Winfried Kurth

Some new formalisms for modelling the interactions between plant architecture, competition and carbon allocation

4th Workshop on Individual-based Structural and Functional Models
in Ecology, Wallenfels, 25.–28. 9. 1996,
Bayreuther Forum Ökologie (in press)

Some new formalisms for modelling the interactions between plant architecture, competition and carbon allocation

Winfried Kurth
Universität Göttingen
Institut für forstliche Biometrie und Informatik
Büsgenweg 4, D–37077 Göttingen
wkurth@ufobi4.uni-forst.gwdg.de

Abstract:
Dynamical simulation models of tree growth and morphology were developed to address questions about the interrelations between tree structure, functions and environmental conditions on a mesoscale level. The modelling approach has a formal basis in stochastic growth grammars (i.e., extended L-systems), which can be interpreted by the software GROGRA (Growth Grammar Interpreter) yielding time series of attributed 3-D architectural structures representing trees. With appropriate extensions of this string rewriting language (sensitive functions, arithmetical-structural operators, local registers) it is possible to represent environmental control of shoot growth and some simple allocation strategies, and to reproduce typical competition effects in tree stands qualitatively. The new GROGRA extensions which enable these results are completely documented.

Keywords: tree growth, plant architecture, competition, allocation, sensitivity, L-systems, growth grammars, GROGRA

1. Introduction

In the framework of the Forest Ecosystem Research Centre Göttingen (FZW), a subproject on morphological modelling of tree growth was initiated. Its main emphasis lies on exploring the relevance of the spatio-structural aspect on a mesoscale range (from plant organs to small groups of trees) in forests, which was widely ignored in tree and stand growth models and ecosystem models of the last decades. Essential processes in forests like competition for light and space, water flux, carbon allocation and, e.g., bending under mechanical load, which all have strong impact on the growth of single trees as well as on ecosystem dynamics, take place on this mesoscale and are intimately related to 3-dimensional structures.

Simulation models of plant architecture were developed for the purposes of visualization (Jaeger & Reffye 1992, Prusinkiewicz & Lindenmayer 1990), in agronomical projects (Goel *et al.* 1991, Reffye *et al.* 1995b), for botanical-morphological research (Bell 1976) as well as for the simulation of light interception (Dauzat 1993, Kranigk *et al.* 1994) and water flow (Früh 1995, Kurth 1994) in tree crowns. However, most of these models were "purely structural" in an exclusive sense: They ignored external influences and physiological constraints on growth and therefore represented an equally unbalanced concept of reality than the typical physiological or stand-level process models (like Bossel 1994) which ignored (to a great extent) the structural aspect. It was only in the last years that attempts were made

to reconcile structure and process in the framework of integrated plant simulation models (List *et al.* 1994, Reffye *et al.* 1995, Perttunen *et al.* 1996, Breckling 1996, Měch & Prusinkiewicz 1996).
One purpose of these modelling efforts is to organize knowledge concisely. The other purpose is to generate researchable questions — questions of apparent relevance for forestry.
In contrast to most of the above-cited architectural plant models, which were implemented in an ad-hoc manner in some standard programming language, the approach presented here is based on a *formal specification language*, the language of *sensitive growth grammars* (Kurth 1994), which forms an extension of the well-known Lindenmayer systems (Prusinkiewicz & Lindenmayer 1990). Rule systems for growth written in this specification language can be read and interpreted by an appropriate generic software shell named GROGRA (Growth Grammar interpreter). The growth grammar syntax and semantics as well as a thorough description of the program functionality were given in Kurth (1994).
However, to deal with the problems arising in connection with competition and carbon allocation in trees (and also for some morphological subtleties), certain extensions to the growth grammar language had to be made, which are listed in the Appendix of this article in an (as far as possible) systematic way. The most important of these extensions was the introduction of local register variables, whose relevance is explained along with some examples in the Sections 4 and 5 below. For the reader not familiar with the growth grammar approach, Sections 2 and 3 present some basic knowledge and an introduction to sensitive grammars which were already treated in Kurth (1994) and which are fundamental for the new extensions. The inclusion of these parts makes this paper rather self-contained. In Section 6, some perspectives and open problems are briefly discussed.

But why did we choose such a rather formalistic approach to growth and architecture modelling instead of utilizing some of the above-cited ad-hoc models? The main reasons for working with grammars are the following:

- The reachable level of universality is probably higher when a rather general language is chosen as the basis for modelling,
- models gain a higher transparency when they are described in an exact specification language designed for that purpose — in contrast to the situation when a lengthy computer source code in some general-purpose programming language has to be analyzed to understand a model completely,
- the specifications are normally shorter and more elegantly expressed when a formal language is used,
- like in the formalisms of mathematical physics, the essential causalities and laws can better be separated from unimportant boundary conditions, and can be more precisely stated when an exact formal language is at hand,
- some general theories and tools for formal languages become possibly applicable.

Another important aspect is the consistency of growth models with botanical knowledge. This is a requirement which cannot be assured by the use of formal languages and which can also be fulfilled in software systems not relying on an explicit formal basis, like AMAP (Reffye *et al.* 1995b). But a clear formal specification can help to ensure botanical validity; some botanical rules can directly be encoded as growth grammar rules.

However, it has to be emphasized that — in the opinion of the author — the language of growth grammars in its present status represents only an intermediate step on the way towards a general language for the description of pattern and process in plant growth, and that transparency as well as formal elegance certainly need further improvement. This will also become obvious from the examples presented in Section 5 below.

2. Basic facts about growth grammars

The specification of structural development can be conceived as being stratified in three levels of formal description, each with its own ideas, mathematical background and symbolization (Kurth 1994):

(1) The specification of a structure at a fixed moment in time (statical description). This level of specification can be further differentiated into
(1a) topology,
(1b) geometry,
(1c) additional, non-geometric attributes.
In the growth-grammar approach all these specifications are achieved by the so-called *turtle geometry* (Abelson & diSessa 1982), an interpretation method relating command strings (input) to geometrical structures (output).

(2) The specification of how these structures develop in time. This is done by *rule systems* operating in a parallel manner on *strings* (which are in turn interpreted in a geometrical manner on level 1). We speak of Lindenmayer systems (*L-systems*) in the narrower sense.

(3) The control which of several applicable rules is actually applied in a given situation. This control is carried out in the framework of *growth grammars* with the help of features like *stochastic rules*, *random variables*, *parametrization*, *conditions*, *two-phase* rule application, *sensitivity* and *local register handling*.

Some of these specification tools and features are only briefly sketched here, some are explained in more detail below.

2.1. Turtle geometry

When the dynamical aspect is neglected, the objects which are created by the GROGRA software are simply *branching structures* in 3-dimensional space which are built up of *cylindrical units* — each with its own length and diameter — which can bear additional, non-geometric *attributes* (standing e.g. for colour or assimilate content). We speak of *(geometrical) structures*. The cylinders are called *elementary units* and normally have the meaning of botanical *growth units* or *shoots*. (In some applications concerned with a more detailed level of resolution in plant morphology they can also stand for *internodes*, see Kurth (1996) for examples). Each unit has at most one *mother unit* (supporting unit) from which it emerges.
Turtle geometry is a formal tool to describe such structures. It can be extended to include further geometrical possibilities like surface constructions or simply other fixed geometrical objects additional

to cylinders, but these possibilities are currently not implemented in GROGRA 2.7. Its underlying philosophy is that there is a device, called the *turtle*, which can be told by *commands* to move, to change its orientation in space and other attributes, and *to construct elementary units with given attributes* while moving. The *state* of the turtle is a data record containing all relevant informations on its position, orientation, and all attributes to be used in the next construction step. More precisely, the state variables of the turtle include:

$P = (p_x,\ p_y,\ p_z)$, a vector specifying the *position* of the turtle in space,

$H = (h_x,\ h_y,\ h_z)$ (= *head*), a vector of unit length specifying the direction of the turtle's next move,

$L = (l_x,\ l_y,\ l_z)$ (= *left*) and $U = (u_x,\ u_y,\ u_z)$ (= *up*), two vectors of unit length, orthogonal to each other and to H, specifying directions to be considered as "left" and "upwards", respectively, with respect to the turtle,

ℓ, a real number specifying the *length* of the next move and at the same time the length of the next elementary unit to be constructed,

d, a real number specifying the *diameter* of the next elementary unit to be constructed,

v, a real number specifying a *vertical tendency* which is to be obeyed when a special command, the *RV* command, is given,

n, a real number specifying some non-geometrical parameter of the next elementary unit to be constructed, currently used for *leaf* or *needle mass* or *surface* supported by the shoot,

c, a real number specifying another non-geometrical parameter, currently used for the quantity of *carbon* or *assimilate* stored in the shoot,

u, an integer specifying a parameter which can be interpreted as the *number of internodes* of the shoot,

p, an integer specifying the *colour* of the next elementary unit (in the current GROGRA version the colour is encoded according to the simple EGA colour table with 16 colours, but this could easily be extended),

q, a real number between 0 and 1 specifying the *relative position* along the mother unit where the next elementary unit will emerge.

There are some more state variables which are of a more technical nature, like the branching order, a reference to the mother unit and the distance to the basic (i.e., motherless) unit; see Kurth (1994) for details. The default values of most of the state variables can be specified by the user of GROGRA in the declaration part preceding a growth grammar specification. To this purpose, the commands *set* and *ask* (the latter is for interactive request) are to be used (examples below will demonstrate this possibility). To change the values of the state variables while the turtle is working, several commands are available. The *standard assignment commands* all begin with one of the uppercase letters *L, D, V, N, C, U* or *P* and control the value of the respective lowercase state variable. Each of these letters can be used without an argument or with one (numerical) argument, and — when used with an argument — eventually together with some modifyers. E.g., the commands influencing length (*L* commands) have the following rather systematical syntax and semantics:

L *(without argument)*	sets the state variable ℓ back to the default value,
$L(x)$	gives ℓ the new value x,
$L{+}(x)$	adds x to ℓ
$L{*}(x)$	multiplies ℓ by x.

Furthermore, the commands $L\ell(x)$, $L\ell{+}(x)$ and $L\ell{*}(x)$ (*local* commands) are possible which have the same effect than their counterparts $L(x)$, $L{+}(x)$ and $L{*}(x)$, except that their effect is restricted to the next constructed elementary unit. (The effect of the "normal" commands — without "ℓ" — will last until the next command influencing the same state variable occurs.) The other standard assignment commands are built in an analogous manner, with *L* replaced by *D*, *V* etc.

The *movement* of the turtle in space and the *construction of a new unit* are triggered by the commands *f* (movement by length ℓ without construction) and *F* (movement together with the construction of a new unit). Variants like $F(x)$ (enforcing the unit length x without changing the value of ℓ) are also possible, see Kurth (1994) for full details. A special movement command is @(x), where x is usually a number between 0 and 1. This command is equivalent to $f((x-1)\cdot\ell)$, i.e. the turtle is positioned at a place on the last created unit with the distance x from the basis, expressed in terms of the length of the unit (relative positioning of the turtle). This command is normally used when the position of a lateral bud or a branching node on a shoot is to be specified.

Rotations of the turtle can be specified with respect to the directions H, L and U, which are part of the state of the turtle. E.g., the command $RU(x)$ induces the turtle to turn around the U-axis by x degrees. The position remains unchanged. The commands $RH(x)$ and $RL(x)$ have analogous effects. The whole orthonormal system (H, L, U) is rotated by these commands, but the relation of H, L, U relative to each other remains unchanged.

Another command changing the direction of the turtle is *RV*. In contrast to *RH*, *RU* and *RL*, this command takes an information of global character into account, namely, the *vertical* (geotropic) *direction*. The inclusion of tropism vectors in L-systems was already described in Prusinkiewicz & Lindenmayer (1990) and can be viewed as a first step towards global sensitivity. Technically, *RV* calculates the new H direction of the turtle as the normalized sum of H and (0, 0, $-v$), the latter vector pointing strictly downwards. A sequence of alternating commands *F* and *RV* thus leads to a "bending down" of the generated branch. The greater the value of v, the stronger the bending. Negative values of v are also possible, leading to orthotropic growth.

Two important special turtle commands are the brackets "[" and "]". By their use the occurrence of branching is controlled. When an open bracket "[" is read, the complete actual state of the turtle is stored in a pushdown memory (stack). When the corresponding closed bracket "]" is reached, the turtle stops the construction of the branch which was begun with "[", forgets its current state and adopts again the old state which was actual when the "[" was entered, removing the state from the stack and jumping to the old position. Thus, the commands after "]" will induce the turtle to continue the construction of the "main branch" (see Figure 1a). Another possible interpretation — which leads to the same result — is that the turtle "divides" itself at each open bracket and constructs the side branch (corresponding to the string inside the brackets) and the main branch (string after the closed bracket) in a parallel manner (Figure 1b). This interpretation is closer to the reality of plant growth, where meristems can divide itself and work in parallel, but due to the lack of properly parallel computers, the current implementation of GROGRA works according to the first, sequential interpretation.

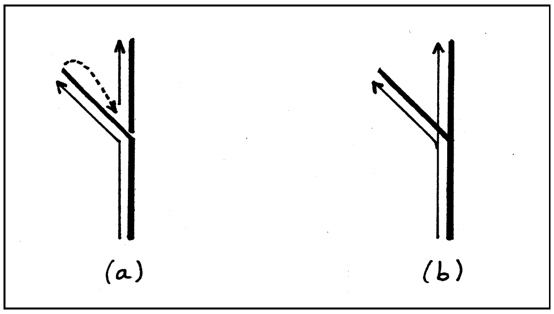


Fig. 1: Sequential (a) and parallel (b) interpretation
of the turtle command sequence *F [RU-45 F] F*.

2.2. L-systems

A classical *Lindenmayer system* (0L system; the "0" stands for "zero context") consists of

- an *alphabet* Σ, consisting of a finite number of *symbols* (e.g. a, b, c),
- a *start string* (or *axiom*) α which is made up of symbols from Σ (e.g., aba),
- a set of *replacement rules*, each of the form

  $$\text{symbol} \longrightarrow \text{string of symbols}$$

  (e.g.: $b \longrightarrow cca$), which are to be applied in parallel to all symbols of a string at time t in order to get a new string at time $t+1$.

