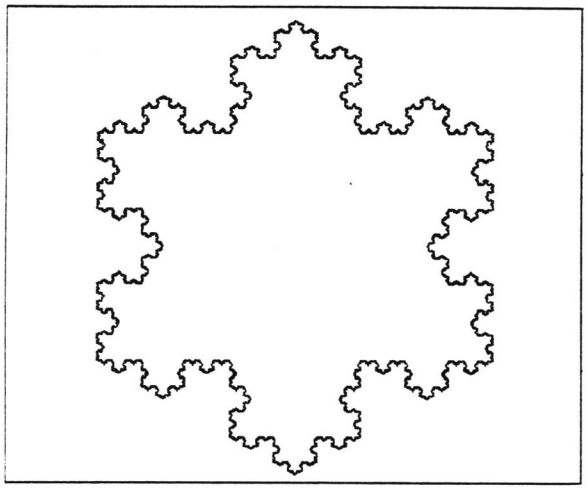

Figure 2: Koch's curve [Koch (1905)]

Table 1: Objects' data for figures 2 and 3

Object	Axiom	Productions Rules
Koch	F--F--F	F→F+F--F+F
Sierp.	FXF--FF--FF	F→FF X→--FXF++FXF++FXF--

All plants generated by the same DOL-system are identical: an attempt to combine then in the same picture would produce an artificial regularity. To solve this problem, Prusinkiewcz [Prusinkiewicz-Lindenmayer (1990)] sugest stochastic application of production rules. This warrants that both topology and geometry will change from plant to plant, preserving general aspects of a plant, but modifying its details. Another important aspect covers context-sensitive L-systems. In this case, production rules are applicable, or not, according to the context they appear. Thus, somes rules can only be applied to some plant's parts. This enables to model some form changes that occur due, for example, to the flow of nutrients or hormones [Prusinkiewicz - Lindenmayer (1990)].

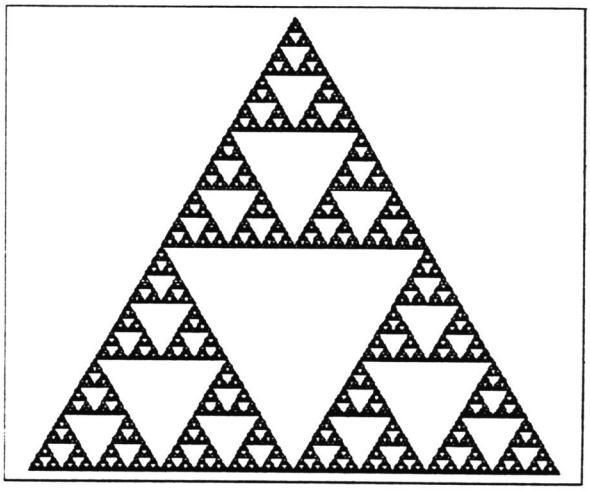

Figure 3: Sierpinsky gasket [Sierpinski (1915)]

Prusinkiewicz also uses special symbols in the production rules to indicate the positioning of plant organs such as leaves and petals. The results obained with this technique are pretty good (figure 4).

Saupe's Algorithm for L-Systems' Turtle Interpretation

Although L-systems are very popular, they are almost always described at grammar-level. Many authors provide long comments on L-systems but no algorithmic description can be found either in [Prusinkiewicz et al. (1988)], [Prusinkiewicz - Lindenmayer (1990)], [Foley et al. (1990)] and [Watt - Watt (1992)]. Saupe [Peitgen-Saupe (1988)] gives an algorithmic description suitable for implementation. The main procedures with their arguments, variables and functions are listed below (extracted from [Peitgen-Saupe (1988)]).

