# Interactive Construction of L-Systems in 2- and 3-Space

CHRISTOPH STREIT
HANSPETER BIERI

University of Berne
Institut für Informatik und angewandte Mathematik
Länggassstrasse 51, CH–3012 Berne, Switzerland
streit@iam.unibe.ch
bieri@iam.unibe.ch

**Abstract.** This paper presents a new approach to the construction of L-systems in computer graphics. We describe methods for the graphical representation of L-system productions and their interactive manipulation in 2- and 3-space. Their use makes it easy, even for unexperienced users, to apply the formalism of L-systems for the purpose of geometric modeling.

## 1 Introduction

*Lindenmayer systems* [Lin68] – or *L-systems* for short – are a powerful tool for the generation of a large variety of geometric objects. However, it is a nontrivial task, in general, to find the descriptions that will lead to the desired models.

An important factor with the design of a tool for the interactive manipulation of L-systems is its simplicity for the user. *Direct manipulation techniques* [Zie91] are widely used. For constructions in 2- and 3-space, a number of methods are known to display and manipulate geometric models. But most of them are not intuitive enough to the user. In this paper we present a new approach by introducing *EdgeTrees* together with an appropriate construction mechanism (in form of a graphical editor), which we expect to improve this situation.

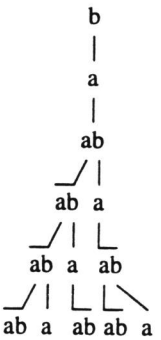
## 2 L-systems

L-systems are a type of *string rewriting systems*. Rewriting is a technique for defining complex objects by repeatedly replacing parts of a string, starting from a simple initial string and using a set of rewriting rules or *productions*. Each production replaces a left-hand side (LHS, predecessor) by a right-hand side (RHS, successor). An L-system definition consists of productions and an initial string, the *axiom*. The production $a \rightarrow ab$, e.g., means that the symbol $a$ is to be replaced by the string $ab$. The LHS of an L-system production consists always of a single symbol to be replaced by the production's RHS. The rewriting process starts at the axiom $\omega$, and the productions are applied in parallel to the appropriate symbols. If no production is provided for a given predecessor, the identity production is assumed to belong to the set of productions.

There are many classes of L-systems. Throughout this paper we will only use deterministic and context-free L-systems which normally are *parametric*. As we will mainly work with simple examples, not much formalism will be needed. For detailed formal definitions and discussions of the properties of various L-systems the reader is referred to [RS74], [RS92] [PL90], [Str93].

The following example of an L-system has only two productions. The string rewriting process is shown for the first five *derivation steps*:

$\omega$: b
$p_1$: a → ab
$p_2$: b → a




## 3 Turtle interpretation of strings

When modeling geometric objects by means of L-systems, a *graphical interpretation* of the derived strings is desired. Several approaches have been proposed to this problem. Prusinkiewicz [PL90] introduced an interpretation based on *turtle graphics* [Ad81] that is widely used nowadays.

The idea consists in associating *turtle commands* with L-system symbols. After generating the desired number of derivations, i.e. executing the L-system, all turtle commands associated with symbols of the resulting string are executed, i.e. the L-system is

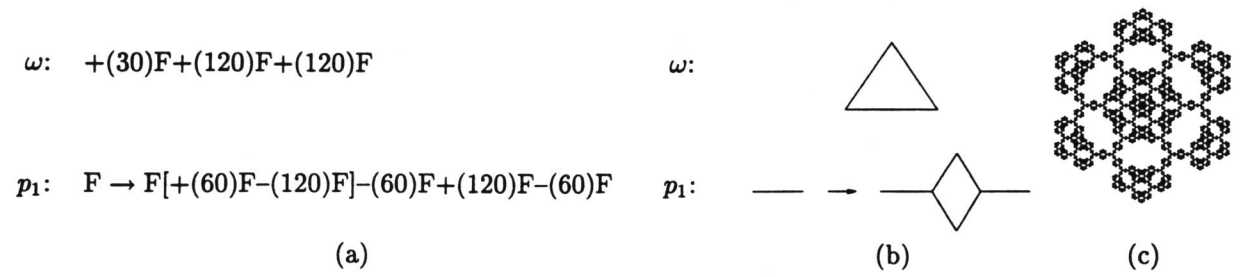



Figure 1: Representation of an extended Koch curve: (a) textual, (b) graphical, (c) the result after 4 derivation steps.