We adopt the convention that a symbol for which no rule is applicable remains unchanged.

The rewriting process, i.e. the application of the rules to the given string, will normally be iterated several times. Thus we get a (potentially infinite) *sequence of strings* σ_0, σ_1, σ_2, ..., where σ_{t+1} is obtained from σ_t by application of the replacement rules, and $\sigma_0 = \alpha$. This rewriting process of strings forms the heart of the L-system modelling approach. We refer to the numbers or timesteps 1, 2, 3... as *generations* and call the above-mentioned replacement rules also *generative rules*. Technically, in the GROGRA syntax the arrow " $\longrightarrow$ " of a generative rule is written as "#". The rules are specified in a file, whereas the start string is asked interactively from the user.

But what is the connection between this rewriting system and turtle geometry, i.e. to the specification of 3-D structures representing plants? In the definition of a growth grammar we have to include, additionally to the above-mentioned three ingredients,

- a *geometrical interpretation* of the strings (i.e. a *semantics*) translating strings into spatial structures,

and this interpretation is given by turtle geometry. The complete turtle command set T is thus included as a subset in the alphabet Σ. Symbols contained in Σ but not in T are ignored by the turtle. They can have a meaning in the rewriting process generating new generations of strings, but not in the geometrical interpretation.
The interpretation is applied to each string of the generated sequence $\sigma_1, \sigma_2, \sigma_3, \ldots$, yielding a *sequence of geometrical structures* $S_1, S_2, S_3, \ldots$. Hence we arrive at the situation visualized in Figure 2. Here, the horizontal arrows stand for the development governed by the generative rules, whereas the vertical arrows correspond to the geometrical interpretation. Only the structures $S_1, S_2, S_3, \ldots$ are the output of the GROGRA software. The strings $\sigma_1, \sigma_2, \sigma_3, \ldots$ used for their creation are only written temporarily into an auxiliary file and are (in the current GROGRA version) not the object of further analysis once they have been used to construct the structures.

$$\begin{tikzcd} \alpha \arrow[r] & \sigma_1 \arrow[r] \arrow[d] & \sigma_2 \arrow[r] \arrow[d] & \sigma_3 \arrow[r] \arrow[d] & \cdots \\ & S_1 & S_2 & S_3 & \cdots \end{tikzcd}$$

Fig. 2: Structure generation by a simple
growth grammar (L system with interpretation)

For example, let a stand for a terminal bud of a main axis, b for a bud of a lateral axis, and let *F* have the already explained turtle command meaning, i.e. *F* stands for a *shoot* (growth unit). Then the L-system composed of the two generative rules

$$\begin{array}{lll} a & \longrightarrow & F\,[\,RU45\; b\,]\; a, \\ b & \longrightarrow & F\, b \end{array}$$

gives, when applied to the start word a, the sequence of strings and the corresponding sequence of structures which are depicted in Figure 3 (only the two first developmental steps are sketched).

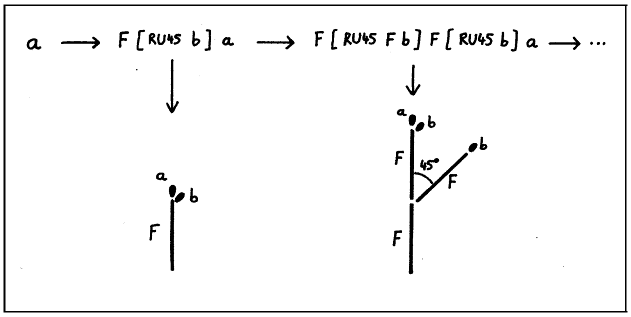


Fig. 3: Structure generation by an example growth grammar.

The buds symbolized by a and b will normally not be visible in the turtle interpretation (if they are not subject to other, interpretative rules; see below), they are depicted as small ovals here to enhance the understanding of the process. Note that *F* remains unchanged in the second step because no rule is applicable to *F*.

2.3. Parametric and stochastic L-systems and variables

In a *parametric L-system* (Prusinkiewicz & Lindenmayer 1990), to each symbol there can be attached a finite list of real-valued parameters, e.g. a(7, 42, -0.5) instead of a. We speak of *modules* instead of symbols in this case. These modules form *parametric strings*, i.e. strings of modules. The L-system rules may now contain formal parameters, e.g.

$$a(x,\ y,\ t)\ \longrightarrow\ b(2*x+y,\ t)\ a(x,\ y,\ t+1)\ c.$$

If this rule is applied to, let us say, $a(-1,\ 5,\ 4)$, we get the parametric string $b(3,\ 4)\ a(-1,\ 5,\ 5)\ c$. Furthermore, *conditions* like $(t = 0\ \&\&\ x > y/2)$ containing the formal parameters may restrict the applicability of a rule.
In the arguments attached to the symbols on the right-hand side and in the condition, variables and constants can be combined in *arithmetical expressions*, using the (slightly extended) syntax for such expressions from the programming language C. Analogously, the *condition* is generally a *logical expression*. (Further details can be found in Kurth 1994.)

In a *stochastic* L-system, there may exist several rules with the same symbol (or module) on the left-hand side, each of them being attributed by a probability. The probabilities of all rules having the same l.h.s. should sum up to 1. In the repeated application of rules, the choice is done randomly with the given probabilities. This implies that the generative process now is no longer necessarily deterministic: The random choice of rules can lead to different sequences $\sigma_1,\ \sigma_2,\ \sigma_3,\ \ldots$ from the same start string α. Syntactically, in GROGRA the probability is written after a "?" at the end of a rule. It can be specified by a floating-point number or by an arithmetical expression. The general form of a growth grammar rule is hence

$$(condition)\ \ l.h.s.\ \ \#\ \ \ r.h.s.\ ?(probability\_expression),$$

where "(*condition*)" and "?(*probability_expression*)" can be omitted.

If several rules apply to a module, and if no conditions and probabilities are involved, the order in which the rules are written down in the grammar file dictates which rule will be applied: It is the first one.

The potential stochasticity of rule applications is further enlarged by the possible *declaration* of variables as "random" (with a specified distribution controlling the outcome of their value) and by their use in expressions occurring in modules on the r.h.s. of rules or in conditions or probability expressions of rules. GROGRA allows the declaration of variables of a multitude of different types in a declaration part preceding the list of rules of a growth grammar. An especially simple form of declaration is that of a *constant*. It has the form

\const *name* *value*,

where *value* is a floating-point number, and the effect is that every appearance of *name* in a position in the rules where a numerical value is permitted is replaced by *value*. The general syntax of a *variable* declaration is

$$\backslash\text{var} \ \ \textit{name} \ \ \textit{specification} \ \ \textit{parameter\_list},$$

where *specification* must be one of several possible keywords which specifies from where the variable gets its value when it appears in a rule which is to be applied. E.g., "uniform" qualifies a uniformly distributed random variable, "normal" one with a normal distribution, "poisson" one of Poisson type etc. (see Kurth (1994) and the Appendix of this paper for a complete list of the possible types of variables). The *parameter_list* consists of one or several numerical values; it can also be empty. How many values are required and what their meaning is depends on the specification. E.g., *var* $x1$ *binomial* 0.3 20 specifies $x1$ as a random variable with binomial distribution, described by the parameters $p = 0.3$ (probability of success of single Bernoulli experiment) and $n = 20$ (number of repetitions).

Attention is to be paid to the situation when a certain random variable appears twice (or even more often) on the r.h.s. of the same rule. E.g., in $a(x1)\ b(x1)$, the argument of a is taken from a random-number generator working according to the specified distribution, and the same holds for the argument of b — with the consequence that both values will normally differ from each other. If b shall have the same value than a, the *memory operator*, symbolized by an underline (_), has to be applied to the second instance of $x1$: $a(x1)\ b(x1\_)$.

It is also possible to declare a variable with specification "function", which makes it refer to one of a predefined set of special functions implemented in GROGRA. The *parameter_list* is $n\ a$, where n is the number (identifyer) of the function and a its arity (number of arguments). Only numerical arguments are counted here, not a possible dependence of the outcome of the function from the generated structure (sensitivity; see below). The arity may be 0. When a variable declared as *function* appears in an expression to be evaluated, it must be followed by a list of a parameters. The possible functions are listed in Kurth (1994) and in the Appendix.

2.4. Two-phase growth grammars

The structure-generating process is further flexibilized by the introduction of a second set of rules (possibly empty), named *interpretative rules* or *rules of the second phase*. Their application intervenes between the generation of a string σ_k and its interpretation. That means, the creation of the structure S_k is delayed until an eventual application of rules of the second phase has transformed σ_k into a string σ'_k. (If the set of interpretative rules is empty or if no such rule is applicable, $\sigma'_k = \sigma_k$ is assumed.) Note that the interpretative rules — in contrast to the "ordinary", generative rules — have no influence on the next generation string σ_{k+1}. Figure 4 summarizes this modified structure generation process, with the vertical double arrows indicating the application of interpretative rules.

Two-phase grammars have proved very useful for abbreviating complex geometrical constructions which stand in no connection with the developmental process modelled by the generative rules. Biologically, we may think of structure-forming processes running on different hierarchical levels and time-scales, one of them handled as taking place "immediately" in the model and thus requiring no generative step but only rules of the second phase. Syntactically, the interpretative rules in GROGRA are distinguished from the generative rules by the use of two double-crosses ($\#\#$) instead of one, fulfilling the function of the arrow of the rule.

$$
\begin{tikzcd}
\alpha \arrow[r] & \sigma_1 \arrow[r] \arrow[d, Rightarrow] & \sigma_2 \arrow[r] \arrow[d, Rightarrow] & \sigma_3 \arrow[r] \arrow[d, Rightarrow] & \cdots \\
& \sigma_1' \arrow[d] & \sigma_2' \arrow[d] & \sigma_3' \arrow[d] & \cdots \\
& S_1 & S_2 & S_3 & \cdots
\end{tikzcd}
$$

Fig. 4: Structure generation by a two-phase growth grammar

2.5. Sensitivity

The most important extension to the classical L-system concept is the inclusion of sensitivity, i.e. the possibility to let a rule application depend on environmental influence (as well as on influences from — near or far — parts of the generated plant structure itself). Here, *context sensitivity* of L-systems like in Prusinkiewicz & Lindenmayer (1990) does not suffice because the signals having impact on the development of a plant do not necessarily come from the immediate context (in the sense of the string encoding) of the plant organ under consideration. Thus, *global sensitivity* (Kurth 1994) is required.

Independently from the approach to sensitive growth grammars presented here, environmentally sensitive L-systems were also developed by the team of Prusinkiewicz (Prusinkiewicz & McFadzean 1992, Prusinkiewicz *et al.* 1994, Měch & Prusinkiewicz 1996, Prusinkiewicz *et al.* 1996). Sensitivity — though not in connection with a formal grammar description — was also included in the AMAPpara plant growth and architecture model by Blaise (1991) and in other modelling approaches partially inspired by ecological or physiological questions, like List *et al.* (1994), Hauhs *et al.* (1995), Gavrikov & Sekretenko (1992), Perttunen *et al.* (1996), Breckling (1996).

The key feature of globally sensitive growth grammars in the GROGRA system is that functional parameters (i.e. variables declared as "*function*") which depend on the last generated geometrical structure are allowed in the generative rules. Hence, in the scheme of Fig. 4 we have to add an influence from S_k on the creation of σ_{k+1} (and thereby on S_{k+1}), indicated by dotted arrows in Figure 5.

Technically, GROGRA enables sensitivity by saving the internal addresses of the elementary units of the last created structure in the order of the associated turtle commands and thereby establishing a link between the string to which the generative rules are applied and the last structure. The unit in the last structure to which a symbol is associated is always the unit which was created by the last preceding *F* command. Examples will follow in the next Section.

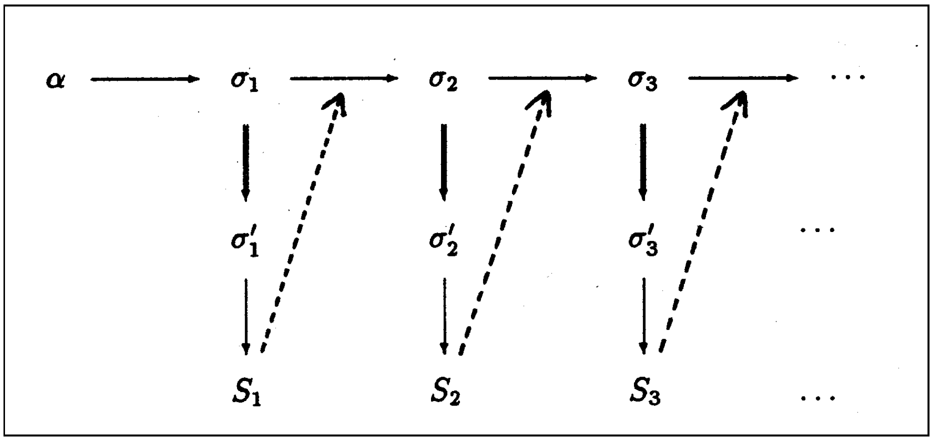


Fig. 5: Structure generation by a sensitive, two-phase growth grammar

3. How to make plants grow sensitively

Principally, all kinds of information from the (local or global) context of a plant part can be used to exert control in the model on the development of that plant part. We will discuss the grammar implementation of such sensitivity and the resulting behaviour of the model for three types of information commonly considered as important for the morphological development of trees and tree parts: orientation in space, density of plant organs in space, and availability of light. The examples presented here are not meant to describe "realistic" plant models (in the sense of yielding numerical values on the same scale and directly comparable to field data), but only to demonstrate some fundamental possibilities in a heavily oversimplified manner.

