Appendix C

A unified approach to fractal curves and plants

Dietmar Saupe

C.1 String rewriting systems

In this appendix we introduce so called L-systems or string rewriting systems, which produce character strings to be interpreted as curves and pictures. The method, which is very interesting in itself, fills two gaps that the main chapters leave open. First, rewriting systems provide an elegant way to generate the classic fractal curves, e.g. the von Koch snowflake curve and the space filling curves of Peano and Hilbert. Secondly, the method had been designed by its inventors to model the topology and geometry of plants, in particular of the branching patterns of trees and bushes. Thus, this appendix provides a modeling technique which readily yields relatively realistic looking plants.

In the course of the algorithm a long string of characters is generated. The characters are letters of the alphabet or special characters such as '+', '−', '[', etc. Such a string corresponds to a picture. The correspondence is established via a LOGO-like turtle which interprets the characters sequentially as basic commands such as "move forward", "turn left", "turn right", etc. The main ingredient of the method is the algorithm for the string generation, of course. A first string consisting of only a few characters must be given. It is called the *axiom*. Then each character of the axiom is replaced by a string taken from a table of *production rules*. This substitution procedure is repeated a prescribed number of times to produce the end result. In summary we have that the final picture is completely determined by

- the axiom,
- the production rules,
- the number of cycles,

and the definition for the actions of the turtle, which draws the picture from the output string.

ALGORITHM	**Plot-0L-System** (maxlevel)	
Title	Fractal curves and plants using 0L-systems	
Arguments	maxlevel	number of cycles in string generation
Globals	axiom	string containing the axiom of 0L-system
	Kar[]	character array (size num), inputs of production rules
	Rule[]	string array (size num), outputs of production rules
	num	number of production rules
	TurtleDirN	number of possible directions of turtle
Variables	str	string to be generated and interpreted
	xmin, xmax	range covered by curve in x direction
	ymin, ymax	range covered by curve in y direction

```
BEGIN
    GenerateString (maxlevel, str)
    CleanUpString (str)
    GetCurveSize (str, xmin, xmax, ymin, ymax)
    TurtleInterpretation (str, xmin, xmax, ymin, ymax)
END
```

We will see that in order to specify a complex curve or tree, only a few production rules will suffice. The axiom along with the production rules may be regarded as the genes which control the "growth" of the object. This information can be very small as compared to the complexity of the resulting picture. The same is true for the iterated function systems of Chapter 5. The challenge for iterated functions systems as well as for L-systems is to establish the system information necessary to produce a given object or an object with given properties. This is a topic of current research.

L-systems were introduced by A. Lindenmayer in 1968 for the purpose of modeling the growth of living organisms, in particular the branching patterns of plants. A. R. Smith in 1984 [96] and P. Prusinkiewicz in 1986 [88] incorporated L-systems into computer graphics. This Appendix is based on the work of P.

ALGORITHM	**GenerateString** (maxlevel, str)	
Title	String generation in 0L-System	
Arguments	maxlevel	number of cycles in string generation
	str	output string
Globals	axiom	string containing the axiom of 0L-system
	Kar[]	character array (size num), inputs of production rules
	Rule[]	string array (size num), outputs of production rules
	num	number of production rules
Variables	level	integer, number 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 k-th character of string s
	strcpy(s, t)	procedure to copy string t into string s

```
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
```

Prusinkiewicz. All definitions and most examples are taken from his paper[1], where the interested reader will also find a more detailed account of the history of L-systems and further references.

C.2 The von Koch snowflake curve revisited

The principle of L-systems can be exemplified with the von Koch snowflake curve of Section 1.1.1 (compare Figure 1.3). Let us assume that the LOGO-like turtle is equipped with the following character instruction set:

[1] See also his recent publication "Applications of L-systems to computer imagery", to appear in: "Graph Grammars and their Application to Computer Science; Third International Workshop", H. Ehrig, M. Nagl, A. Rosenfeld and G. Rozenberg (eds.), (Springer-Verlag, New York, 1988).