interpreted. It is advantageous with L-systems to keep the interpretation step independent from the execution step. In the resulting string there may also occur *meta symbols*, i.e. with no turtle commands associated. These symbols serve as dummies and allow a modular definition of the productions. E.g. the definition for a tree could include a production describing the trunk: `trunk` → ... `branch`, and productions describing the branches and the leaves: `branch` → ... `leaf`, `leaf` → .... In this example `trunk`, `branch` and `leaf` represent meta symbols. They may be replaced, e.g., when the model is refined. Meta symbols are ignored when executing the turtle commands.

Interpreting an L-system by executing turtle commands results in the representation of a geometric object, as it is desired. It is necessary, of course, to associate the appropriate turtle commands with the symbols used by the L-system.

At any time, a turtle is represented by its current *state*, i.e. its position and orientation. The following turtle commands are common when generating objects in 2-space:

| | |
|---|---|
| F, F($d$) | Move forward a distance of length $d$ and draw the corresponding line segment. |
| f, f($d$) | Move forward a distance of length $d$ without drawing a line segment. |
| +, +($\delta$) | Turn to the left by angle $\delta$. |
| −, −($\delta$) | Turn to the right by angle $\delta$. |
| [ | Push the current state of the turtle onto a pushdown stack. |
| ] | Pop the top element from the stack and make it the current state of the turtle. |

If the commands `F`, `f`, `+` and `-` are used without parameters, a common setting is $d = 1$, $\delta = 90°$. `[` and `]` indicate the beginning and the end of an additional path.

Turtle interpretation of L-systems is also possible in 3-space [Ad81], [PL90]. To control the additional degree of freedom two rotation commands have to be added. For examples, we refer to the following sections.

## 4 Graphical representation of L-systems

The axiom and the productions of an L-system are normally defined *textually*. When using turtle graphics, the result looks as the example in Figure 1(a) shows, i.e. it is rather difficult to understand by a nonexpert. Much more user-friendly is specifying L-systems graphically, as indicated in Figure 1(b). Of course, the result must be the same, in this case an *extended Koch curve*, as shown in Figure 1(c).

In the following, we will present a new graphical representation which has proved to be surprisingly user-friendly, in particular with regard to complex applications in 3-space. For as users we would like to 'play' interactively and comfortably with L-systems, i.e. to modify their definition and to get quickly a graphical representation of the result. The main tool we propose for this purpose is the *EdgeTree* which will be introduced in the next section.

### 4.1 The EdgeTree

The *EdgeTree* is a new graphical tool for the interactive specification of L-systems. It is also a (simple) data structure.

Figure 2 shows the ordinary textual representation of the RHS of a production and the equivalent graphical representation as an EdgeTree: Each `F`-command is replaced by a line segment, and each `f`-command by a dotted line segment. These line segments are oriented. The position of the turtle after a move is indicated by a small white square. The meta symbol `Leaf` is represented as a marker attached to the representation of the command to which it belongs. In the same way, any production using turtle

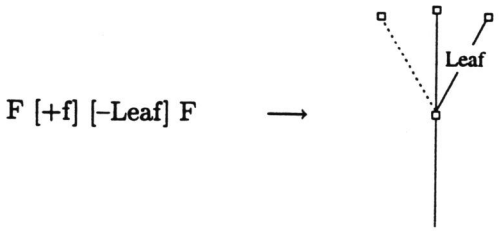



Figure 2: A textual turtle representation and its corresponding EdgeTree.

commands can be represented graphically by means of EdgeTrees (one for the LHS and one for the RHS), and the same holds for axioms. This turns out to be very practical for the user working interactively.