3.1. Sensing orientation

Table 1 shows the grammar specification of a plant growing in a strictly *epitonic* manner, i.e. buds are only allowed to grow out into lateral shoots if they happen to have an orientation which is nearly vertical and if they are situated on the top side of the supporting shoot (cf. Kurth 1996). Figure 6 shows the developmental steps 7, 11, 17 and 20 of the resulting output produced by the software GROGRA when fed with this sensitive growth grammar. The start word is "$*$" here.

The growth of the main axis is governed by the rule in line 7. The terminal bud, symbolized by t, produces in each time step a growth unit (*gu*), surrounded by a ring of 8 lateral sleeping buds (*bd*) in basal position, and a new terminal bud $t(n+1)$. The parameter of *t* counts the age of the meristem. If this age reaches the value 10, the rule is no longer applicable due to the condition $(n < 10)$, and the axis stops growing. By *RH*(u) a random rotation around the growth direction is enforced (cf. the declaration of u in line 3), such that the lateral buds are not always directed into exactly the same directions after each timestep.

```
1   \var i index,
2   \var e uniform -5 5,
3   \var u uniform 0 360,
4   \var f function 7 0,
5   \set V 0.7,
6   * # t(0),
7   (n < 10) t(n) # RH(u) gu(1,10-n) &(8) < [ @0.2 RH(i*45) RL80 bd(0) ] >
                      RU(e)  RV  L*0.9  t(n+1),   /* terminal growth */
8   (n > 2 && f < 40)  bd(n)  #  L  t(1),   /* outburst of sleeping bud */
9                      bd(n)  #  bd(n+1),   /* further sleeping of bud */
10  (n < k)  gu(n, k)  #  gu(n+1, k),   /* ageing of growth unit */
11  bd(n)  ##  L*0.05  Pl4  F,          /* display of sleeping bud */
12  (n < 7)   gu(n, k)  ##  Dl(n)  F,   /* display of growth unit */
13  (n >= 7)  gu(n, k)  ##  Dl(7)  F
```

Table 1: The sensitive growth grammar *epiton.ssy*

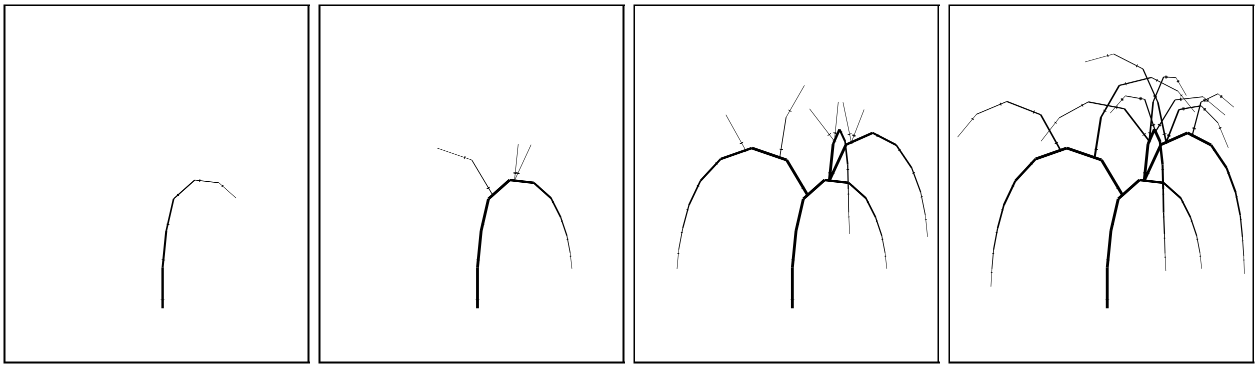


Fig. 6: Plant with epitonic growth habit, produced by GROGRA from the sensitive grammar *epiton.ssy* (see Table 1)

The construction "&(8) $< \ldots >$" stands for a *repetition command* which iterates the part enclosed in $<\ldots>$, i.e. the lateral bud, 8 times. In each iteration the variable i, declared as "index" in line 1, is automatically augmented by 1, which leads to a symmetrical distribution by the command *RH*($i*45$) of the lateral buds in the directions $0°$, $45°$, $90°$, …, $315°$. The branching angle of each lateral bud (and of each branch eventually produced by it) is fixed to $80°$ by the command *RL*80. The further rotation command *RU*(e) in this rule, making use of the random variable e declared in line 2 to vary in the small interval $[-5°, 5°]$, enables small deviations of the growth direction of the axis from the initial vertical direction. Without these deviations, the axis would never hang down, and the lateral buds would never come into the vertical orientation necessary for their outburst.

The geotropic tendency itself ("hanging down") is caused by the operator *RV*. Its strength is specified by the *set* statement in line 5, but it naturally depends also on the deviation from the vertical which the axis has already obtained: the stronger this deviation, the stronger the effect of *RV*.

Furthermore, the length of the produced growth units is steadily diminuished by the factor 0.9 (command $L*0.9$).

The key function introducing sensitivity into this grammar is declared in line 4. The identifying number 7 of this function refers to a routine implemented in GROGRA which calculates the angular deviation of the element associated with the actual symbol (here *bd*) from the vertical. The function f has no numerical argument ("0" in the declaration). If its return value exceeds $40°$, the application of the rule in line 8 is forbidden by the condition "$f < 40$". (The further condition, "$n > 2$", tells us that a certain phase of resting — at least 2 timesteps — must have been passed before a bud can evolve.) If Rule 8 is applied, a terminal bud ($t(1)$) is the result, which is capable of producing a shoot according to Rule 7 again, with the initial shootlength put to the default start value by the "*L*" command without arguments. Botanically, this lateral branch is nothing else than a simple case of reiteration.

If Rule 8 is not applicable, the next rule (line 9) will be applied, which has just the effect to increase the age of the sleeping bud. A similar ageing of already created growth units is executed in the first parameter of *gu* in line 10. This age parameter n controls the thickness of the growth unit in the interpretative rule in line 12 ($D\ell(n)$). However, the application of Rule 10 depends also on the second parameter of *gu*, k, which is decremented along an axis by Rule 7. If n exceeds this threshold k, the growth unit stops its growth in diameter. This prevents distal parts of axes which already have stopped to grow in length from becoming unnaturally thick. A further limit for diameter growth is given by the condition $n < 7$ of Rule 12, together with Rule 13, fixing a final thickness of 7 length units.

As can be seen from Fig. 6, only a small number of lateral buds (displayed as very short lines specified in line 11) comes into a position which enables the application of the sensitive rule 8, i.e. the creation of a reiterated axis. However, after some further timesteps there emerge again reiterations in an epitonic manner from the few new axes, so that a complex of reiterations begins to develop (see right part of Fig. 6). This can often be found in shrubs.

A dependence of shoot growth on spatial orientation is conjectured to occur in many woody plants and forms also an essential part of the control of morphology in the MADEIRA model (List *et al.* 1994).

3.2. Sensing density

In an ordinary, non-sensitive L-system, there is no mechanism foreseen to prevent shoots to grow arbitrarily close to each other, or even to grow "into" each other. To exclude such physically unnatural behaviour, some sensitive function is necessary which detects other shoots (or other kinds of obstacles) in the neighbourhood of a plant organ. In GROGRA, the function with identification number 2 fulfills this task. It can be used with a numerical argument which specifies a maximal length of elementary units which are to be considered as "irrelevant" during the search for neighbouring shoots — e.g. to exclude the buds themselves or their components from having influence on their outburst. The return value of the function is the distance to the nearest other elementary unit (having a length greater than the specified argument). The unit under consideration and its mother unit are excluded from the possible set of nearest neighbours in this search. (To be precise, it is the distance to the nearest unit *end* or unit *basis* which is returned in the present GROGRA version, but a "true" distance to the cylindrical unit could equally well be calculated.)

Table 2 shows a grammar in which this function is applied to prevent two 2-dimensional tree crowns from growing into each other. The two trees are initiated (with a 1-step time delay between them) in line 6. The sensitive rule is to be found in line 10. It is only applied — enabling an elongation of the branch initiated by *b* by one further growth unit — if the distance $f(50)$ to the nearest unit longer than 50 length units is larger than $mdist$, i.e. if a ball with radius $mdist$ around the shoot tip under consideration is free of "large" shoots. The value of $mdist$ is demanded from the user (line 3) and held in a *global register* (declared in line 2) during the simulation. The register variable, once declared, could be referred to by "$mdist$" in every arithmetical expression and in every rule.

A further condition appears in line 10, involving the variable z declared in line 4 as having the type "*zcoordinate*". This demonstrates another kind of sensitivity which will only be briefly mentioned here: z returns the height (z coordinate) of the actual shoot tip in a global Cartesian system. The condition "$z > 90$" prevents the hanging branches from growing into the soil.

The further details of this and of the next grammar example are not discussed here, but the system "*carb2d.ssy*" below (Table 4), which has many similar features, will be commented thoroughly.

```
1  \var f function 2 1,
2  \var mdist register 1,
3  \ask l1 minimal distance?,
4  \var z zcoordinate,
5  \set V 0.2,
6  * # P2 D1 [ L120 a0 tbd(8) ] RU90 f450 RU-90 [ L100 a1 tbd(8) ]
          RU-90 f1250 RU180 P6 F1950,
7  a1 # a0,
8  a0 # i(1.8,0) F [ RU40 L70 b(0) tbd(8) ] [ RU-40 L70 b(0) tbd(8) ] a0,
9  i(s,t) # i(s,t+1),
10 (f(50) > mdist && z > 90) b(t) # i(0.8,0) RV RH(90+t*180) F
                                   [ RU30 tbd(6) ] RH180 b(t+1),
11 i(s,t) ## Dl(1+s*t),
12 tbd(x) ## P6 D0.5 L(x) RU-90 f*0.5 RU180 F RU-104.5 F*2 RU-151 F*2
```

Table 2: The sensitive growth grammar *densi2d.ssy*

The structure generated from the density-dependent grammar *densi2d* after 15 timesteps is shown in Figure 7. The specified minimal distance was 60 here. One has to note that the growth units are not completely prevented from close contact to each other because the distance check is made *before* they grow.

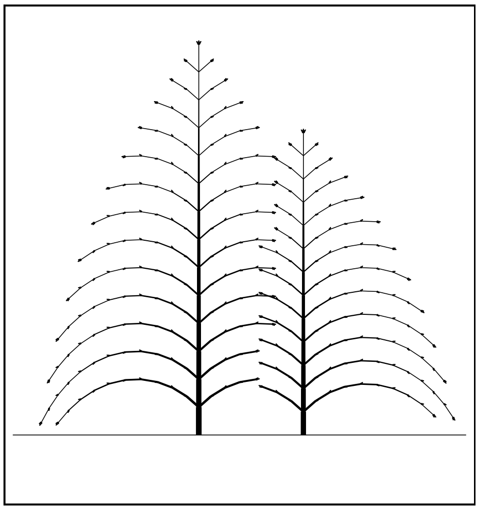


Fig. 7: Two trees with density-dependent shoot growth, produced by GROGRA from the sensitive growth grammar *densi2d.ssy* (see Table 2) after 15 steps

3.3. Sensing overshadowing

The perhaps most important type of sensitivity is the dependence of shoot growth upon light. It is possible to simulate the distribution of light in an artificial canopy very accurately with refined methods (Govaerts 1996), however, for our purposes a strongly simplifying concept will suffice, the notion of the *shadow-relevant cone*. That means, for the estimation of the light conditions of a certain shoot, a cone is considered with its axis of symmetry having vertical direction, with a fixed opening angle and with its apex exactly in the shoot tip under consideration. All other shoots or obstacles lying inside or intersecting this cone are considered to have potential influence on the amount of light which is received by the considered shoot, others have not (Figure 8).

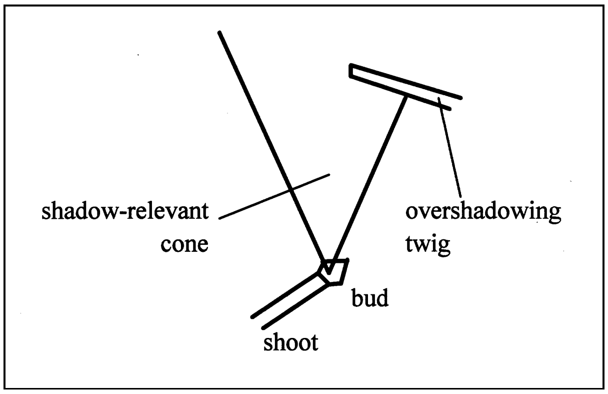


Fig. 8: The shadow-relevant cone of a shoot

Former investigations have yielded evidence that there is a good relation between the leaf area located in a shadow-relevant cone and the potential photosynthetic productivity at its apex (Pfreundt & Sloboda 1996), hence the notion of shadow-relevant cone seems to be justified.

```
1  \const nmax 5,
2  \const ang 40,
3  \var f1 function 1 1,
4  \var f3 function 3 0,
5  \var le1 uniform 80 120,
6  \var le2 uniform 60 80,
7  \var su uniform 0.7 0.9,
8  \set V −0.2,
9  * # P2 D1 [ f0.001 a(-4) tbd ] RU90 f500 RU−90
          [ f0.001 a(0) tbd ] RU−90 f1000 RU90
          [ f0.001 a(−8) tbd ] RU−90 f750 RU180 P6 F2500,
10 (k < 0)                      a(k) # a(k+1),
11 (f1(30)>ang && k >= 0) a(k) # P2 L(le1) seg(1.8, 0, k)
        [ @(su) RU70 D1 b tbd ] [ @(su) RU-70 D1 b tbd ] f0.001 a(k+1),
12 (f1(30)>ang)        b # P2 seg(0.8,0,-1) L(le2) RV RH90 f0.001 b,
13 (f3 >= 5)           seg(s,t,u) # seg(s,t+1,u),
14 (f3 < 5 && u < 0) seg(s,t,u) # %,
15 (f3 < 5 && u = 0) seg(s,t,u) # P15 D(1+s*t) F*0.5 RU100 F*0.5,
16 (t < nmax)   seg(s,t,u) ## N(100−(100/nmax)*t) Dl(1+s*t) F,
17 (t>= nmax)  seg(s,t,u) ## P15 N(0) Dl(1+s*t) F,
18 tbd ## [ P6 D0.7 L5 RU−90 f*0.5 RU180 F RU−104.5 F*2 RU−151 F*2 ]
```

Table 3: The sensitive growth grammar *light2d.ssy*

In the first, 2-dimensional example (Table 3), this concept is further simplified to the degree that the cone has to be completely free from other shoots to allow the bud to develop into a new shoot. This is performed with the help of the sensitive function $f1$, declared in line 3 and applied in the condition in lines 11 and 12, which returns the maximal opening angle of a vertically-oriented cone (or sector) free of elementary units. The argument of $f1$ specifies, like in the previous example, a length threshold for shoots to be considered not relevant. The opening angle is defined to be 40° in line 2.