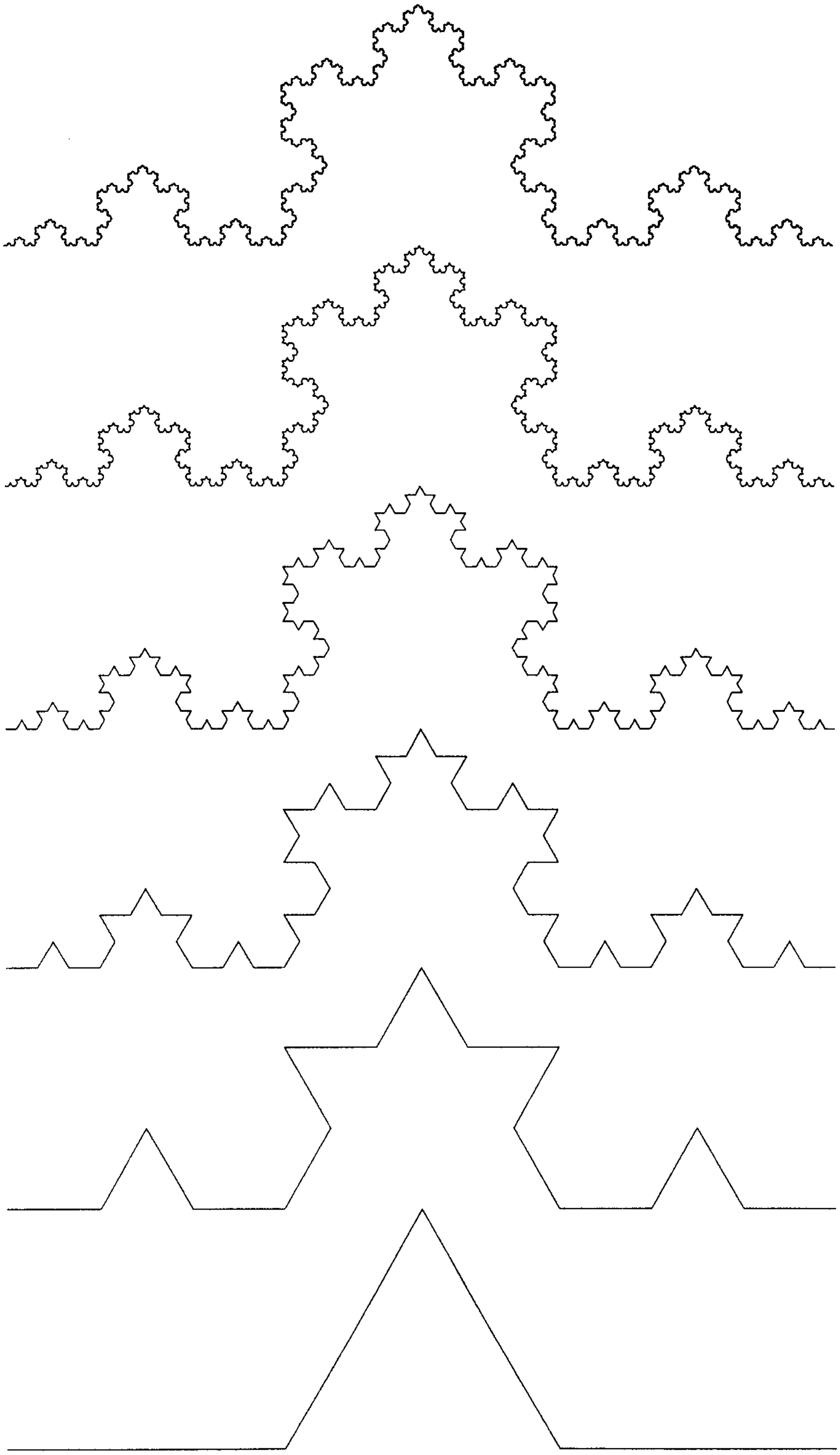

Fig. C.1: The first six stages in the generation of the von Koch snowflake curve. The 0L-system is given by the axiom "F", the angle $\delta = \frac{\pi}{3}$ and the production rule F $\longrightarrow$ F−F++F−F.

```
ALGORITHM CleanUpString (str)
Title         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)  returns string s appended by char c
              getchar(s, k)  returns k-th character of string s
              strcpy(s, t)   procedure to copy string t into string s

BEGIN
    FOR i=1 TO strlen (str) DO
        c := getchar (str, i)
        IF (c='F' OR c='f' OR c='+' OR c='-' OR c='|'
            OR c='[' OR c=']') THEN
            str0 := strappc (str0, c)
        END IF
    END FOR
    strcpy (str, str0)
END
```

'F' draw a line forward,

'+' turn right by 60°,

'–' turn left by 60°.

We start out with a straight line, denoted by "F". This is the axiom of the von Koch snowflake curve. In stage 1 the line is replaced by a line forward, a left turn, a line, two right turns for a total of 120°, a line, a left turn and another line. In the turtle language this can be written as the string "F–F++F–F". We can ignore the specification of the lengths of the line segments at this point. Subsequently, each line; symbolized by the character 'F', again has to be replaced by the string "F–F++F–F". Thus, in stage 2 we have the string

"F–F++F–F–F–F++F–F++F–F++F–F–F–F++F–F",

and in stage 3 we obtain

"F–F++F–F – F–F++F–F ++ F–F++F–F – F–F++F–F –
F–F++F–F – F–F++F–F ++ F–F++F–F – F–F++F–F ++
F–F++F–F – F–F++F–F ++ F–F++F–F – F–F++F–F –
F–F++F–F – F–F++F–F ++ F–F++F–F – F–F++F–F",

and so forth.