An EdgeTree can easily be implemented by a data structure, i.e. an *ordered general tree* - which we call EdgeTree, too. The following code segment in C defines the node structure of an EdgeTree:

```
struct EdgeTreeNode {
   EdgeTreeNode* parent;
   EdgeTreeNodeList* children;
   Vector to; /* terminal point of the edge */
   int endMarker;
};
```

The *end marker* contained in the code segment indicates that we have not yet completely defined what an EdgeTree is. We will complete its definition in Section 4.3.

### 4.2 Operations with EdgeTrees

It must be possible to create and modify EdgeTrees interactively. That is, we need a suitable *graphical editor*. There are many operations which appear to be useful when working with EdgeTrees. However, our experience has shown that we only need a very small set of 5 operations in order to construct and manipulate EdgeTrees interactively. They are shown in Figure 3, and it is seen that 3 of them change the topology of the given EdgeTree and the remaining 2 only its appearance on the screen.

### 4.3 Conversions

The EdgeTree turns out to be a very convenient tool to work interactively with L-systems for modeling geometric objects. On the other hand, it is convenient to make use of standard packages for executing the rewriting process of L-systems. These packages work with the textual representation of turtle commands. Hence, it must be possible to convert each representation into the other.

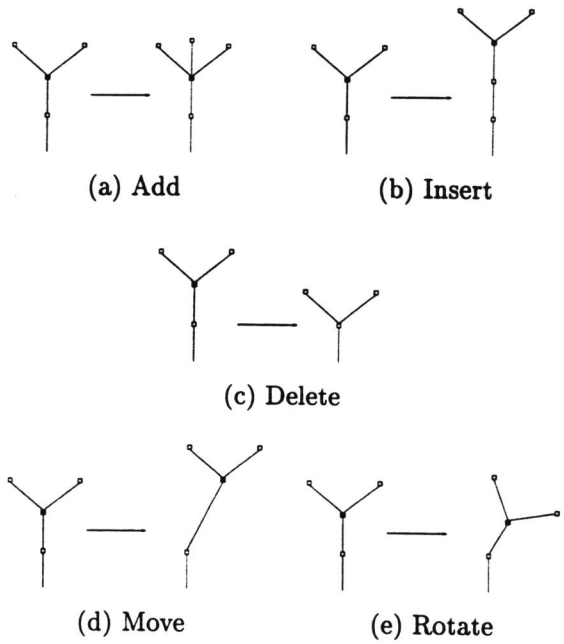



Figure 3: Operations with EdgeTrees.

It is obvious from the foregoing that the EdgeTree belonging to the textually represented LHS or RHS of a production is determined uniquely and can easily be created. The inverse conversion is not so straightforward: It is not hard to see, e.g., that the EdgeTree on the right of Figure 4 corresponds (at

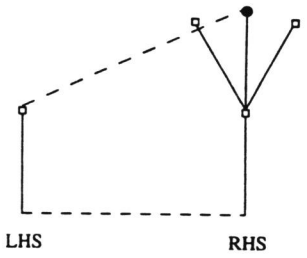



Figure 4: EdgeTree representation of a simple production.

least) to the RHS of each of the following 4 productions. That is, the conversion EdgeTree → textual representation is not unique, in general.

a) F → F [ +(30) F ] [ –(30) F ] F

b) F → F [ [ F ] [ –(30) F ] +(30) F ]

c) F → F [ +(30) F ] [ F ] –(30) F

d) F → F [ F ] [ –(30) F ] +(30) F

Starting at the axiom $\omega = F$ and executing 5 derivation steps in each case, we get 4 different objects, as Figure 5 shows.