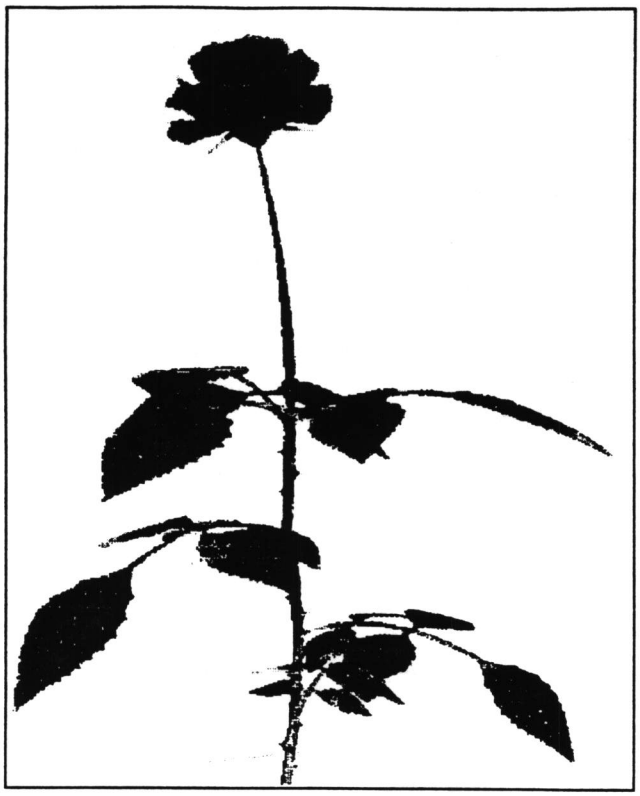

Figure 4: Plant organs: leaves and petals. (After [Prusinkiewicz - Lindenmayer (1990)]

```
{ Fractal curves and plants using OL-systems }
ALGORITHM Plot-OL-System (maxlevel)
Arguments maxlevel   number of cycles in string generation
Globals   axiom      string with the axiom of OL-system
          Kar[]      character array, inputs of production
          Rule[]     string array, outputs of production
          num        number of production rules
          TurtleDir  N# of possible directions of turtle
Variables str        string to be generated & interpreted
          xmin,xmax  range of the curve in x direction
          ymin,ymax  range of the curve in y direction
BEGIN
   GenerateString (maxlevel, str)
   ClearUpString (str)
   GetCurveSize (str, xmin, xmax, ymin, ymax)
   TurtleInterpretation (str, xmin, xmax, ymin, ymax)
END

{ String generation in OL-Systems }
Arguments maxlevel   number of cycles in string generation
          str        output string
Globals   axiom      string with the axiom of OL-system
          Kar[]      character array, inputs of production
          Rule[]     string array, outputs of production
          num        number of production rules
Variables level      integer, # of string generation cycle
          command    character
          str0       string
          i, k       integer
Functions strlen(s)  returns the length of string s
          strapp(s,t) returns string s appended by string t
          getchar(s,k) returns the kth character of string s
          strcpy(s,t) copies string t into string s
ALGORITHM GenerateString (maxlevel, str)
begin
   strcpy (str0, axiom)
   strcpy (str, "")
   for level =1 to maxlevel do
      for k=1 to strlen (str0) do
            command := getchar(str0, k)
            i := 0
            while (i < num and
                  not command = Kar[i]) do
               i := i + 1
            end while
            if (command = Kar[i]) then
               str := strapp(str, Rule[i])
            end if
      end for
      strcpy (str0, str)
   end for
end

{Clean out all redundant state preserving commands}
Arguments str        string
Variables str0       string, initialized to ""
          i          integer
          c          character
Functions strlen(s)  returns the length of string s
          strappc(s,c) ret string s appended by character c
          getchar(s,k) returns the kth character of string s
          strcpy(s,t) copies string t into string s
ALGORITHM CleanUpString (str)
begin
   for level =1 to strlen (str) do
      c := getchar(str, i)
      if (c='F' or c='f' or c='+' or c='-'
         c='|' or c='[' or c=']') then
            str0 := strapp(str0, c)
      end if
   end for
   strcpy (str, str0)
end
```

Saupe [Peitgen-Saupe (1988)] presents three more procedures *GetCurveSize*, *UpdateTurtleState* and *TurtleInterpretation* which are not important in this context.

Saupe's algorithm is based on a two steps process which can be observed in procedure *Plot-OL-System*. Its

first line calls *GenerateString* which generates a string to be interpreted by *TurtleInterpretation* (last line). Another feature of Saupe's method is its iterativity that can be observed in the outer *for* loop in *GenerateString*. Iterativity was used although grammar-based descriptions have a recursive nature [Aho - Sehti - Ullman (1986)]. The use of string-based implementations leads to many unnecessary copy and append string operations, which are critical when the object being modeled needs large recursion level or uses many long production rules. Figure 5 ilustrates Saupe's process for string assembly. Colors were used to clear the process.