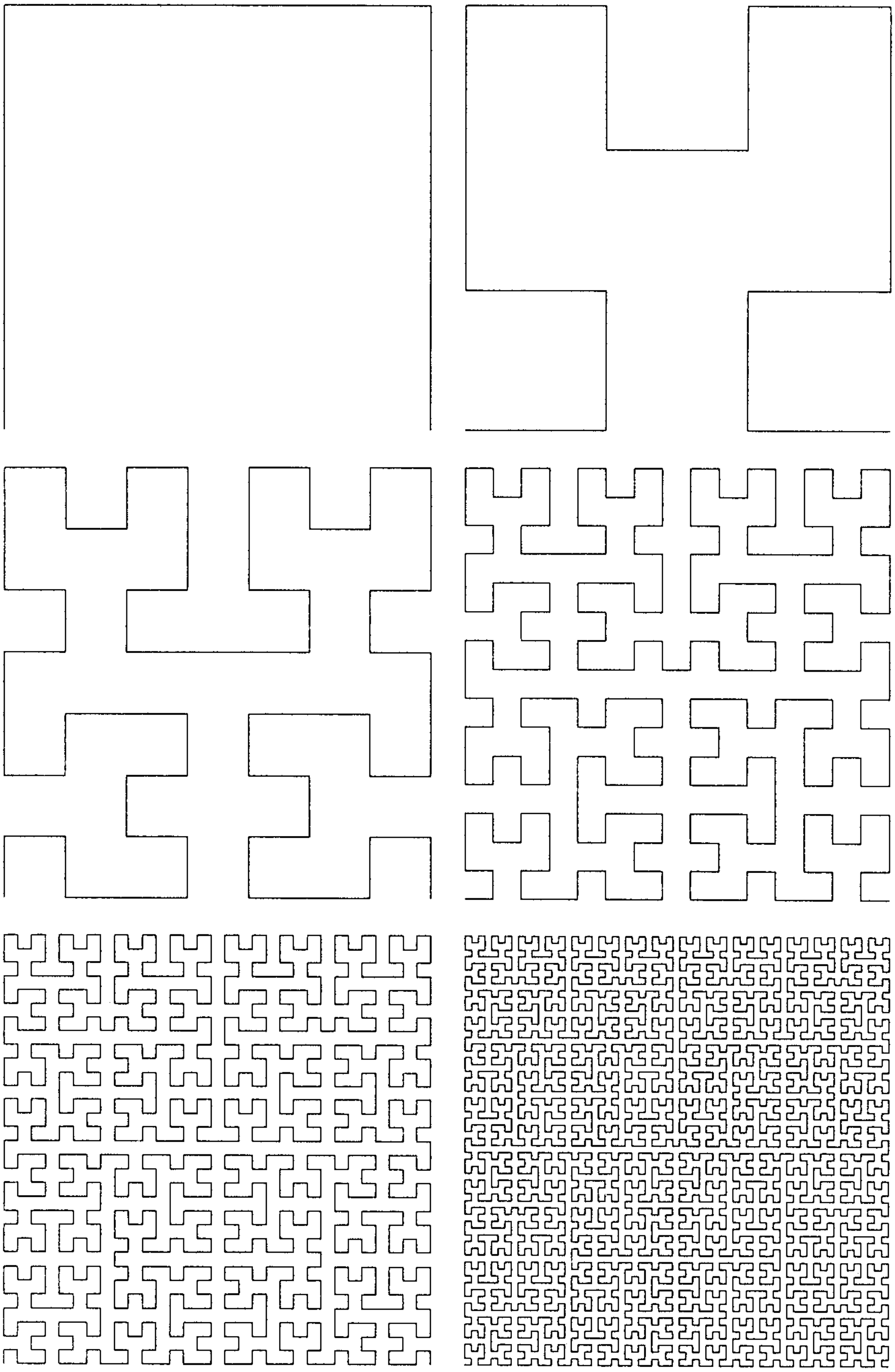

Fig. C.2: The first six stages in the generation of the space filling Hilbert curve. The 0L-system is given by the axiom "X", the angle $\delta = \frac{\pi}{2}$ and the production rules X $\rightarrow$ $-$YF+XFX+FY$-$, Y $\rightarrow$ +XF$-$YFY$-$FX+.

ALGORITHM **GetCurveSize** (str, xmin, xmax, ymin, ymax)		
Title	Compute the size of curve given by string	
Arguments	str	string
	xmin, xmax	range covered by curve in x direction
	ymin, ymax	range covered by curve in y direction
Variables	i	integer
	command	character
Globals	TurtleX	real x position of turtle
	TurtleY	real y position of turtle
	TurtleDir	direction of turtle (coded as integer)
	TurtleDirN	number of possible directions of turtle
	CO[]	array of TurtleDirN cosine values
	SI[]	array of TurtleDirN sine values
Functions	strlen(s)	returns the length of string s
	getchar(s, k)	returns k-th character of string s

```
BEGIN
    FOR i=0 TO TurtleDirN-1 DO
        CO[i] := cos (2 * Pi * i / TurtleDirN)
        SI[i] := sin (2 * Pi * i / TurtleDirN)
    END FOR
    TurtleDir := 0
    TurtleX := TurtleY := 0.0
    xmax := xmin := ymax := ymin := 0.0
    FOR i=1 TO strlen (str) DO
        command := getchar (str, i)
        UpdateTurtleState (command)
        IF (command='F' OR command='f') THEN
            xmax := MAX (TurtleX, xmax)
            xmin := MIN (TurtleX, xmin)
            ymax := MAX (TurtleY, ymax)
            ymin := MIN (TurtleY, ymin)
        END IF
    END FOR
END
```