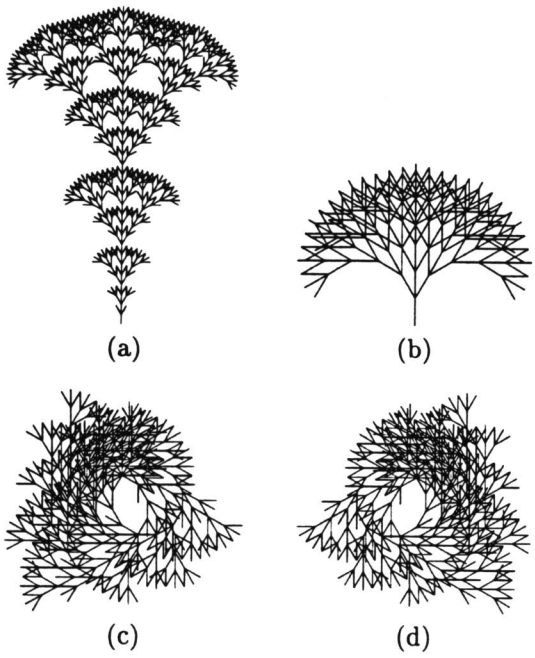



Figure 5: Results after 5 derivation steps.

In order to get a one-to-one correspondence we refine our definition of the EdgeTree by marking one of its nodes as its terminal point. This point will be called *end marker* (see structure `EdgeTreeNode` on page 3). It indicates the final position of the turtle after having moved through the whole tree and will be represented by a small filled circle. It is assumed that the starting position of the turtle is always at the root of the tree. Figure 4 indicates the beginning and the end of a replacement by means of 2 dashed lines. Now, the EdgeTree of the RHS corresponds only to the first of the 4 textual representations.

The following algorithm in pseudo-C transforms an EdgeTree data structure to its corresponding textual representation. The algorithm traverses the EdgeTree in preorder. For each node it visits, turtle commands are generated and `convertToTurtle` is recursively called for all child nodes.

```
void convertToTurtle(EdgeTreeNode* node)
{
  Turtle t;
  generateCommands(node, t);

  for (all children n of node) {
    if (no edge in the subtree of n is
          terminated by the end marker) {
      save turtle t on stack
      print("[");
    }

    generateCommands(n, t);

    convertToTurtle(n);

    if (no edge in the subtree of n is
          terminated by the end marker) {
      restore turtle from stack
      print("]");
    }
  }
}
```

The function `generateCommands` determines the rotation and move commands for the graphical representation of the edge just visited. It has to distinguish between the 2D and 3D case. For the 2D case one rotation command suffices, whereas in 3D two rotation commands are necessary.

Obviously, our algorithm assumes that the end marker always resides in the last branch visited during the traversal. This can easily be ensured by first executing a normalization step, as follows:

```
void normalize(EdgeTreeNode* node)
{
  for (all children n of node)
    if (subtree of n contains the end marker) {
      move n at the end of the children's list
      normalize(n);
    }
}
```

Figure 6 illustrates how our algorithm finds the textual representation of the RHS EdgeTree in Figure 4 starting from its data structure. Numbers show in which order the edges are traversed, and the following list indicates how the required turtle commands are associated by `convertToTurtle`.

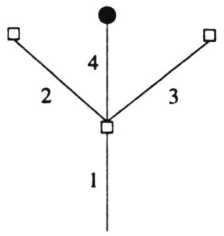



Figure 6: An example illustrating the execution of `convertToTurtle`.

1. Edge 1 is visited and the corresponding subtree contains the end marker
   → F
2. No end marker found for edge 2
   → [-(45)F]
3. No end marker found for edge 3
   → [+(45)F]
4. Edge 4 has the end marker attached
   → F

## 5 The editor GEdit

The EdgeTree data structure together with the operations shown in Figure 3 form the kernel of our graphical editor *GEdit* for the interactive creation and manipulation of L-systems. When designing GEdit, we had especially the inexperienced user in mind and tried to make 'playing' with L-systems in 2- and 3-space for him (or her) intuitive and easy. We implemented direct manipulation techniques [Zie91], above all a new method for constructing in 3-space (cf. Figure 9).