Axiom	Production rule
F - F	F → F + F + F

level	string
0	F - F
1	F+F+F - F+F+F
2	F+F+F + F+F+F + F+F+F - F+F+F + F+F+F + F+F+F

Figure 5: String assembly by Saupe's algorithm

Another feature of Saupe's algorithm is the use of a clean up string step (*ClearUpString* procedure). It is used to extract all non-terminal symbols from the string that will be interpreted. The string interpretation step (*TurtleInterpretation*) must analyze each string symbol in order to decide which actions have to be carried out. This process, implemented as a case statement over string symbols, points out that a clean up step is actually unnecessary.

The Depth First Search (DFS) Strategy

A more natural turtle interpretation algorithm can be supported by recursion. Each time a predecessor is found in the axiom, the current state is stacked and the selected production rule is analyzed in the same way. If the current symbol is not a predecessor (it is said to be a terminal [Sethi (1989)]) or if the maximum recursion level was reached, the symbol must be interpreted. This defines an algorithm in an elegant fashion closely related to the grammar that must be parsed. Figure 6 sumarizes its behavior. Again, colors were used to clarify the process. Its main procedures, parameters, variables and functions are presented below using the same syntax used by Saupe in [Peitgen-Saupe (1988)].

```
{ Fractal curves and plants using OL-systems }
ALGORITHM Depth_First_Search (maxlevel)
Arguments maxlevel   maximum recursion level
Globals   axiom      string with the axiom of OL-system
          Kar[]      character array, inputs of production
          Rule[]     string array, outputs of production
Variables xmin,xmax  range of the curve in x direction
          ymin,ymax  range of the curve in y direction
begin
   GetCurveSize (xmin, ymin, xmax, ymax)
   InterpDraw (0, maxlevel, 1, axiom)
end

{Interprets and draws the picture}
Arguments clevel     current recursion level
          maxlevel   maximun recursion level
          pos        position for analysis
          str        axiom or production rule
Functions getchar(s,k) returns the kth character of string s
          act_over(c) interprets the symbol s - uses a
                      case statement
ALGORITHM InterpDraw(clevel,maxlevel,pos, str)
begin
  if (clevel = maxlevel) then
     c = getchar(str,pos)
     while ( not (c = eol))
     if (c='F' or c='f' or c='+' or c='-'
         c='|' or c='[' or c=']') then
         act_over(c)
     end if
     c = getchar(str,pos)
     end while
   else
    c = getchar(str,pos)
    while ( not (c = eol))
    if (c in Rule[])
        InterpDraw(clevel+1,maxlevel,1, Rule)
    end if
    c = getchar(str,pos)
    end while
 end if
end
```

Using DFS, L-systems interpretation is started as soon as a terminal symbol is found or the maximum recursion level is reached. It is not necessary to wait for knowing the complete sequence of symbols. The process is started very early in the analysis step and each time a portion of the L-system is interpreted, the memory used to store it is liberated. This leads to more clever memory management.

DFS algorithm is a more natural solution for L-systems' turtle interpretation, since it reflects grammar's recursive nature. It leads to a more concise and clear code, as can be observed in the presented procedures. Observe that *InterpDraw* embodies all comands used by

either Saupe's *GenerateString*, *ClearUpString* and *TurtleInterpretation* procedures. It also shows that *ClearUpString* is unnecessary.

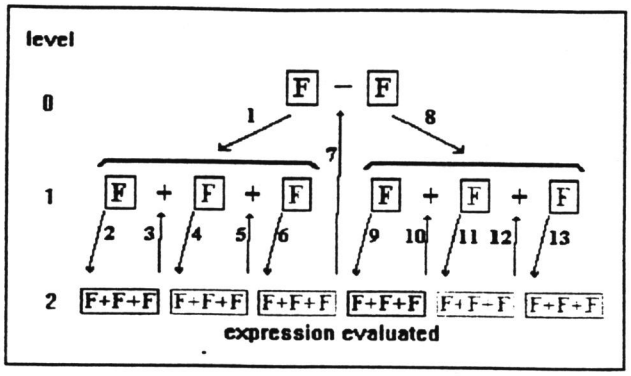

Figure 6: Depth first solution for axiom **F - F** and production rule **F→F+F+F.** Numbered arrows indicate the flow (order) of avaliation.

As there no time spent with copy and append string operations and with a clean up procedure, DFS is faster than Saupe's algorithm.