Another sensitive function, $f3$, comes into action in lines 13, 14 and 15. It measures the mass of leaves supported by a branch. The call of this function will be replaced by a more explicit construction in the next section, hence we will not discuss it further here. The ageing and loss of leaves (lines 1, 16 and 17), together with the appearance of a *cut operator* (%) in line 14 when the leaf mass on a branch falls under a threshold of 5, causes a self-pruning of overshadowed branches. Furthermore, if a whole tree has lost its leaves, it will fall down, an event which is caused by a transformation of its basic stem segment in line 15.

The growth of 3 trees with different ages is encoded by this grammar (line 9). Branch growth is orthotropic here, which is enabled by the negative value of V (line 8). As there is some randomness in the lengths of growth units and in the positions of branching nodes (lines 5–7), the outcome of the simulation is not deterministic. Figure 9 shows a simulation run where the middle tree dies from overshadowing, but in other runs the middle tree will survive.

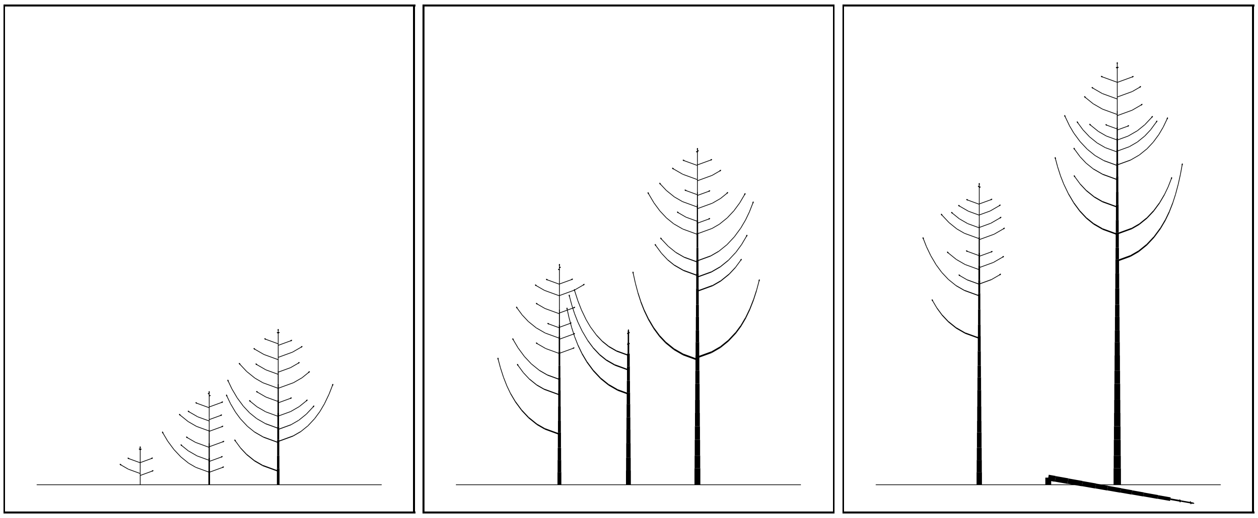


Fig. 9: Competition for light in a simple 2D tree stand, consisting of 3 uneven-aged trees, produced by GROGRA from the sensitive growth grammar *light2d.ssy* (see Table 3).
The steps 12, 25 and 31 are shown.

It should be emphasized that the trees growing in this example have no equivalent of a carbon gain and allocation. The growth, and also — indirectly — the pruning of their branches depends on the (very roughly modelled) light interception, but the amount of growth (in length and thickness) is not influenced by any assumed matter relations or by external factors. The problem of overcoming this restriction is addressed in the next two sections.

4. Calculations within a tree

To realize the desired model of carbon economy in a grammar-based framework, it is necessary to keep track of numerical values which are associated to certain elementary units or to parts of a tree — or even to a complete tree, if a global carbon storage (e.g., in the root stock) is to be included in the model. To this purpose, some new constructions are introduced into growth grammars.

4.1. The arithmetical-structural operators *sum* and *sump*

Sometimes it is necessary to sum up the value of some variable for all elementary units (shoots) which are accessible from a specified shoot when one proceeds in distal direction (Fig. 10). This operation can be considered as a summation over a branch complex. The value which is summed up can e.g. be leaf mass, like in the preceding example *light2d* (there the task of adding up the values was performed by an extra function, $f3$). An arithmetical operator, sum, is introduced to this purpose. It can appear in any arithmetical expression in a grammar. The syntax is

$$sum(ex)$$

or

$$sum(cond,\ ex).$$

ex is an arithmetical expression which will normally contain some sensitive variable referring to the created structure, e.g. of type *n_value*, *length* or *carbon*.

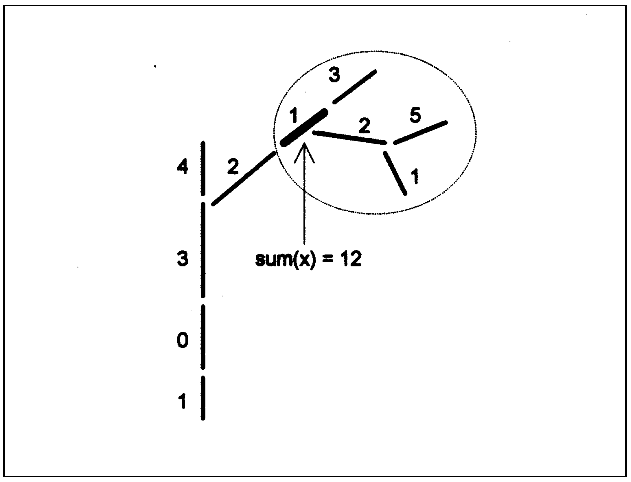


Fig. 10: The result of the operator sum, applied to a specified elementary unit (thick line). Summation is done over all units in the encircled part of the structure. The numbers associated to the units are arbitrary here and can stand e.g. for carbon content or for other quantitative attributes of shoots.

The value of ex is determined for each elementary unit in the branch system emerging from the unit under consideration (including this unit itself), and all these values are added up (see Figure 10). If there are two arguments, the first of them, $cond$, must be a condition which will be checked for each elementary unit in the branch system. In this case, the summation is done only for the units fulfilling the condition. E.g., if len is of type *length*, dia of type *diameter* and col of type *color*, $sum(col = 2,\ 3.14 * len * dia)$ adds up the surfaces of all elementary units with the colour 2 (= green). sum can also be used for counting, e.g. $sum(dia > 10,\ 1)$ calculates the number of all shoots with diameter exceeding 10 length units which belong to the considered branch.

Another summation operator, $sump$ (standing for *sum*mation along *p*ath), has the same syntax, but works in the other direction: The summation includes the elementary unit under consideration and all units down a path to the basis of the structure. I.e., the unit, its mother, the mother of its mother etc. are considered. The addition stops when a unit without mother unit is reached (Fig. 11). Again, an additional condition can restrict the realm of summation further.
The two operators sum and $sump$ can be nested and combined with each other. E.g., the expression

$$\textit{len} * sum(\textit{carb} \,/\, (sump(\textit{len}) + 0.01)\ )$$

evaluates for the given elementary unit a the formula

$$\sum_{b \in B(a)} carb(b) \cdot \frac{len(a)}{0.01 + \sum_{c \in P(b)} len(c)}$$

which specifies the amount of carbon which shoot a will obtain if the carbon masses $carb(b)$ stored in distal shoots are distributed uniformly along the paths down from these "sources" to the tree basis. (Here, $B(a)$ stands for the branch system supported by a (cf. Fig. 10), $P(b)$ for the path from b to the basis (cf. Fig. 11), and $carb$ should be declared as a variable of type *carbon*. The addition of 0.01 is done to prevent the denominator from becoming 0.) The factor "$len(a)$" is put in front of the sum operator because it refers to the (fixed) length of the unit under consideration and not to the lengths of the units in $B(a)$.

Of course, the operators sum and $sump$ can only appear in sensitive growth grammars, as the reference to the previously created structure is necessary to evaluate them. For the same reason, they are forbidden in interpretative rules.

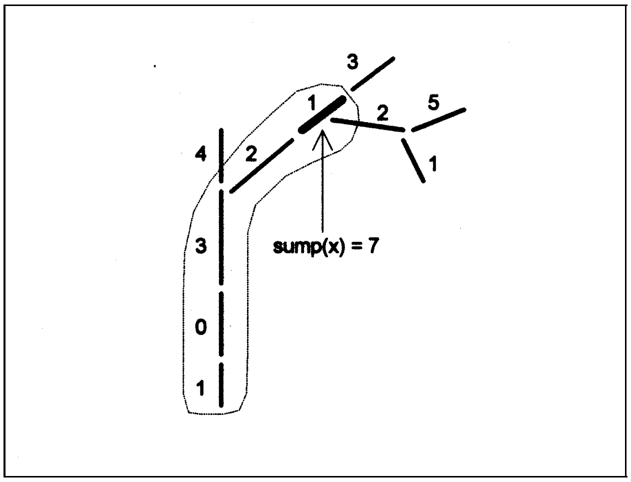


Fig. 11: The result of $sump$. Summation is done over all units along the enclosed path downwards.

4.2. Local registers

For models with a carbon pool associated to branches or to whole trees, the utilization of global variables — like the register storing the critical distance in the example *densi2d* above — is not sufficient, because each plant (or even each plant part) must have its own local variables. Therefore it is possible to tell GROGRA at an arbritrary point in the structure created from a grammar to attach some amount of free memory to the elementary unit just created. This memory is called a *local register* and can be referred to by a variable of type *local*, e.g. x when x is declared as

```
\var x local 0
```

(the number 0 specifies a default value for this variable). The creation of the local register is done by the turtle command

$$K(x).$$

When this command is given several times, different instances of the local register will be created which are distinguished by their position in the constructed structure, not by their name. If the variable x

appears in an expression, GROGRA searches along the path down from the actual elementary unit the next unit where a local register with the identifyer "x" is attached and takes the value found in that register (Fig. 12).

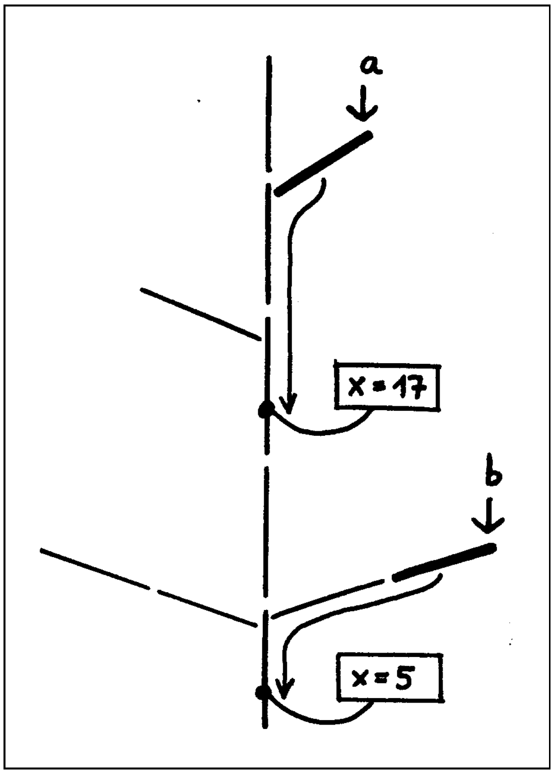


Fig. 12: The relation of local registers to shoots. In this example, the local register variable x yields 17 when evaluated for shoot a, and 5, when evaluated for shoot b.

The *assignment* of a new value to a local register variable follows the same pattern. It is performed by the pseudo-turtle command

$$A(x,\ ex),$$

where ex stands for an arbitrary arithmetical expression which is evaluated and whose value overwrites the content of the local register x. The assignment command *A* is not a proper turtle command because it is executed not in the moment when the string is interpreted by the turtle to build the new geometrical structure, but already when the previous string is read and transformed according to the generative rules. This early evaluation has the advantage that the overwritten local register values can already be referred to and used during the construction of the new string.

For local registers holds the same as for the summation operators: Their use is restricted to sensitive grammars. In the following examples, local registers will only be used for complete trees (Fig. 13).

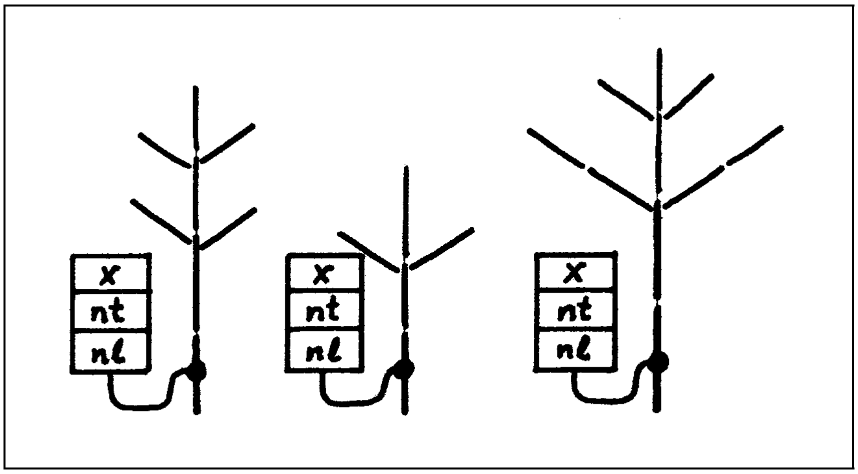


Fig. 13: Three trees, each of them having its own set of local registers, x, nt and $n\ell$ (cf. the following example, *carb2d*).