In summary we have that when proceeding from one stage to the next we must replace a character 'F' by the string "F−F++F−F", while the characters '+' and '−' are preserved. Thus, the L-system consists of the axiom "F" and the production rules

$$F \rightarrow F{-}F{+}{+}F{-}F, \quad + \rightarrow +, \quad - \rightarrow -.$$

C.3 Formal definitions and implementation

There are different kinds of L-systems. Here we consider only 0L-systems, in which characters are replaced by strings (in pL-systems more complicated substitution rules may apply).

```
ALGORITHM UpdateTurtleState (command)
Title          Change the state of turtle according to given command

Arguments   command      character command
Globals     TurtleX      real x position of turtle
            TurtleY      real y position of turtle
            TurtleDir    direction of turtle (coded as integer, should be even)
            TurtleDirN   number of possible directions (initialized to 0)
            TStackX[]    stack of turtle x positions
            TStackY[]    stack of turtle y positions
            TStackDir[]  stack of turtle directions
            TStackSize   size of turtle stack
            TStackMax    maximal size of turtle stack
            CO[]         array of TurtleDirN cosine values
            SI[]         array of TurtleDirN sine values

BEGIN
    IF (command = 'F' OR command = 'f') THEN
        TurtleX := TurtleX + CO[TurtleDir]
        TurtleY := TurtleY + SI[TurtleDir]
    ELSE IF (command = '+') THEN
        TurtleDir := TurtleDir - 1
        IF (TurtleDir < 0) THEN
            TurtleDir := TurtleDirN - 1
        END IF
    ELSE IF (command = '-') THEN
        TurtleDir := TurtleDir + 1
        IF (TurtleDir = TurtleDirN) THEN
            TurtleDir := 0
        END IF
    ELSE IF (command = '|') THEN
        TurtleDir := TurtleDir + TutleDirN / 2
        IF (TurtleDir > TurtleDirN) THEN
            TurtleDir := TurtleDir - TurtleDirN
        END IF
    ELSE IF (command = '[') THEN
        IF (TStackSize == TStackMax) THEN
            PRINT ("ERROR : Maximal stack size exceeded.")
            EXIT PROGRAM
        END IF
        TStackX[TStackSize] := TurtleX
        TStackY[TStackSize] := TurtleY
        TStackDir[TStackSize] := TurtleDir
        TStackSize := TStackSize + 1
    ELSE IF (command = ']') THEN
        IF (TStackSize == 0) THEN
            PRINT ("ERROR : Stack empty.")
            EXIT PROGRAM
        END IF
        TStackSize := TStackSize - 1
        TurtleX := TStackX[TStackSize]
        TurtleY := TStackY[TStackSize]
        TurtleDir := TStackDir[TStackSize]
    END IF
END
```

```
ALGORITHM TurtleInterpretation (str, xmin, xmax, ymin, ymax)
Title        Plot the curve given by string

Arguments    str             string
             xmin, xmax      range covered by curve in x direction
             ymin, ymax      range covered by curve in y direction
Variables    i               integer
             command         character
             Factor          real
             ix, iy          integers
Globals      xsize, ysize    size of screen in pixels
Functions    strlen(s)       returns the length of string s
             getchar(s, k)   returns k-th character of string s
             MoveTo ()       move the graphics position
             DrawTo ()       draw a line to the new graphics position
             XScreen (x) = INT (Factor * (x - xmin))
             YScreen (y) = INT (Factor * (y - ymin))

BEGIN
    Factor := MIN ((xsize-1)/(xmax-xmin), (ysize-1)/(ymax-ymin))
    TurtleDir := 0
    TurtleX := TurtleY := 0.0
    MoveTo (XScreen (TurtleX), YScreen (TurtleY))
    FOR i=1 TO strlen (str) DO
        command := getchar (str, i)
        UpdateTurtleState (command)
        IF (command='F') THEN
            DrawTo (XScreen (TurtleX), YScreen (TurtleY))
        ELSE IF (command='f') THEN
            MoveTo (XScreen (TurtleX), YScreen (TurtleY))
        END IF
    END FOR
END
```

Let V denote an alphabet and V^* the set of all words over V. A 0L-system is a triplet $< V, \omega, P >$, where V is the alphabet, $\omega \in V^*$ a nonempty word called the axiom and $P \subset V \times V^*$ is a finite set of production rules. If a pair (c, s) is a production, we write $c \rightarrow s$. For each letter $c \in V$ there is at least one word $s \in V^*$ such that $c \rightarrow s$. A 0L-system is deterministic, if and only if for each $c \in V$ there is exactly one $s \in V^*$ such that $c \rightarrow s$.