The same strategy is used, with few modifications, to calculate the limits of the picture (*GetCurveSize*). It is only necessary to update the limit coordinates (*xmin, ymin, xmax, ymax*) during a first exploration, using a standard segment length *d* (usually one). After that, the actual length is obtained by scaling *d* according to viewport dimensions.

Results

In order to compare DFS with Saupe's approach, both algorithms were implemented. DFS was implemented as described in the last section. For Saupe's version, it was used a linked linear list [Tremblay-Sorenson (1984)] to contain the string: each node containing a symbol field and a pointer to the next node of the list. This was used to avoid problems with insufficient static memory allocation. In such a way, the algorithm can use all avaliable memory. It also avoids the need of copying the string, since new symbols can be inserted in any part of the list with a low computational cost. The clean up step was ignored. The analysis was carried out with the described improved Saupe's version.

Both programs were coded in Turbo Pascal 6.0[1] and run on a IBM PC compatible 386 SX 40 MHz with 4 megabytes of RAM memory and Microsoft DOS 5.0 operating system. Images were displayed on a color monitor with SVGA board with 1 megabyte of video memory. Execution times were measured as the difference of the values returned by the Turbo Pascal's *gettime* function. This function was called immediately before the start of the procedures and immediately after their finish. It must be noted that as the clock time is not updated each hundredth of second, *gettime* does not provide a safe measure for differences less than some treshold.

Both programs were submitted to sets of runs. Figure 7 shows the times recorded for the interpretation of the following L-system which corresponds to tree presented in figure 8 (extracted from [Prusinkiewicz-Lindenmayer (1990)]):

axiom : X

production rules: X→F-[[X]+X]+F[+FX]-X
F→FF

For this particular L-system, Saupe's improved version did not supported more than 5 iterations. Since then, the program returned a *heap overflow error* message. The same message was presented for all L-systems interpretations using Saupe's algorithm, since a certain number of iterations are executed (this number varied from model to model). With such a restriction, the algorithms could not be compared for complex images, which require large number of recursive calls and many and complex production rules. These complex models would be used to compare algorithms' performances.

The algorithms were also executed for 5 more L-systems presented in [Prusinkiewicz - Lindenmayer (1990)]. These L-systems are described in table 2.

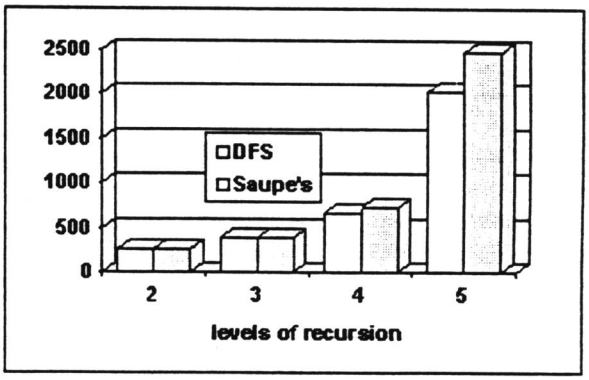

Figure 7: Time (in thousandths of second) for a set of runs involving DFS and Saupe's improved algorithms.

The maximum number of iterations supported by Saupe's algorithm for the models listed in table 2 was, respectively, 5, 5, 4, 7 and 7. Coincidentally, these are the same recursion levels used to generate these pictures in [Prusinkiewicz-Lindenmayer (1990)]. Although, at first glance, these values seems to be too small to cause

[1]Trade mark of Borland International Inc.

a program halt, they hide an exponential growth of the string length. Table 3 shows the behavior of the string length for the first L-system from table 2. Figure 9 ilustrates the growth of the string.

Table 2: L-systems used to compare DSF and Saupe's improved algorithm.

Tree #	Axiom	Productions
1	F	F→F[+F]F[-F]F
2	F	F→F[+F]F[-F][F]
3	F	F→FF-[-F+F+F]+[+F-F-F]
4	X	X→F[+X]F[-X]+X F→FF
5	X	X→F[+X][-X]FX F→FF

On the other hand, DFS supported 57 levels of recursion for all these models and for all others used to test it. DFS fails only when the system reaches its stack size (64 Kbytes). DFS does not need to know the final sequence of symbols to start the process of interpretation. It is started very early in the analysis step and each time a portion of the L-system is interpreted, the memory used to store it is liberated.