5. Plants with a carbon economy

The key question in modelling the carbon relations of trees is how the photosynthetically produced assimilates are distributed among the parts of the plant and among the different processes consuming assimilates (respiration, turnover, growth of new plant organs). This is referred to as the allocation problem. There is a vast literature about the empirical findings as well as on refined concepts for modelling allocation in trees, see e.g. Cannell & Dewar (1994). Here we will restrict ourselves to very simple assumptions about allocation and concentrate on the technical question of how to embed such assumptions in a growth grammar.

5.1. Two simple allocation concepts

We consider two extreme concepts. If the growth grammar formalism is capable of handling them both, it is reasonable to believe that it will also manage the more realistic concepts resulting from some "mixture" of the extreme strategies.
In the first approach, which we call the "centralistic concept", the assimilates produced during one year are completely collected in a "carbon pool" of the individual tree and are distributed according to some (fixed or variable) key to the different tree parts for maintenance and next year's growth (Fig. 14 a).
This concept — with some variations, especially concerning the distribution key — can be found rather often in the literature, e.g. in the LIGNUM model of structural tree growth for Scots pine (Perttunen *et al.* 1996).

The other approach will be called the "branch-autonomy concept" (Fig. 14 b). It assumes that newly produced assimilates can only be consumed in the shoot where they were produced (for maintenance and growth from the buds located there) or in the segments belonging to the path downwards to the roots. (We will not consider the allocation in the root system here.) A trivial consequence of this approach is that branches having a negative net carbon balance during one year will die immediately, i.e. they get no carbon support from the rest of the tree. Again, the distribution key defining which part of assimilate produced by a shoot is used for new growth and which part is transferred down the path can be varied.

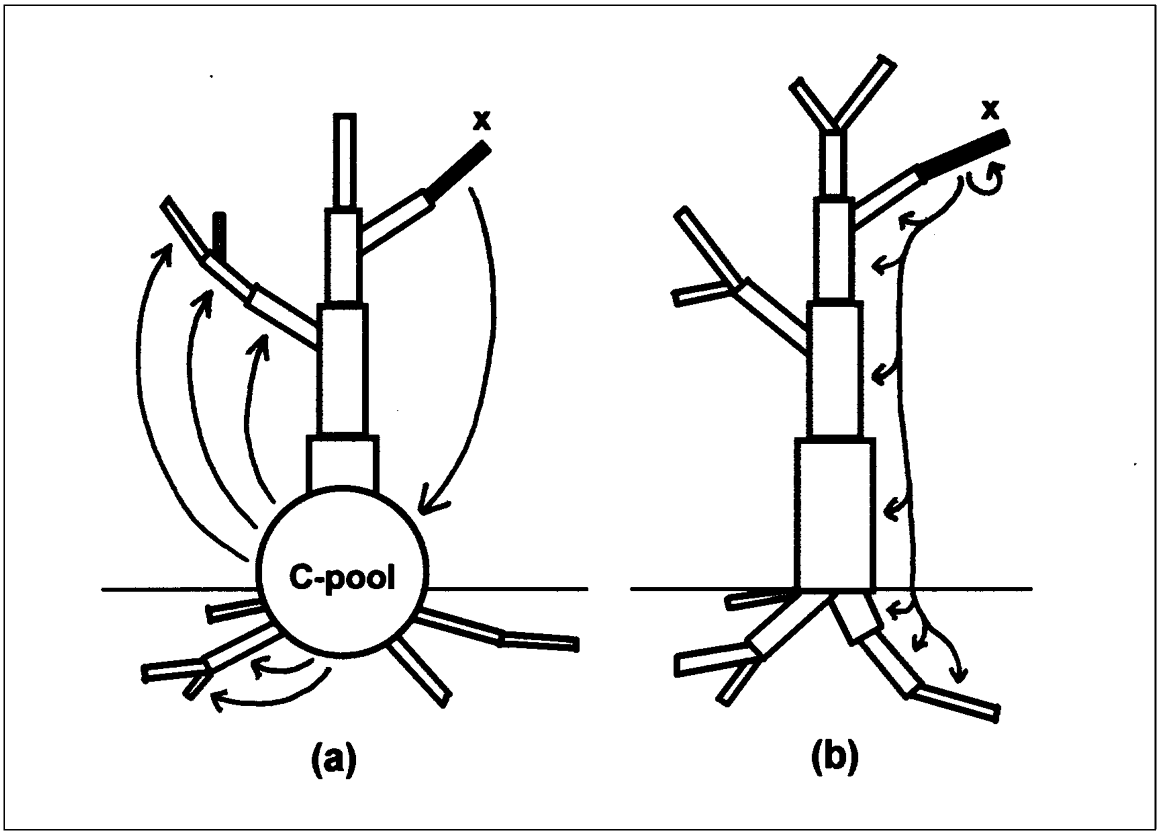


Fig. 14: The centralistic concept (a) and the branch-autonomy concept (b) for carbon allocation in trees. In (a), the assimilates produced by shoot x are distributed to potentially all shoots via the central C-pool. In (b), the assimilates from x can be consumed only by x itself and by shoots belonging to the path from x downwards.

This concept of assimilate distribution was implemented in the AMAPpara software (Reffye *et al.* 1993). With some reasonable assumptions about diameter growth, the empirical relations associated to the so-called "pipe model" (Shinozaki *et al.* 1964) — correlating sapwood diameter and leaf area — are fulfilled almost automatically in this concept (Reffye *et al.* 1993), whereas in the centralistic concept some structural assumption of pipe-model type has explicitely to be included in the distribution key to meet empirical findings.

5.2. A 2-dimensional example

The following example realizes the first, centralistic allocation concept. Assimilates are produced in all elementary units having a positive value of the non-geometric variable *lsf* (of type *n_value*, standing for leaf surface in this context). The amount of assimilate produced depends only on the value of *lsf* and — for the sake of simplicity — not on the shadow-relevant cone (however, such a dependence could easily be included). Each new unit growing from a bud consumes a certain amount of assimilate for length growth, such that the length is obtained as $c * a$, where a is the amount of assimilate invested in that new shoot and c is a constant. Assimilates are also consumed by all existing elementary units, each of them requiring an amount of $p * len * dia$ (which is meant to satisfy the necessities of maintenance respiration as well as diameter growth, which are both considered as activities of constant intensity here). Here, p is again a constant, len the length of the unit and dia its diameter, i.e. consumption is assumed to be proportional to shoot surface.

The consumption by the existing units is calculated together with the photosynthetic production. The remaining assimilates — the net balance — are collected in the central C-pool, represented by the local register x, and are redistributed to all buds for new shoot growth. The growth potential of a bud, and thereby the amount of assimilates which it receives, depends only on its morphological position: The length of lateral axis units is always shorter by a factor $\ell 1$ compared to the corresponding stem unit of the same developmental step.

The morphological rules for growth are taken from the previous, shadow-sensitive example *light2d* (see 3.3.): We have only one branching order, orthotropic growth, leaf dieback after a certain number $nmax$ of steps (which is reduced to 3 here), and the survival of a bud depends on the emptiness of the shadow-relevant cone. Branch pruning occurs when the ratio between the leaf-surface sum and the shootlength sum of a branch falls below a threshold mq. Table 4 shows the complete system.

In lines 1–8, the global constants are defined, including the above-mentioned proportionality factors c and p. (The factor relating leaf surface (*lsf*) to the production of assimilates was normed arbitrarily to 1 in this example.) Lines 9–13 declare some sensitive variables referring to the corresponding values of the actually considered elementary unit in the created structure. Line 14 enables a small random variation of branch positions. In line 15, reference is made to the sensitive function searching for obstacles in the shadow-relevant cone, like in the example *light2d*. Lines 16–18 declare three local register variables: x stands for the central assimilate pool of the plant, nt counts the number of terminal buds of the stem, and $n\ell$ the number of buds of lateral branches. (Because no forking occurs in this example, the calculation of nt is, strictly seen, redundant; the number nt is 1 for each tree and in each timestep. However, the principal possibility to handle different types of buds shall be demonstrated here.)

Line 19 initializes the 3 trees, with a distance of 600 length units (*f600*) between neighbouring ones. The local registers named x, $n\ell$ and nt are created for each tree and associated to a dummy segment at the tree basis (*F(0)*). The simulation of photosynthesis and storage of produced assimilates make sense only when a structure has already developed. Thus, lines 20–21 ensure the necessary time delay at the beginning. After that delay, the symbol $z0$ develops into a sequence of assignment commands (line 22) fulfilling in each step the task of calculating the stored carbon and counting the buds, whereas the symbol *t*(0) stands for a terminal bud (of the stem) of age 0 and develops into a stem shoot and a new terminal bud (line 24). Likewise, s(t) embodies a branch bud.

```
1  \const c 1.1,    /* coeff. C amount ⟶ length growth       */
2  \const p 0.1,    /* coeff. respiration                        */
3  \const mq 0.3,  /* threshold (min. ratio) for branch dieback   */
4  \const l1 0.8,    /* shortening factor stem ⟶ lateral branches */
5  \const nmax 3,  /* maximal needle age                          */
6  \const ang 40,   /* critical opening angle for shadow cone      */
7  \angle 70,        /* branching angle                          */
8  \set V -0.2,      /* parameter for orthotropic growth        */
9  \var len length,
10 \var lsf n_value,
11 \var dia diameter,
12 \var col color,
13 \var ord order,
14 \var su uniform 0.7 0.9,   /* possible subapical positions of branches   */
15 \var f function 1 1,        /* sensitive function (shadow cone function)  */
16 \var x local 100,
17 \var nt local 1,             /* local registers (with initialization)          */
18 \var nl local 0,
19 * # P2 D1 &(3) < [ F(0) K(x) K(nl) K(nt) z0 t(0) ] RU90 f600 RU-90 >,
                                   /*  3 trees    */
20 z0 # z1,                        /*  delayed activation   */
21 z1 # z2,
22 z2 # A(x, max(0,sum((col!=14 && col!=3), lsf-p*len*dia)))  /* amount of C */
         A(nl, sum((col=3 && ord=1),1)) A(nt, sum((col=3 && ord=0), 1)),
                                                 /*   bud counting   */
23 (f(10) > ang) t(t) # t'(t),                   /*  two steps per year: t, t'  */
24 t'(t) # di(1.8,0) P2 L(c*x/(nt+l1*nl)) seg(0,0,t)
             [ @(su) + s(0) ] [ @(su) - s(0) ] t(t+1),   /* terminal shoot */
25 (f(10) > ang) s(t) # s'(t),
26 s'(t) # di(0.8,0) P2 L(l1*c*x/(nt+l1*nl)) V seg(0,1,t) s(t+1),
                                            /* lateral shoot */
27 di(s,t) # di(s,t+0.5),                   /* secondary growth in thickness */
28 (u=0 && o=0 && sum(lsf) <= 0) seg(t,o,u) # %,
29 (u=0 && o=1 && sum(lsf)/(sum(len)+0.1) < mq) seg(t,o,u) # %,
                                            /* self-pruning */
30 seg(t,o,u) # seg(t+0.5,o,u),             /* ageing     */
31 di(s,t) ## Dl(1+s*t),
32 t(t) ## Pl14 D0.5 L4 N(0) F,         /* display of buds as short lines */
33 s(t) ## Pl14 D0.5 L3 N(0) F,
34 t'(t) ## Pl3 D0.5 L8 N(0) F,
35 s'(t) ## Pl3 D0.5 L6 N(0) F,
36 (t>=nmax) seg(t,o,u) ## N(0) P15 V*(len/100) RV F,       /* shoot display */
37 (t<nmax) seg(t,o,u) ## N(len) V*(len/100) RV F
```

Table 4: The sensitive growth grammar *carb2d.ssy*

However, to ensure that nt and $n\ell$ count the correct numbers of active buds, the calculation of the shadow-sensitive cones (together with the inactivation of overshadowed buds) had to be separated from the distribution of assimilates for new shoot growth. Hence, a rhythm of "two steps per growth phase" was introduced into the system with the help of the auxiliary symbols t' and s', standing for buds which have passed the overshadowing tests (lines 23 and 25) and are ready to grow. Inactive and active buds are displayed as small lines with different colours, which is done by the interpretative rules in lines 32–35.

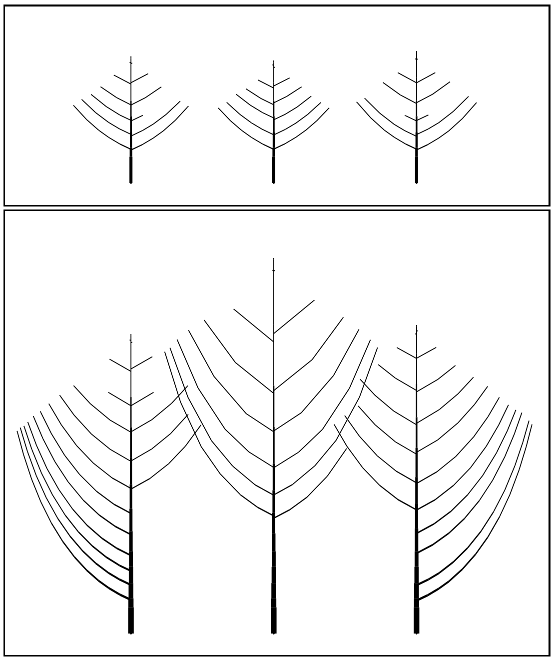


Fig. 15: Three trees with centralistic carbon allocation and light-dependent shoot growth, produced by GROGRA from the sensitive growth grammar *carb2d.ssy* (see Table 4). Steps 16 and 28 are shown.

Diameter growth is assumed to be constant (line 27) and is not connected with the carbon economy except by the general consumption term, $p * len * dia$ for each shoot (line 22), which encloses respiration and investment in secondary growth. Self-pruning is controlled for branches (line 29) and for the whole tree (line 28) by checking the sum of supported leaf surface for the basal units ($u = 0$) and by transforming these units into cut symbols (%) if this surface is too small.