In all of the examples we do not specify the alphabet V explicitly, it consists of all characters that occur in the axiom and the production rules. Also, if a specific rule for a character is not stated, then it is assumed to be the identity, i.e. $+ \rightarrow +$, $- \rightarrow -$, and usually F $\rightarrow$ F.

The included pseudo code implements the string generation procedure and the graphical interpretation. The program first expands the axiom, then expands the resulting string, and so forth. *CleanUpString()* is called next. It removes all

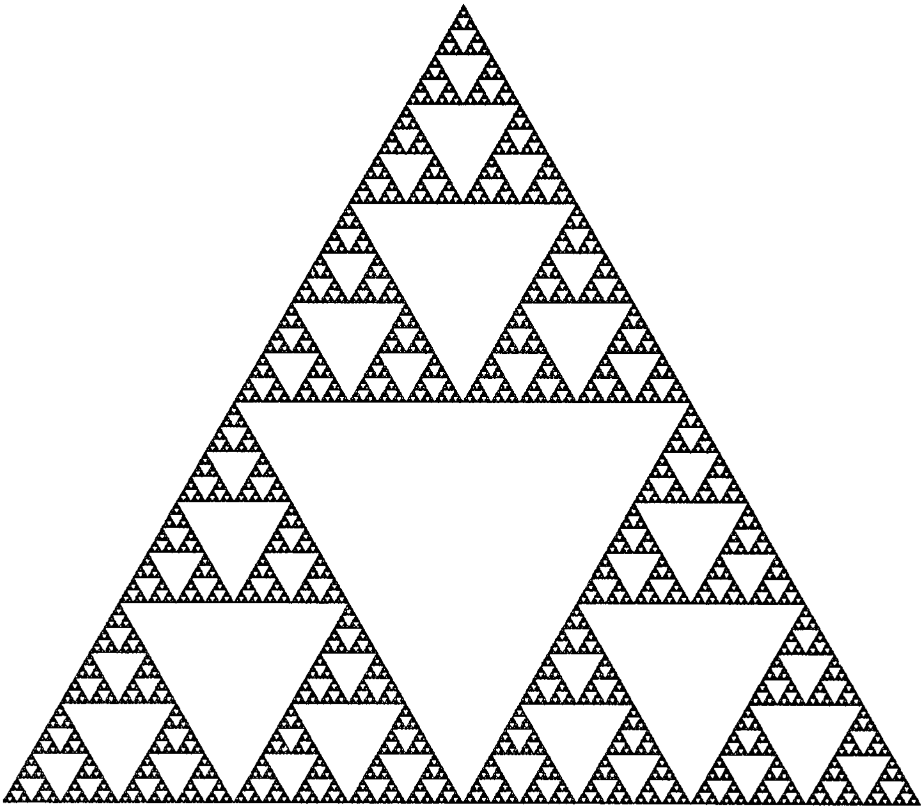

Fig. C.3: The 8-th stage in the generation of the classic Sierpinsky gasket. The 0L-system is given by the axiom "FXF−−FF−−FF", the angle $\delta = \frac{\pi}{3}$ and the production rules F $\longrightarrow$ −−FXF++FXF++FXF−−.

Graphical interpretation of a string : The algorithm which interprets the output string of an L-system (the "turtle") acts upon a character command as follows. We define the state of the turtle as a vector of three numbers denoting the position of the turtle (x-position and y-position) and the direction in which the turtle is heading (an angle). The following character commands are recognized by the turtle :

'F': move one step forward in the present direction and draw the line,
'f': move one step forward in the present direction but do not draw the line,
'+': turn right by an angle given a priori,
'−': turn left by an angle given a priori,
'|': turn back (turn by 180^0),
'[': save the state of the turtle on a stack,
']': put the turtle into the state on the top of the stack and remove that item from the stack.

All other commands are ignored by the turtle, i.e. the turtle preserves its state.

those letters from the string which have significance only for the string reproduction but not for the graphical interpretation. The purpose of this is merely to avoid unnecessary operations. The algorithms *GetCurveSize()* and *TurtleIn-*