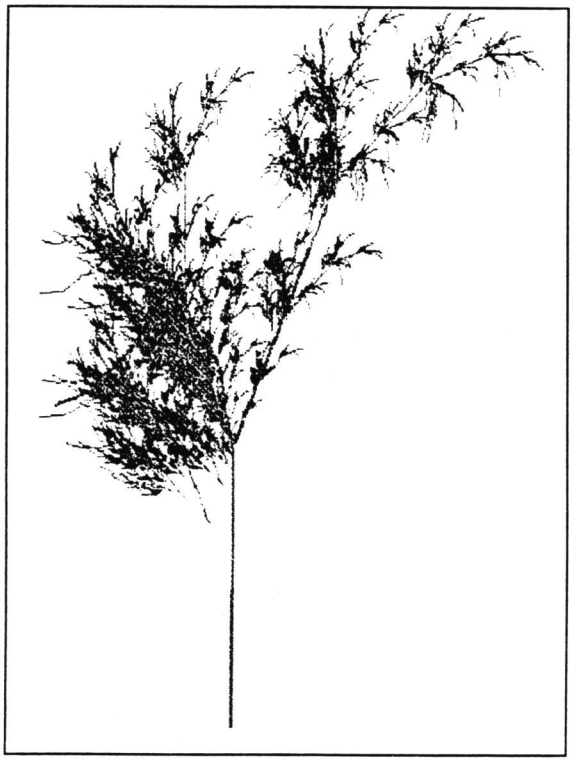

Figure 8: A tree model used to compare the algorithms.

Table3: Length of the string versus levels of recursion

level of recursion	length of the string
0	11
1	61
2	311
3	1,562
4	7,813
5	39,080

For a typical SVGA resolution with addressability of 1024x768 pixels, it was observed that for models like those described in tables 1 and 2, more than eigth or nine levels of recursion can not improve the quality of the pictures. They are limited by the impossibility to draw lines shorter than a pixel.

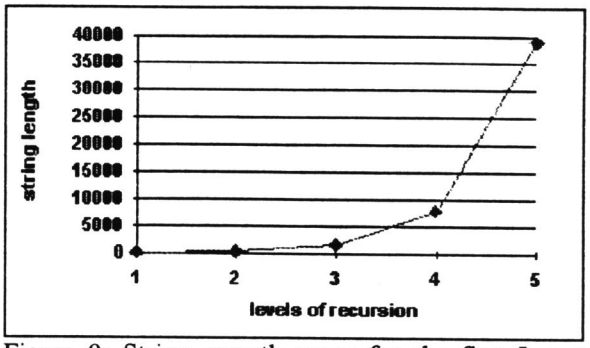

Figure 9: String growth curve for the first L-system from table 2.

Figures 2, 3, 8 and 9 were rendered using DFS algorithm.

Conclusions

Although L-systems have become one of the most popular approaches for plants and trees modeling, few attention has been given to implementation issues. This paper detailed L-systems concepts and analyzed Saupe's algorithm for L-systems turtle interpretation. It points some drawbacks of Saupe's approach and proposes a depth first search strategy as a better solution to the problem of interpretation. DFS proved to be a more concise, natural and elegant way for approaching this problem, since it reflects the recursive nature of the grammar to be parsed. It is faster than Saupe's algorithm since it eliminates some unecessary steps such as assembly and clean up of a string. It also proved to be more robust than Saupe's, supporting until 57 levels of recursion.

Althoug DFS can be used to generate realistic images of plants and trees which can be used in a

rendering pipeline [Foley et al. (1990)], it is not suitable for animating plants developmental processes. Anyway, the proposed algorithm can be used in any situation in which this kind of animation is not required. To the purpose of developmental animation, Saupe's algorithm, as stated earlier, is not suitable either, since a left to right interpretation of an already assembled string will produce the same effect as a depth first search would.