The symbol $seg(t,\ o,\ u)$ stands for a segment, i.e. elementary unit, of the main axis or of a branch. The three parameters are: $t =$ age of the segment, $o =$ branching order (0 or 1), and $u =$ position index of the segment along the axis to which it belongs (0 = basis). Leaf surface, bending up and colour of the units are controlled in lines 36–37. The central rules for growth are, of course, those in lines 24 and 26, specifying how an activated bud produces a new elementary unit. In line 24, $t'(t)$ is the bud. $di(1.8,\ 0)$ specifies a fixed diameter increment (cf. line 31).

The calculation of the length of the new unit is performed in the argument of *L*: $x/(nt + \ell 1 * n\ell)$ is the amount of assimilate which the new shoot obtains according to the assumed balanced distribution between all active buds, and this number is multiplied by the proportionality factor c. Here, x, nt and $n\ell$ refer to the respective local register of the individual tree. $seg(0,\ 0,\ t)$ is the new segment, and with [@(su) + s(0)] [@(su) − s(0)] two lateral buds are positioned at (random) subapical positions, eventually generating branches with angles of $+70°$ and $-70°$ (line 7) with respect to the stem. The rule in line 26 has an analogous structure, but no branches of second order are initialized.

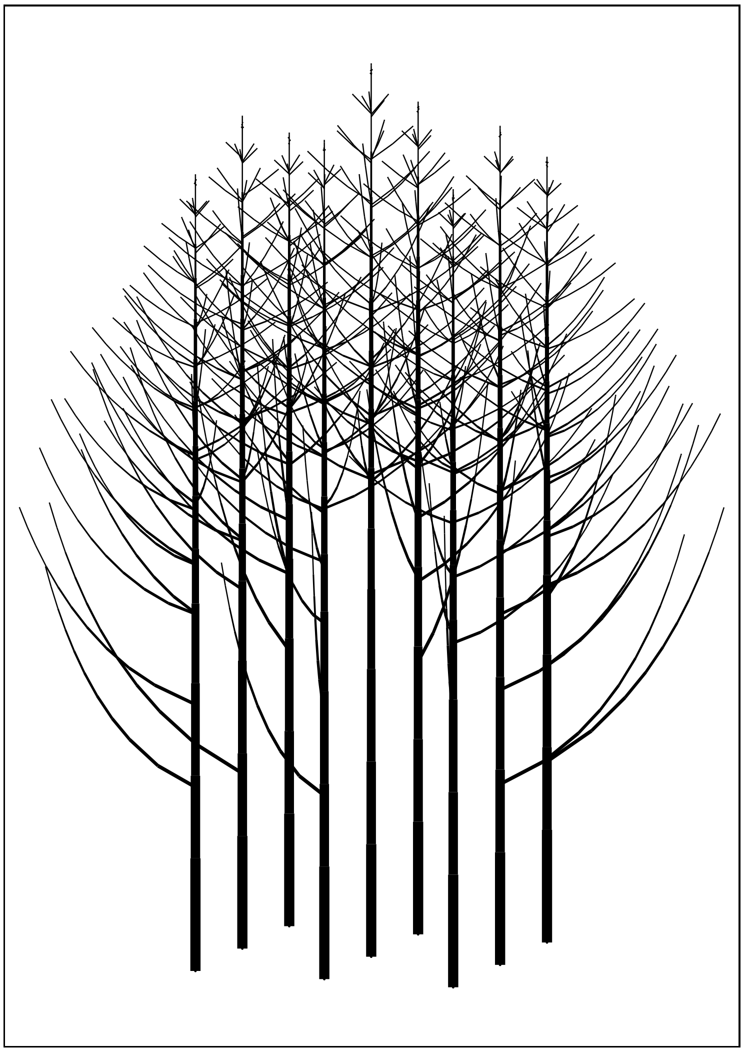


Fig. 16: Nine trees with centralistic carbon allocation and light-dependent shoot growth, produced by GROGRA from the sensitive growth grammar *carb3d.ssy* after 30 steps.

Two developmental steps of the resulting competition between the 3 trees are shown in Fig. 15. In the beginning, the trees do not come into contact with each other, and their development is nearly equal, the small differences resulting only from the random variation of branch positions (line 14). However, when crown contact begins, the middle tree is pruned more rigorously due to the overshadowing from the two neighbours, and consequently, it has less active buds than the two other trees. As a result, more assimilates are left to the upper branches and to the stem than in the two neighbour trees, and height growth is hence pushed. By the choice of the parameters c, p and mq, this effect can be enlarged or diminished. — One has to emphasize that growth in thickness is not influenced by competition in this model of allocation, making it unrealistic in this respect.

5.3. 3-dimensional examples

Some artificial effects may result from the embedding in 2-dimensional space. Therefore it is better to use 3-D models. A 3-D version of the previous example *carb2d*, named *carb3d*, was realized. Some further improvements were made in this grammar (not shown in detail here): The shadow-relevant cone function now sums up the leaf surfaces located in the cone, and the corresponding sensitive rule does still accept a value less than a certain threshold, such that the cone does not have to be completely empty. Furthermore, the branching angle becomes larger when the branch grows older, yielding a somewhat more realistic crown shape.

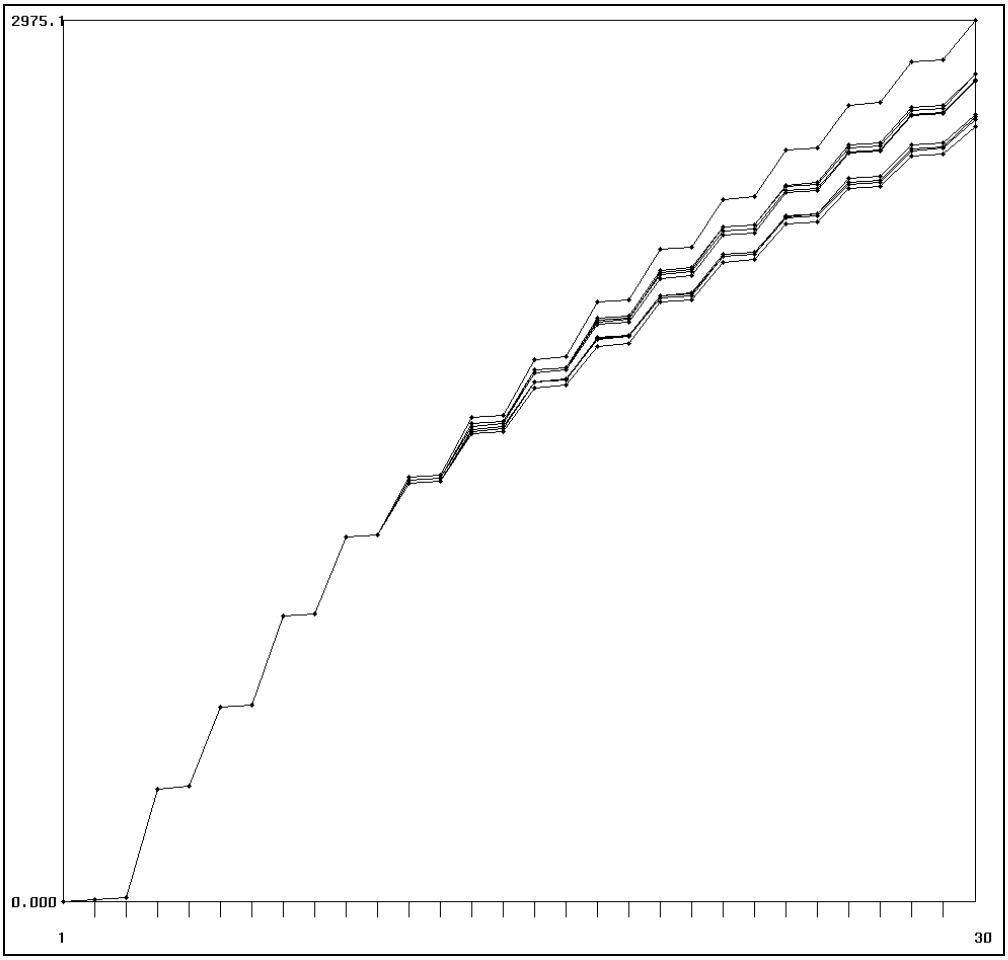


Fig. 17: Height growth curves of the 9 trees shown in Figure 16.
The lower curves correspond to the corner trees, the upper one to the central tree. Display from GROGRA.

The result of a simulation run with 9 trees, planted in an equidistant square grid pattern, is shown in Figure 16. Qualitatively, we have the same effect than in the previous example: The central tree gains an advantage in height growth from the loss of branches due to competition. Diameter growth is not affected by competition in this model.

The height growth curves of the nine trees in this simulation run are shown in Fig. 17. The upper curve corresponds to the central tree. The "staircase" shape of the curves results from the two developmental steps corresponding to one growth phase.

The parameters were chosen in a way such that growth did not run into extremes — exponential increase and immediate mortality had to be avoided. However, the parameter spaces of such models have still to be explored systematically.

The next example, *ucarb3d*, realizes the other extreme strategy for allocation, the "branch-autonomy" concept. To this purpose, the accounting of assimilates is completely done in a shoot-related variable $carb$ declared as

\var carb carbon,

and the declaration of local registers disappears from the grammar. The distribution key is a uniform one (cf. Reffye *et al.* 1993), i.e. each elementary unit a of the path $P(b)$ down from a unit b receives from b an amount of assimilates proportional to $len(a)/L$ for diameter growth, if $L = \sum_{c \in P(b)} len(c)$ is the total length of the path (cf. Fig. 18). Previous to this distribution along the path, a fixed proportion of the assimilates produced by the distal shoots is allocated into new shoots emerging from the buds.

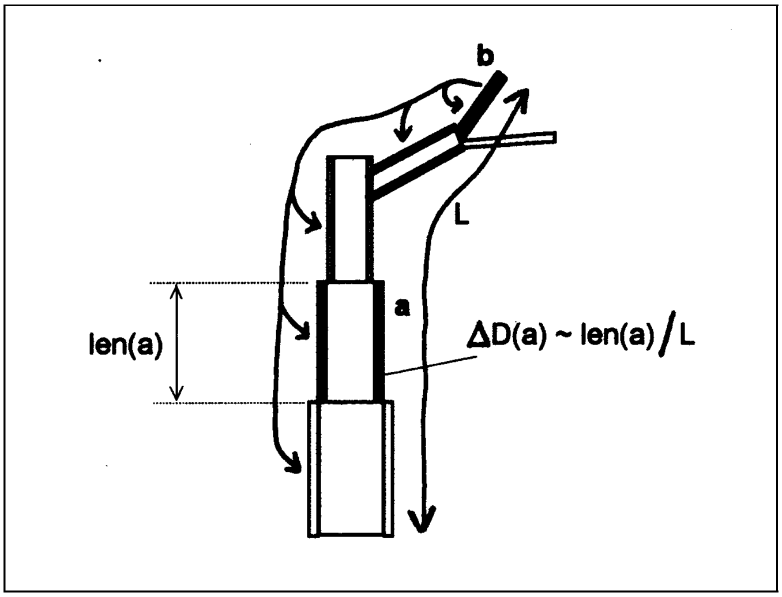


Fig. 18: Uniform mode of assimilate distribution. Unit a receives from b an amount of assimilates corresponding to its length, related to the total pathlength L.

To keep track of the diameter increments of all shoots, a new parameter d was attached to the symbol seg for an elementary unit. d is initialized with a constant value, id (= initial diameter).
The grammar rule describing diameter increment (cf. 4.1.) is now

seg(s, t, o, d) # seg'(s, t, o, d+cd∗len∗sum(carb/(sump(len)+0.01))),

and replaces, together with

seg'(s, t, o, d) # seg(s, t+1, o, d),

the corresponding rule for seg in line 30 of Table 4.
The key rules describing length growth, replacing lines 24 and 26 from Table 4, are now

```
t'(t) # P2 RH(x2) L(min(lmax, c*carb/(1+l1*nb))) seg(t,0,0,id)
        &(nb) < [ @(su) RH(i*360/nb_) an(0) s(t,0) ] > t(t+1),
```

(for the stem), and

```
s'(u,t) # P2 L(if(t=0,l1,1)*min(lmax, c*carb/(1+ln*nb)))
          V seg(t,0,1,id) s(u+1,t+1),
```