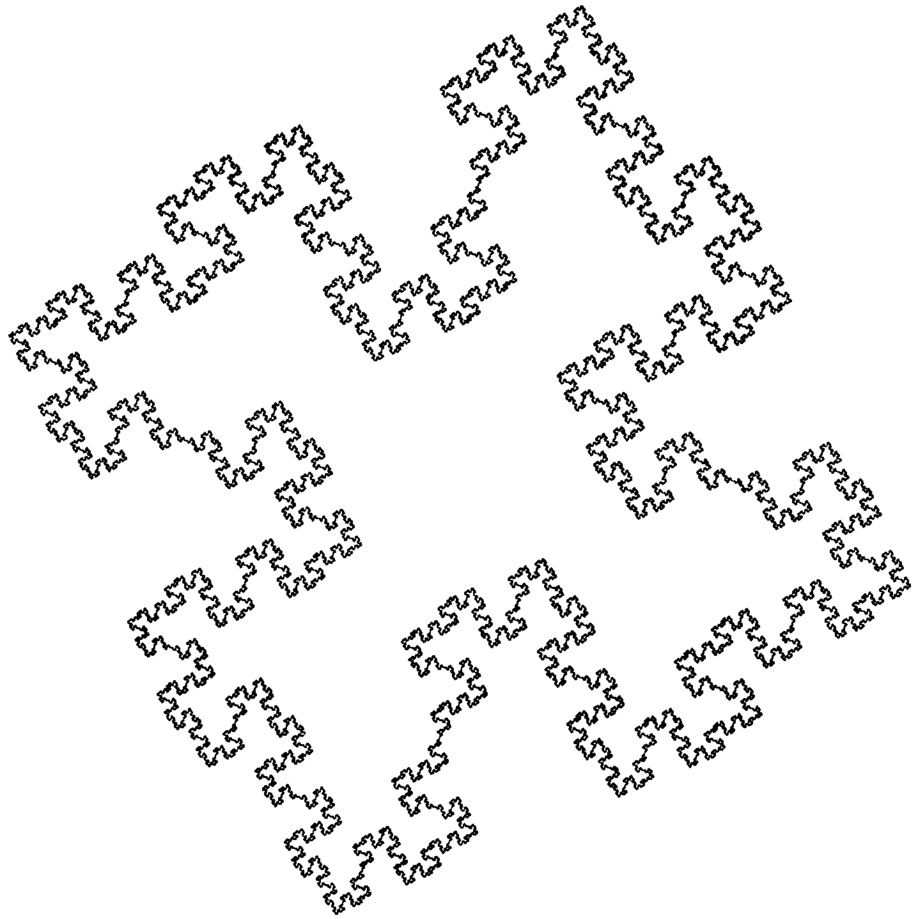

Fig. C.4: The quadratic Koch island (5-th stage) [68]. The 0L-system is given by the axiom "F+F+F+F", the angle $\delta = \frac{\pi}{2}$ and the production rule F $\longrightarrow$ F+F−F−FFF+F+F−F.

terpretation() finally determine the size of the picture and produce the output plot. It should be noted that in the case of a 0L-system an equivalent recursive routine can be written. It would expand one character at a time, according to the corresponding production rule and then recurse for each of the characters of the resulting string until the desired level of detail is achieved. The final characters can immediately be interpreted graphically provided that the overall dimensions of the picture are known a priori.

The routines for string manipulation provided by programming languages differ greatly from one language to another. For our implementation we have borrowed some routines from the C language. However, these should easily be imitated in different languages such as Pascal or Basic.

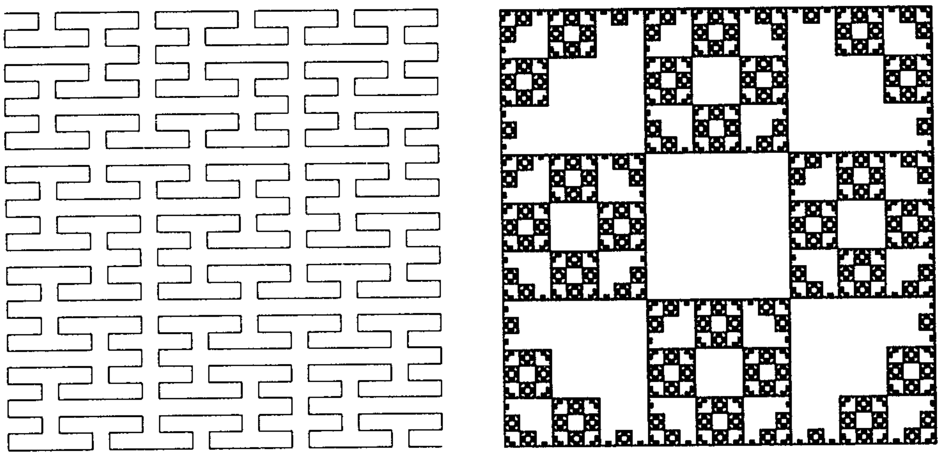

Fig. C.5: On the left the space filling Peano curve (third stage). The 0L-system is given by the axiom "X", the angle $\delta = \frac{\pi}{2}$ and the production rules X $\rightarrow$ XFYFX+F+YFXFY−F−XFYFX, Y $\rightarrow$ YFXFY−F−XFYFX+F+YFXFY. On the right a square Sierpinsky curve (5-th stage). The 0L-system is given by the axiom "F+F+F+F", the angle $\delta = \frac{\pi}{2}$ and the production rule F $\rightarrow$ FF+F+F+F+FF.