GEdit consists of the following three parts:

1. The *main window* (cf. Figure 7). It displays the current L-system definition and is the starting point for all subsequent manipulations.
2. The *axiom editor* (cf. Figure 10). It allows the interactive construction of a single EdgeTree in 2- or 3-space. Our new technique to make constructions in 3-space easier, is included. Switching from the plain 2D view to the perspective 3D view is possible without restrictions.
3. The *production editor* (cf. Figure 8). It displays the EdgeTrees representing the LHS and the RHS of a production and allows to manipulate interactively both of them.

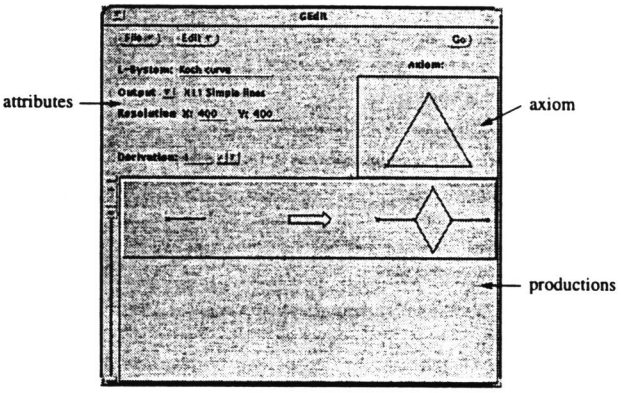



Figure 7: The main window of GEdit.

Although the user may work either with 2D or 3D EdgeTrees, their internal representation is always 3D. This offers the advantage that the more appropriate mode can always easily be chosen. Even switching between the two modes is possible at any time.

Some of our figures (e.g. the RHS in Figure 8) look rather like graphs than trees. The reason is, of course, that turtle positions belonging to different paths may coincide. This indicates that also structures more general than trees can be modeled using GEdit.

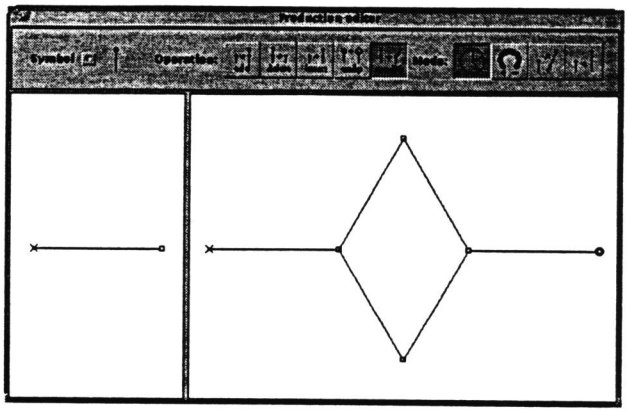


Figure 8: Interactive manipulation of a single production.

## 6 Construction in 3-space

As we have already emphasized, we intended to design an editor that should be very easy to use. Therefore, we considered the known methods for constructing interactively in 3-space, i.e. the *multi view techniques* [FvDFH90], not appropriate enough. Our approach uses a combination of the 3 *orthographic projections* ground view, front view and side view and a *perspective projection* in order to construct and manipulate EdgeTrees in 3-space (cf. Figure 9). New is that these 4 views are combined within one representation, so that the relations between them are more intuitive to the user. The actual construction process uses only the 3 orthographic views (this appeared to us more natural), but the topological operations, i.e add, insert and delete (cf. Figure 3), may also be performed using the perspective view.

Figure 10 shows how the axiom editor of GEdit allows constructions in 3-space by providing the 4 views incorporated in a cube, as Figure 9 explains. This cube – and the construction it contains – can arbitrarily be rotated, which allows the user to comprehend the 3D situation more quickly. This rotation has been implemented applying the *rolling ball technique* [Han92], which makes interactive 3D rotations possible by using 2D control devices. A good alternative could be the use of the arcball technique as introduced by Shoemake [Sho94].

The production editor, too, allows to use these techniques for constructions in 3-space (cf. Figure 11).