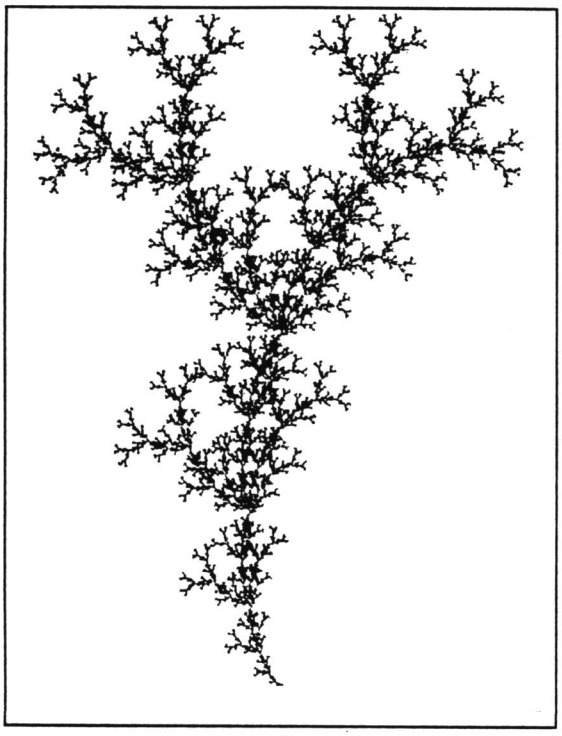

Figure 10: Example of a plant generated with DFS.

Acknowledgments

The authors would like to thank Rui Bastos, at National Supercomputing Center of Federal University of Rio Grande do Sul, for his comments. We gratefully thank Marcelo Walter at University of British Columbia for providing useful information about related works.

References

A. Aho, R. Sehti, J. Ullman, Compilers - Principles, Techniques and Tools, Addison Wesley, 1986.

M.Barnsley, Fractals Everywhere, Academic Press, 1988.

M. Barnsley et al., Harnessing chaos for image synthesis, *Computer Graphics* **22** (1988) 131-140, (Proceedings of SIGGRAPH '88).

M. Cohen, J. Wallace, Radiosity and Realistic Image Synthesis, Academic Press, 1993.

J. Foley et al., Computer Graphics principles and practice, Addison Wesley, 1990.

H. Koch, Une méthode géométric élementaire pour l'étude de certaines questions de la théorie des courbes planes. Acta mathematica **30** (1905), 145-174.

B. Mandelbrot, The Fractal Geometry of Nature, H. W. Freeman, 1982.

P. Prusinkiewicz, A. Lindenmayer, J. Hanan, Developmental models of herbaceous plants for computer imagery purposes, *Computer Graphics* **22** (1988) 141-150, (Proceedings of SIGGRAPH '88).

P. Prusinkiewicz, A. Lindemayer, The Algorithmic Beauty of Plants, Springer Verlag, 1990.

W. Reeves, R. Blau, Approximate and probabilistic algorithms for shading and rendering particle systems, *Computer Graphics* **19** (1985) 359-376, (Proceedings of SIGGRAPH '85).

P. Reffye et al., Plant models faithful to botanical structure and development, *Computer Graphics* **22** (1988) 151-158, (Proceedings of SIGGRAPH '88).

D. Saupe, A Unified approach to fractal curves and plants. In H. Peitgen, D. Saupe, editors, The Science of Fractal Images, 273-286, Springer Verlag, 1988.

R. Sethi, Programming Languages: Concepts and Constructs, Addison Wesley, 1989.

W. Sierpinski, Sur une courbe dont tout point est un point de ramification. Comptes Rendus hebdomadaire des séances de l'Académie des Sciences, **160** (1915) 302-305.

R. Siromoney, K. Subramanian, Space-filling curves and infinite graphs. In H. Ehrig, M. Nagel, G. Rozemberg, editors, Graph grammar and ther application to computer science, Second International Workshop, Lecture Notes in Computer Science, 380-391, Springer Verlag, 1983.

A. Smith, Plants, fractals and formal languages, *Computer Graphics* **18** (1984) 1-10, (Proceedings of SIGGRAPH '84).

A. Szilard, R. Quinton, An Interpretation for DOL systems by computer graphics,. The Science Terrapin, **4** (1979), 8-13.

J. Tremblay, P. Sorenson, An Introduction to Data Structures with Applications, McGraw-Hill, 1984.

X.Viennot et al., Combinatotial analysis of ramified patterns and computer imagery of trees, *Computer Graphics* **23** (1989) 31-40, (Proceedings of SIGGRAPH '89).

A. Watt, M. Watt, Advanced Animation and Rendering Techniques theory and practice, Addison Wesley, 1992.

Figure 4

Figure 8

Figuras a cores do artigo ***A depth first search strategy for L-systems' Turtle interpretation***