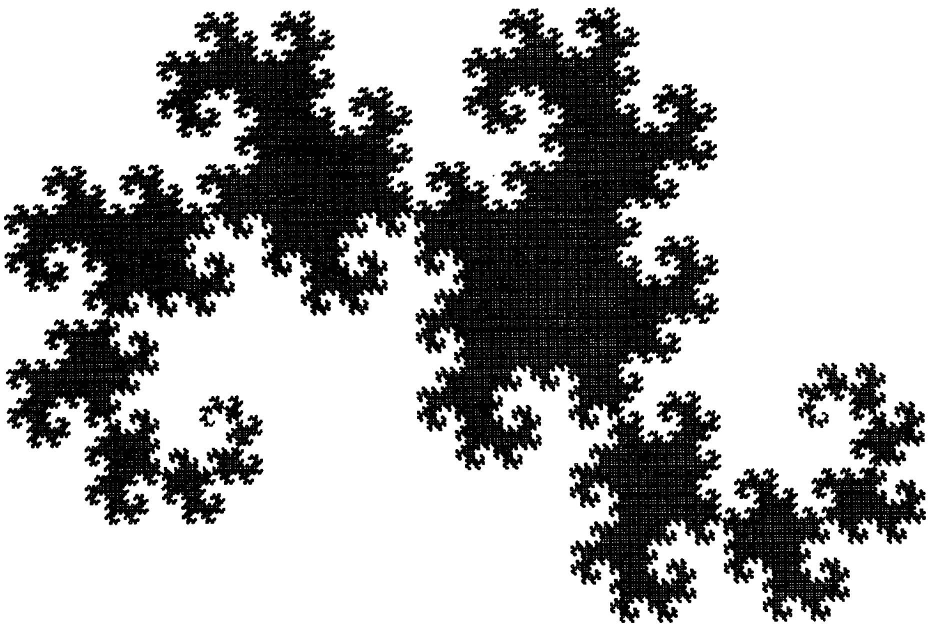

Fig. C.6: Dragon curve (16-th stage). The 0L-system is given by the axiom "X", the angle $\delta = \frac{\pi}{2}$ and the production rules X $\rightarrow$ X+YF+, Y $\rightarrow$ −FX−Y.

The pseudo code is not optimized for speed. Many improvements can be made. For example, characters can be identified with their integer ASCII representation. For each of the 128 characters there can be one (possibly empty) rule stored in an array of rules. In this setup the search for the appropriate rule