## 7 Examples

Figure 11 shows the definition for a simple L-system in graphical form. The EdgeTree representing the RHS of the only production is ternary. This example has been taken from [PL90], but it has been modi-

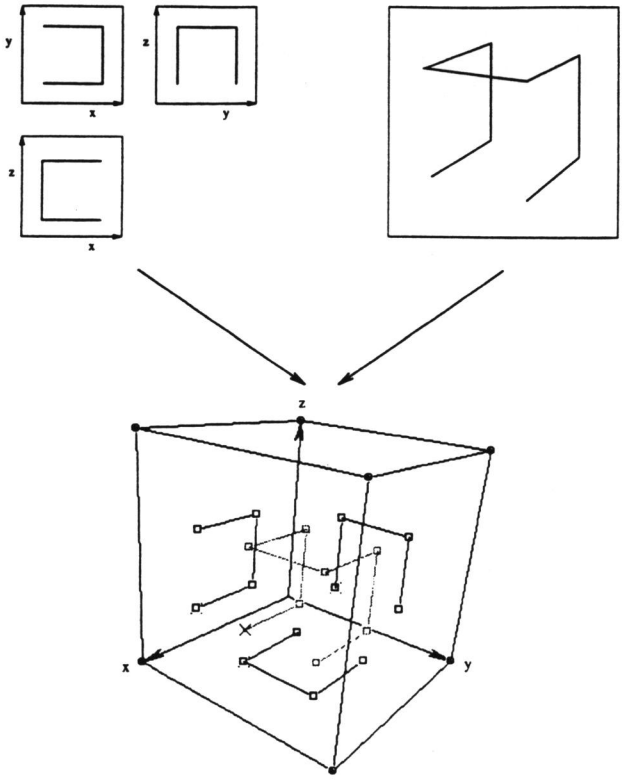



Figure 9: The combination of 3 orthographic and 1 perspective projections in a single representation.

fied to get just one production. Figure 12(a) states the corresponding textual representation. (The not yet explained command $/(\delta)$ rotates the turtle in 3-space around the axis of its movement by the angle $\delta$ (counterclockwise). $l$ denotes a length parameter.[1]) Figure 12(b) shows the result after 6 derivation steps.

Figure 13 shows the definition of the L-system for generating the picture in Figure 14, i.e. a branch with leaves. The final picture has been obtained by means of an additional rendering step using the software Rayshade.

## 8 Conclusion

We have presented techniques and algorithms for the interactive construction of L-systems in 2- and 3-space. By using a graphical representation for the axiom and the productions, the task of L-system construction can be considerably simplified. Our interactive approach encourages the user to *play* with L-system productions, and as a result geometric objects can be generated which often turn out to be very surprising.

[1]Our implementation uses parametric L-systems [Lin74] which allow the association of any number of attributes with L-system symbols. The values of these attributes may be computed as required.

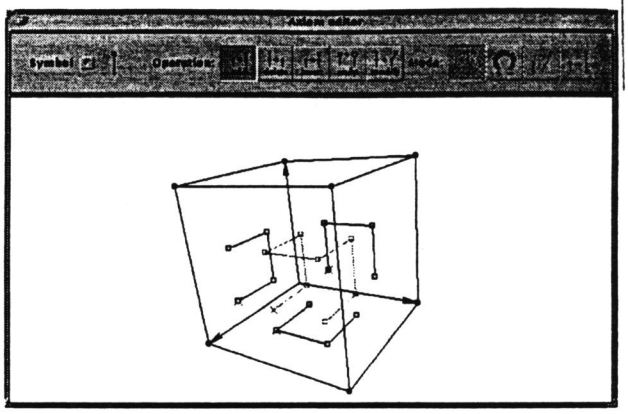


Figure 10: The manipulation of EdgeTrees in 3-space.