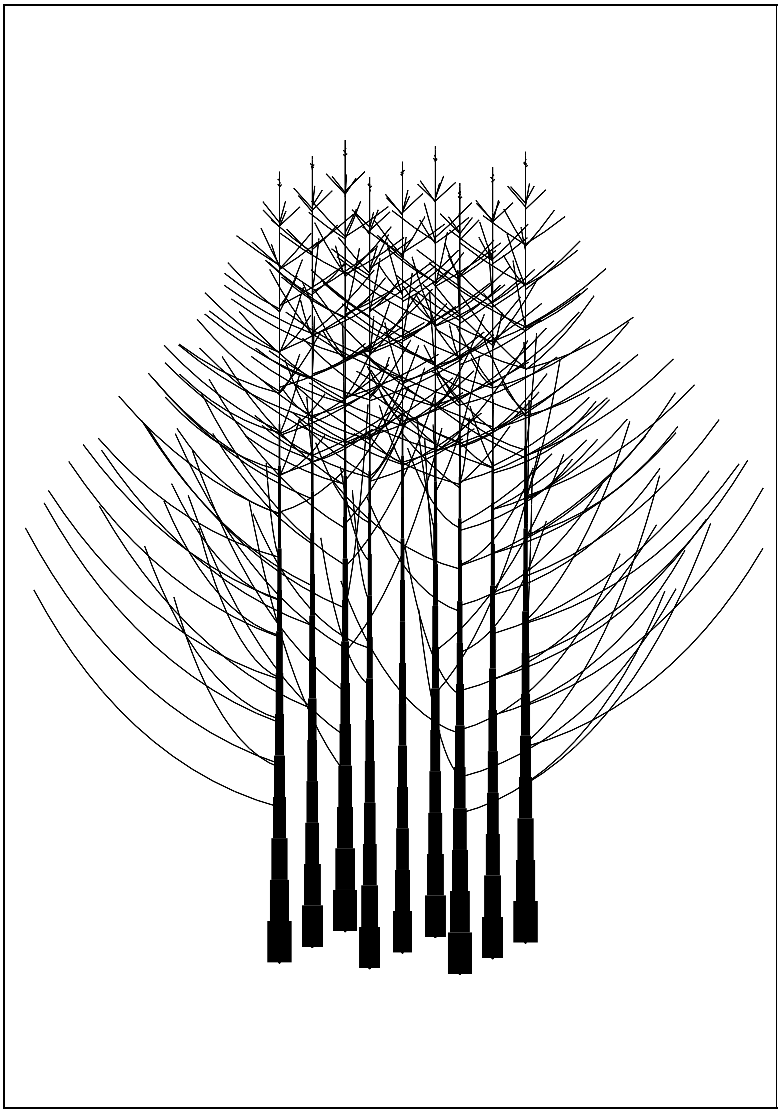


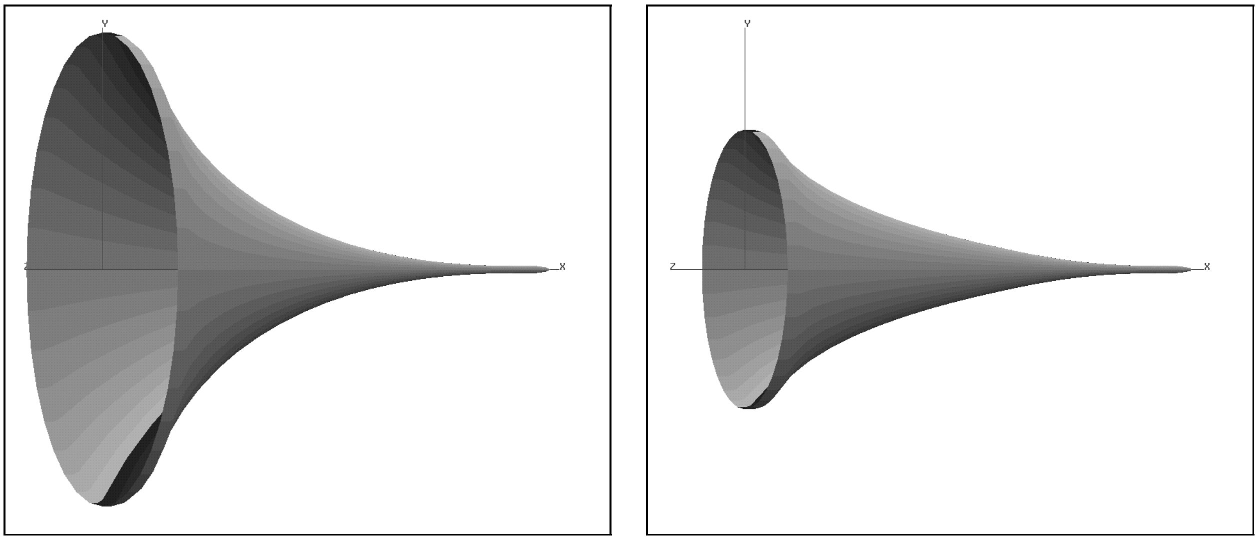
Fig. 19: Nine trees with "branch autonomy"-carbon allocation and light-dependent shoot growth, produced with GROGRA from the sensitive growth grammar *ucarb3d.ssy* after 40 steps.

where the "if"-construct ensures the smaller length of branch units compared with the stem. c, $\ell 1$ and su have the same meaning as in *carb2d* (Table 4). The "min" operation ensures that a specified maximal length, $lmax$, is not exceeded. nb is the number of lateral branches in each whorl (5 in the simulation run the result of which is shown in Fig. 19). $x2$ is a random variable between $0°$ and $360°$, and i is of type "index" (cf. *epiton.ssy* in 3.1.). To update the variable *carbon* of each shoot, the interpretative rules for seg' obtain an additional assignment command *C*, e.g. for the stem units:

(t<nmax && o=0) seg'(s,t,o,d) ## N(cn∗len) C(q∗cn∗len) Dℓ(d) F,

where cn and q are constants.

When applied to 9 trees, these rules enforce a uniform height growth of all the competing trees (Fig. 19). However, if one compares the stem shape of the central tree of these nine (Fig. 20 b) with that of a solitary tree, i.e. a tree without competitors (Fig. 20 a), it becomes visible that competition does now have influence on secondary growth (cf. also the results of AMAPpara simulations in Reffye *et al.* 1993).



(a) (b)

Fig. 20: Comparison of the stem shapes resulting from *ucarb3d.ssy*. (a) solitary tree, (b) central tree from a group of 9 trees (cf. Fig. 19). Diameters are artificially enlarged with respect to tree height (x-axis) but are of the same scale in (a) and (b). Displays produced by the software GROBOL (D. Lanwert) from GROGRA output.

6. Conclusions

With the above examples, sensitive growth grammars have revealed a certain potential for specifying tree growth models which do not only take architectural features into account, but also the physiological basis of plant growth, i.e. carbon uptake and allocation. Surely, neither the height-growth curves of Figure 17, nor the stem shapes of Figure 20 have a realistic form in comparison with field data — the models will have to be refined to a great extent. However, in a gross qualitative sense, the models — however simple their basic assumptions — are capable to reproduce certain effects of competition (enhanced height growth, slim stems) which are well-known from forestry experience.

A strategy which would repair the oversimplifications of the above models and which would lead to *quantitative* models capable of incorporating field data would be the unification of the sensitive grammar approach discussed here with that of botanically-accurate tree modelling which is also possible in the growth-grammar framework (cf. Kurth & Lanwert 1995). However, to succeed with this unification, more data are necessary about the morphological sensitivity of tree growth (i.e., about reactions on the meristem level) on environmental conditions. A relevant question would e.g. be the relative importance of shadow, mechanical stress and other constraints on the production of new buds and on the elongation of shoots. Only few investigations have addressed such problems on the appropriate scale so far (for the influence of light conditions, see Tsel'niker (1994), Stoll (1995)).
Besides this lack of a sound empirical basis, some other disadvantages and problems of the presented approach will have to be resolved in the future:

– The calculation time required by the globally-sensitive functions in GROGRA is not negligible under the present implementation and hardware conditions. The time necessary for the above 9-tree-arrays on an SGI workstation lies in the range of some minutes, but for more refined models and larger tree stands, calculation time becomes inhibitive. This is due to the "brute force" - realisation of most of the sensitive functions, which scan all elements of the structure to find e.g. the possible intersections with a shadow-relevant cone. However, there are possibilities to accelerate these search operations which are partially already realized in other plant growth models like AMAPpara (Blaise 1991), MADEIRA (List *et al.* 1994) or GPM (Breckling 1996), especially an organisation of the model space into voxels (3D-cells) which allows to restrict search to a subset of all voxels.

– In its present state, the formalism of sensitive growth grammars is a mixture of a precisely defined rule language and some auxiliary functions (like the shadow cone function) which can also be defined precisely in mathematical terms or in some standard programming language like C, but not in the rule language. The borderline between those functions and processes which are part of the growth grammar language and those which have to be specified separately is not always quite clear. There are two strategies to handle this discontinuity:

- One can define a strict border, e.g. between plant and environment, and treat the processes in the two realms by completely different formalisms. In the model implementation, the two systems will then be managed as communicating concurrent processes. This is the approach followed in Měch & Prusinkiewicz (1996).
- One could extend the rule language to enable it also to describe the processes which are excluded so far. This would lead to a unified model not only on the software level, but also in an abstract sense.

Both strategies are applied to GROGRA at the moment. Whereas the first approach is advantageous to combine quickly the morphological plant model concept with well-tried simulation models based on physics or physiology (see, e.g., Dauzat 1993, Früh 1995), the second approach is more appealing in a theoretical sense. It has also the advantage to avoid the somewhat artificial barrier between plant and environment.

– Some relevant mechanisms for the development of tree architecture are not included in the models presented here, e.g. hormone regulation, or the interaction between growth and water flow in the plant (cf. Reffye *et al.* 1995). However, a context-sensitive L-system implementation (Prusinkiewicz *et al.* 1996) of a model of Borchert & Honda (1984) simulating a dependence between growth and matter flow shows that growth grammars are generally capable of taking such mechanisms into account. Again, it is more a lack in the empirical basis which lets one hesitate to include such approaches in realistic tree models.

– A glance at grammar examples like that of *carb2d.ssy* in Table 4 leads to the impression that — though precise — the grammar notation could need further improvements in readability, transparency and shortness. Future developments have to take this aspect into account. On the other hand, some level of abstraction, like in formulas from physics or chemistry, will be unavoidable if a universal formalism for plant growth is to be developed.

References

Abelson, H., diSessa, A. A. 1982: Turtle Geometry. M.I.T. Press, Cambridge. 477 p.

Bell, A. 1976: Computerized vegetative mobility in rhizomatous plants. - In: Lindenmayer, A., Rozenberg, G. (eds.): Automata, Languages, Development. Amsterdam: 3–14.

Blaise, F. 1991: Simulation du parallélisme dans la croissance des plantes et applications. Thèse, Université Louis Pasteur, Strasbourg, Département d'Informatique. 186 p.

Borchert, R., Honda, H. 1984: Control of development in the bifurcating branch system of *Tabebuia rosea*: A computer simulation. Botanical Gazette **145**: 184–195.

Bossel, H. 1994: TREEDYN 3 Forest Simulation Model. Mathematical model, program documentation, and simulation results. Berichte des Forschungszentrums Waldökosysteme, **B 35**, Göttingen. 118 p.

Breckling, B. 1996: An individual based model for the study of pattern and process in plant ecology: An application of object oriented programming. EcoSys, **4**: 241–254.

Cannell, M. G. R., Dewar, R. C. 1994: Carbon allocation in trees: A review of concepts for modelling. Advances in Ecological Research, **25**: 59–104.

Dauzat, J. 1993: Simulated plants and radiative transfer simulations. - In: C. Varlet-Grancher, C., Bonhomme, R., Sinoquet, H. (eds.): Crop Structure and Light Microclimate. INRA, Paris: 271–278.

Früh, Th. 1995: Entwicklung eines Simulationsmodells zur Untersuchung des Wasserflusses in der verzweigten Baumarchitektur. Berichte des Forschungszentrums Waldökosysteme, **A 131**. Dissertation, Universität Göttingen. 192 p.

Gavrikov, V. L., Sekretenko, O. P. 1992: Dynamics of three-dimensional population of twigs: Crown of Scots pine. Modelling, Measurement and Control, **C 33**: 41–64.

Goel, N. S., Knox, L. B., Norman, J. M. 1991: From artificial life to real life: Computer simulation of plant growth. International Journal of General Systems, **18**: 291–319.

Govaerts, Y. M. 1996: A model of light scattering in three-dimensional plant canopies: A Monte Carlo ray tracing approach. Ph. D. Dissertation, Université Catholique de Louvain (Belgium), Joint Research Centre of the European Commission, Doc. EUR16394EN. 165 p.

Hauhs, M., Kastner-Maresch, A., Rost-Siebert, K. 1995: A model relating forest growth to ecosystem-scale budgets of energy and nutrients. Ecological Modelling, **83**: 229–243.

Jaeger, M., Reffye, Ph. de, 1992: Basic concepts of computer simulation of plant growth. J. Biosci., **17**: 275–291.

Johnson, N. L., Kotz, S. 1969: Distributions in Statistics. Discrete Distributions. Houghton Mifflin, Boston. 328 p.

Kranigk, J., Gruber, F., Heimann, J., Thorwest, A. 1994: Ein Modell für die Kronenraumstruktur und die räumliche Verteilung der Nadeloberfläche in einem Fichtenbestand. Allg. Forst- u. Jagdztg., **165**: 193–197.

Kurth, W. 1994: Growth Grammar Interpreter GROGRA 2.4. A software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modelling. Introduction and Reference Manual. Berichte des Forschungszentrums Waldökosysteme, **B 38**. Göttingen. 192 p.

Kurth, W. 1996: Elemente einer Regelsprache zur dreidimensionalen Modellierung des Triebwachstums von Laubbäumen. - In: Hempel, G. (Hrsg.): 8. Tagung der Sektion Forstliche Biometrie und Informatik im Deutschen Verband Forstlicher Forschungsanstalten (Tharandt / Grillenburg, 25.–28. 9. 1995), Ljubljana: 174–187.

Kurth, W., Lanwert, D. 1995: Biometrische Grundlagen für ein dynamisches Architekturmodell der Fichte (*Picea abies* (L.) KARST.). Allg. Forst- u. Jagdztg., **166**: 177–184.

List, R., Küppers, M., Schröder, F. 1994: Madeira — ein Programm zur Simulation des Wachstums (Biomasse und Architektur) von Holzgewächsen. Verhandlungen der Gesellschaft für Ökologie, **23**: 39–48.

Měch, R., Prusinkiewicz, P. 1996: Visual models of plants interacting with their environment. - In: Proceedings of SIGGRAPH'96 (New Orleans, Aug. 1996), ACM SIGGRAPH, New York.

Perttunen, J., Sievänen, R., Nikinmaa, E., Salminen, H., Saarenmaa, H., Väkevä, J. 1996: LIGNUM: A tree model based on simple structural units. Annals of Botany, **77**: 87–98.

Pfreundt, J., Sloboda, B. 1996: The relation of local stand structure to photosynthetic capacity in a spruce stand: A model calculation. Lesnictvi / Forestry, **42**: 149–160.

Prusinkiewicz, P., Hammel, M., Hanan, J., Měch, R. 1996: Visual models of plant development. - In: Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. III. Berlin 1997, 535–597.

Prusinkiewicz, P., James, M., Měch, R. 1994: Synthetic topiary. Computer Graphics Proceedings, Annual Conf. Ser., SIGGRAPH'94 (Orlando, Florida, 24.–29. 7. 1994), ACM SIGGRAPH, New York: 351–358.