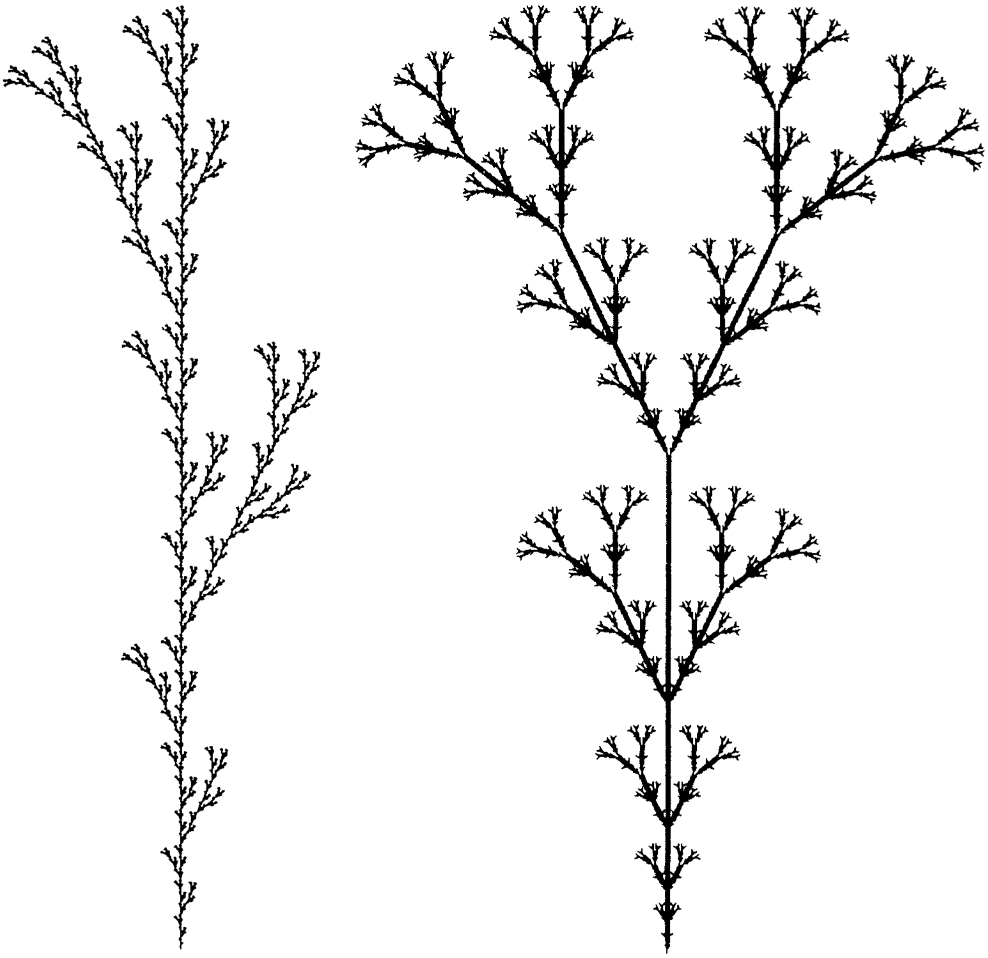

Fig. C.7: Two bushes generated by 0L-systems. The left one (6-th stage) is given by the axiom "F", the angle $\delta = \frac{\pi}{7}$ and the production rule F $\rightarrow$ F[+F]F[−F]F. The right one (8-th stage) is given by the axiom "G", the angle $\delta = \frac{\pi}{7}$ and the production rules G $\rightarrow$ GFX[+G][−G], X $\rightarrow$ X[−FFF][+FFF]FX. **Figure on page 272 :** Bush (5-th stage) with axiom "F", the angle $\delta = \frac{\pi}{8}$ and the production rule F $\rightarrow$ FF+[+F−F−F]−[−F+F+F].

to apply in the expansion of a letter is simpler and faster. As a second remark let us point out that the system routine *strapp()*, which appends a string to another string, is called very often in the algorithm *GenerateString()*. This process generates very long strings, and as a consequence the performance of *strapp()* may decrease dramatically. It is easy to circumvent this problem, but this again depends to a large extent on the choice of the programming language. Thus, we omit these details here.

Our approach to L-systems can be extended in many ways. We give a few ideas (see [88]).

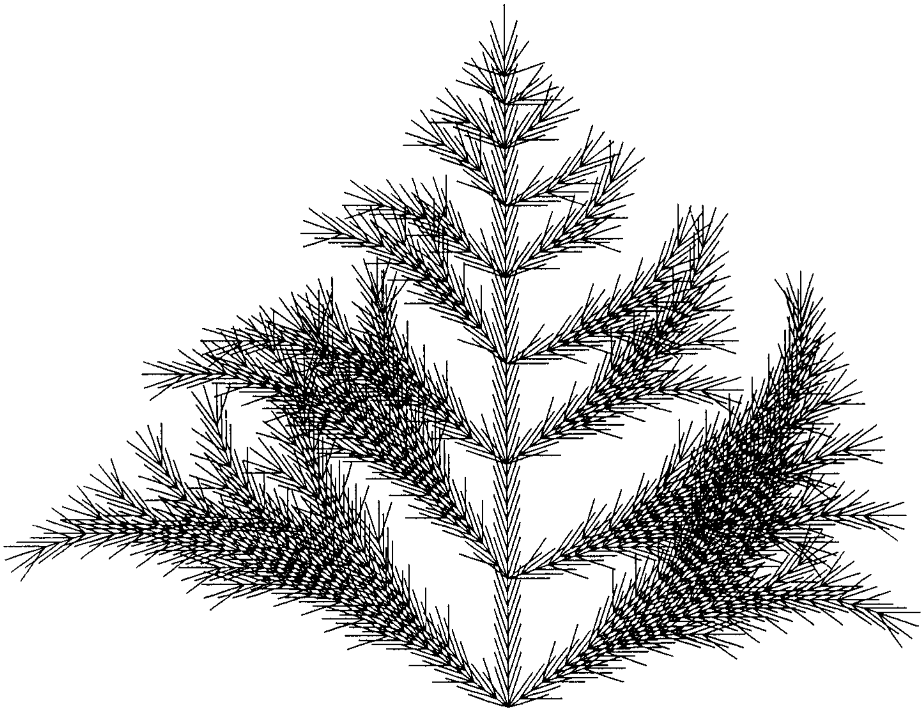

Fig. C.8: Bush (11-th stage) with axiom "SLFFF", the angle $\delta = \frac{\pi}{10}$ and the production rules S $\rightarrow$ [+++G][− − −G]TS, G $\rightarrow$ +H[−G]L, H $\rightarrow$ −G[+H]L, T $\rightarrow$ TL, L $\rightarrow$ [−FFF][+FFF]F.

1. The strategy for the expansion of characters can be modified so that one or another rule may apply according to preceding and succeeding letters. This is a case of context dependence, and the resulting systems are called pL-systems (pseudo L-systems).

2. All objects generated by deterministic L-systems are fixed. There are no variations. In order to obtain several different specimen of the same species, some randomness must be introduced along with a non-deterministic L-system. Thus there are several possible rules to apply in a given situation. Some probabilities should be attached to these rules a priori. Then a random number generator may decide which of the rules is eventually applied. This procedure is very reminiscent of the iterated function systems in Chapter 5.

3. The turtle which interprets the expanded strings may be allowed to learn a wider vocabulary of commands. E. g. parentheses can be used to group drawing commands which define the boundary of a polygon to be filled. Of course, linestyle and color are parameters of interest. Also, the turtle may be instructed to move and draw in three dimensions. Curved surfaces may be considered in addition to polygons.