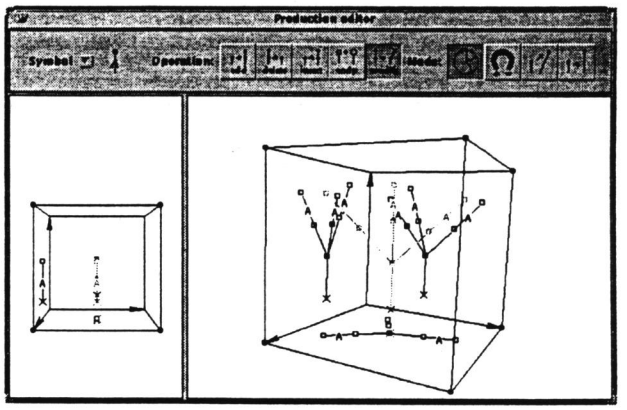


Figure 11: The LHS and the RHS of the production of the tree example.

$\omega$ : F(400) A(100)
$p_1$: A($l$) → F($l$) [ /(135) +(36) F($l$) A($l \cdot 0.93$) ]
[ /(-44.5037) +(36) F($l$)
/(50.5403) A($l \cdot 0.93$) ]
/(-152.005) +(36) F($l$)
/(66.7288) A($l \cdot 0.93$)

(a)

(b)

Figure 12: (a) Turtle graphics definition for trees. (b) The tree generated by 5 derivations steps.

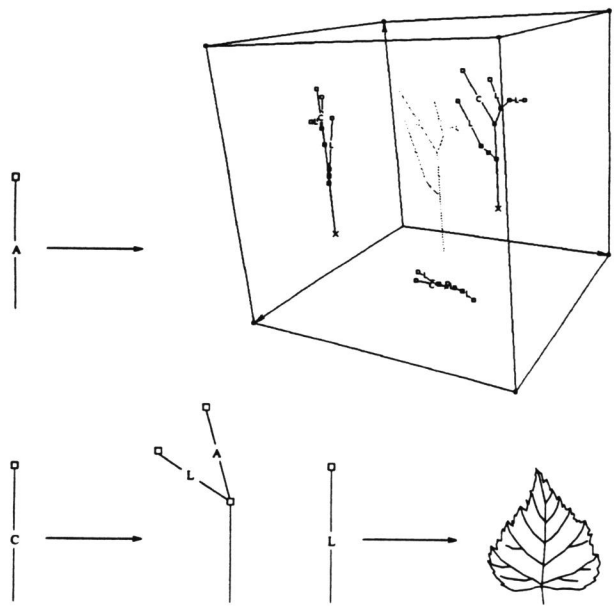



Figure 13: The 3 productions for generating branches with leaves.

Figure 14: The resulting branch with leaves after 7 derivation steps.

## References

[Ad81] H. Abelson and A.A. diSessa. *Turtle Geometry.* The MIT Press, 1981.

[FvDFH90] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1990.

[Han92] A.J. Hanson. The Rolling Ball. In D. Kirk, editor, *Graphics Gems III,* pages 51–60. Academic Press, 1992.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology,* 18:280–315, 1968.

[Lin74] A. Lindenmayer. Adding continuous components to L-systems. In G. Rozenberg and A. Salomaa, editors, *L Systems,* pages 55–68. Springer-Verlag, 1974.

[PL90] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, 1990.

[RS74] G. Rozenberg and A. Salomaa, editors. *L Systems.* Springer-Verlag, 1974.

[RS92] G. Rozenberg and A. Salomaa, editors. *Lindenmayer Systems.* Springer-Verlag, 1992.

[Sho94] K. Shoemake. Arcball rotation control. In P. Heckbert, editor, *Graphics Gems IV,* pages 175–192. Academic Press, 1994.

[Str93] C. Streit. Modeling with Lindenmayer-Systems in Computer Graphics (in German). Master's thesis, University of Berne, Institut für Informatik und angewandte Mathematik, Berne, Switzerland, 1993.

[Zie91] J.E. Ziegler. Direct manipulation techniques for the human computer interface. In *Eurographics Seminars, Advances in Computer Graphics VI,* pages 421–448. Springer-Verlag, 1991.