Prusinkiewicz, P., Lindenmayer, A. 1990: The Algorithmic Beauty of Plants. Springer, New York. 228 p.

Prusinkiewicz, P., McFadzean, D. 1992: Modelling plants in environmental context. - In: Proceedings of the fourth annual western computer graphics symposium (Banff, Alberta, 6.–8. 4. 1992): 47–51.

Reffye, Ph. de, Barthélémy, D., Blaise, F., Fourcaud, Th., Houllier, F. 1995: Essai sur la simulation des relations fonctionnelles entre architecture racinaire, architecture caulinaire et croissance en épaisseur des axes chez les végétaux. - In: Colloque "L'Arbre" (Montpellier, 11.–16. 9. 1995), Naturalia Monspeliensia (to appear).

Reffye, Ph. de, Houllier, F., Blaise, F., Fourcaud, Th. 1993: Essai sur les relations entre l'architecture d'un arbre et la grosseur de ses axes végétatifs. - In: Bouchon, J., Reffye, Ph. de, Barthélémy, D. (eds.): Modélisation et Simulation de l'Architecture des Végétaux. Colloque INRA - CIRAD "Modélisation et Simulation de l'Architecture des Arbres Fruitiers et Forestiers", 23.–25. 11. 1993, Montpellier. INRA, Science Update, Paris 1997, 255–423.

Reffye, Ph. de, Houllier, F., Blaise, F., Barthélémy, D., Dauzat, J., Auclair, D. 1995b: A model simulating above- and below-ground tree architecture with agroforestry applications. Agroforestry Systems, **30**: 175–197.

Shinozaki, K., Yoda, K., Hozumi, K., Kira, T. 1964: A quantitative analysis of plant form — the pipe model theory. I. Basic analysis. Jap. J. Ecol., **14/3**: 97–105.

Stoll, P. 1995: Modular growth and foraging strategies in rhizome systems of *Solidago altissima* L. and branches of *Pinus sylvestris* L. Dissertation, Universität Zürich. 97 p.

Tsel'niker, Yu. L. 1995: The influence of light intensity on the crown-structure parameters of spruce. Lesovedenie, 1995: 73–78 (in Russian).

Unité de modélisation des plantes du CIRAD (Ed.) 1996: AMAP Software Version 2.2 Reference Guide. CIRAD, Montpellier. 188 p.

Appendix: New features of GROGRA 2.7

Here the extensions of GROGRA with respect to version 2.4 — which was described in the Reference Manual (Kurth 1994) — are listed.

1. Turtle commands

1.1. Number of internodes

The turtle commands *U*, *U*+, *U*∗, *Uℓ*, *Uℓ*+, *Uℓ*∗ were introduced for the manipulation of the current number of internodes used in the creation of the next elementary unit as an attribute. The number of internodes has no effect on the graphical display, but it is counted in the "cubic grid analysis". The semantics of the different variants of the *U* command is analogous to the semantics of the commands *L*, *D*, *N* and *V* (see Kurth 1994). The only difference is that the argument is always rounded to the nearest integer.
The "number of internodes" has to be inserted into the list of elementary unit variables (Kurth 1994, p. 81–83) as an additional variable of type "long integer". The default value of this variable is 0.

1.2. Carbon content and heartwood diameter

The turtle commands *C* (without argument) and *C*, *C*+, *C*∗, *Cℓ*, *Cℓ*+, *Cℓ*∗ (with 1 argument) can be used analogously to the *N* command to manipulate a floating-point elementary unit variable standing for "carbon" or "assimilate content" of the shoot. In the present GROGRA implementation, the elementary unit variable x_b (see Kurth 1994, p. 82) is used for this purpose. Its default value is 0.
In an analogous manner, the turtle commands *H* (without argument) and *H*, *H*+, *H*∗, *Hℓ*, *Hℓ*+, *Hℓ*∗ (with 1 argument) can be used to manipulate a floating-point elementary unit variable standing for "heartwood diameter" of the shoot under consideration. In the present GROGRA implementation, the elementary unit variable d_b (see Kurth 1994, p. 82) is used for this purpose. If no *H*-command is given, the ordinary diameter value of the turtle (specified by the *D*-command and controlling primarily the elementary unit variable d_t) is taken for d_b.

1.3. Saving and connecting points

The two following commands are useful for geometrical constructions:

$$\begin{array}{lll} S(n) & = & \text{"save"}, \\ C(m,\, n) & = & \text{"connect" (with two arguments).} \end{array}$$

S(n) (n being an arbitrary integer) saves the current state of the turtle in an internal list under the identificator n. *C*(m, n) (m and n being integers which were used before in *S*-commands) puts the turtle at the position which was its position when *S*(m) was executed, and moves it afterwards to the position corresponding to the state which was actual when *S*(n) was executed, thereby creating a new elementary unit between the two positions. Diameter, colour, N- and V-value are not taken from the saved state, i.e. they can be specified before the *C*(m, n)-command is executed. Length and direction vectors (H, L, U) are calculated from the vector connecting the two positions, the other state variables of the turtle (including the mother shoot) are taken from the state saved by *S*(m). The commands *S* and *C* are especially useful for the construction of closed lines (polygons).

1.4. Specifying the order

The command *OR*(n) enforces the branching order of the next elementary unit to be n (which must be an integer) and overrides thus the automatical determination of the branching order. This is useful for specifying an initial value when a branch with higher order shall be modelled separately.

1.5. Local register commands

K(x) creates a local register named x attached to the last created elementary unit. x must be a string. *A*(x, ex) evaluates the arithmetical expression ex and assigns the obtained value to the local register named x which lies nearest below the actual elementary unit in the last generated structure. The *A* command is executed already during generative rule application, in contrast to the other turtle commands (except *J*, see 5.4. below). The commands *K* and *A* can only be used in sensitive grammars. See Section 4.2 above for an explanation of local registers.

2. Types of variables

2.1. Random variables

Three new types of random variables are available:

\var x binomial p n specifies x to follow a binomial distribution with parameters p (= probability of success of the single Bernoulli experiment) and n (= number of repetitions).

\var x negbinomial p m specifies x to follow a negative binomial distribution with parameters p (= probability of success, $p < 1$) and m (= number of successes to obtain, $m \geq 0$), $x + m$ being the number of necessary trials (see Johnson & Kotz 1969, p. 122ff.).

\var x poisson v specifies x to follow a Poisson distribution with parameter v.

2.2. Table variables

Variables of type "table" can now be called in the r.h.s. of rules alternatively with 0 parameters (like before; the actual step number is used as argument in this case, see Kurth 1994, p. 70) or with 1 parameter. The parameter specifies which number from the table is to be taken as argument. E.g.,

```
\var k table 5 4 3,
* # F(k(1))
```

causes F to take the argument $k(1) = 4$. ($k(0)$ would be 5, $k(n)$ for $n \geq 2$ would be 3).

2.3. Elementary unit variables

Variables of the types "length", "diameter", "n_value", "color", "q_value" (see Kurth 1994, p. 71), "order" and "carbon" (see below) are now also allowed in generative rules. In this case, the growth grammar must be a sensitive one. They refer to the respective shoot attributes of the corresponding elementary unit in the structure just created (like "xcoordinate" etc.). Their use in interpretative rules remains unchanged.

2.4. Branching order

A new variable type "order" (without parameter list) was introduced. In interpretative rules, a variable of this type takes the value of the current branching order ("b" in Kurth 1994, p. 56) calculated by the turtle, in generative rules (of sensitive systems) it takes the value of the branching order ("b" in Kurth 1994, p. 82–83) of the corresponding elementary unit in the structure just created.

2.5. *Carbon content*

Analogous to "n_value", a variable of the new type "carbon" (declared without parameter list) refers to the actual carbon content of an elementary unit, currently represented by the variable x_b (see 1.2 above).

2.6. *Local register variables*

The declaration \var x local v, allowed only in sensitive grammars, specifies x to refer to a local register. v is a floating-point number which will be used in every instance of this register which is created by the $K(x)$ command (see 1.5 above) as a default value for the register content.

2.7. *Depth*

Variables of type "depth" (declared without parameter list) can be used to refer to the level of nesting in nested &- and *E*-constructions, see 5.1. and 5.2. below.

3. Sensitive functions

The following table of sensitive functions extends Table 3 from Kurth (1994), p. 72:

Number of function	*Arity*	*Return value*
4	1 or 2	$f_4(\alpha,\ 0)$ or $f_4(\alpha,\ m)$
5	1	$f_5(c)$
6	0	f_6
7	0	f_7

The return values have the following meaning:

$f_4(\alpha,\ m)$ = sum of all n_values in a cone of opening angle α emerging from the tip of the current unit upwards. Only elementary units longer than m are considered in the summation. For units intersecting the cone surface, a proportional part of the total n_value of the unit is calculated, corresponding to the ratio "length in the interior of the cone : total length of the unit".

$f_5(c)$ = amount of assimilate obtained by the current elementary unit when branch autonomy and uniform distribution (cf. Section 5.1) is assumed. The amount of exported assimilate is proportional to the n_value here, c being the proportionality factor.

f_6 = sum of all *carbon* values of the branch system which is supported by the current elementary unit (including this unit).

f_7 = angle between the H-direction of the current unit and the direction vertically upwards.

4. Arithmetical expressions

4.1. *sum and sump*

The syntax is *sum*$(cond,\ ex)$ or *sum*(ex), *sump*$(cond,\ ex)$ or *sump*(ex), with a condition (logical expression) $cond$ and an arithmetical expression ex. Both operators are allowed only in generative rules of sensitive grammars. See Section 4.1. above for the semantics.

4.2. *"if" construct*

The ternary operator *if*$(cond,\ ex1,\ ex2)$ corresponds to the expression "$cond$? $ex1$: $ex2$" in the programming language C, i.e. the value of the arithmetical expression $ex1$ is returned, if the condition $cond$ is true, and the value of $ex2$ otherwise.

5. Constructions in the rules

5.1. Nested repetition operators

In contradiction to Kurth (1994), the encapsulation (nesting) of repetition operators (&) is possible now. In the declaration of a variable of type "index", a nesting level can be specified like in the following example:

> \var i index 0,
> \var j index 1,
> $* \ \# \ \&(3) < RU20 \ \mathrm{L}(i*10+3) \ \&(2) < RU5 \ P(j+1) \ F > >$

The declaration \var i index is equivalent to \var i index 0, and up to $20 < \ldots >$ - levels can be nested.

5.2. The expand operator E

The nesting of rules is possible with the help of the operator *E* ("expand"). The rule

$$a \ \# \ E(5) \ < \ b \ c \ >,$$

specifies b and c to be further evaluated for 5 steps using the current grammar in the moment when a is replaced. That means, an arbitrary number of rewriting steps can be executed in one developmental step. *E* operators can also be nested.
Index variables can be used for the *E* operator, analogously to their use in connection with the & operator. Furthermore, a variable of type "depth" can be used to specify which rules may be used on a certain level of evaluation. E.g., in the case

> \var d depth,
> $(d=0)\ a \ \# \ E(5) < b >$,
> $b \ \# \ a \ b$,

the symbols "a" produced from b by the rule $b \ \# \ a \ b$ when the b inside the $E(5) < b >$ construction is evaluated are excluded from transformation by the first rule during the *E*-evaluation, because the depth is 1 when the second rule is applied on the b inside the $E(5) < b >$ construction and the application of the first rule is restricted to level 0 (i.e. "ordinary" developmental) steps. For applications and further examples of nested grammars see Kurth (1996).

5.3. Reversal of constructed plant parts

If the numerical argument n of a repetition operator $\&(n)$ or an expand operator $E(n)$ has a negative value, the corresponding absolute value is taken as the number of repetitions / expansion steps, and afterwards the part of the structure constructed is reverted in its direction, i.e. turned around by 180°. The turtle position afterwards is the same as if the positive argument would have been given.

5.4. The global register operator J

The semantics of the operator *J* has changed. In contrast to Kurth (1994), p. 65, a *J* operator appearing in the r.h.s. of a rule is executed in the moment when this rule is applied. (The same convention holds for the *A* operator, see 1.5. above.)

6. Descriptive data (*dtd*) syntax

Several symbols were added to the *dtd* specification language (cf. Kurth 1994, p. 114ff.), especially for enabling the encoding of branches of deciduous trees. The new symbols are:

B (with 1 integer argument)	number of leaves
C (with 1 integer argument)	colour index (referring to EGA table)
E (with 1 integer argument)	number of internodes of growth unit
F (with 1 integer argument)	number of flowers or fruits of g.u.
I (with 1 integer argument)	node index at the mother g.u. where the current g.u. originates (counted from tip of mother g.u.)
Q (with 1 integer argument)	short shoot series (several short shoots with fixed length are specified at once)
T (with 1 integer argument)	number of daughter buds (not interpreted by GROGRA)
Y (with 1 integer argument)	number of sleeping buds (not interpreted by GROGRA)
Z (with 1 integer argument)	number of dead buds (not interpreted by GROGRA)
$ (without argument)	enforces absolute orientation in space (like in case 1 on p. 118 in Kurth 1994)
. (without argument)	enforces interpretation of values given by W, R and S as describing angles in a global spherical coordinate system (instead of the usual local interpretation)
D (with 1 floating point argument)	diameter specification (no longer necessarily in the form (Dx y))

In case 2 (Kurth 1994, p. 118), L lies now in the plane which is spanned up by H and the *left* direction $L(m)$ of the mother shoot. The colour specification can be chosen interactively by the user before a *dtd* file is interpreted.

7. Interfaces

7.1. AMAP

An interface for exchange with the software AMAP-Glance (Unité... 1996) was implemented in the SGI version of GROGRA.

7.2. HYDRA

The transformation procedure for SBG structures now offers the additional possibility to specify whether the basal node of an axis is included in the reorganisation of branching nodes or not.

7.3. Interface to 3dCLIP (now called MicroEnv)

(cf. Kurth 1994, p. 129 ff.) A new *.kub* data format was developed which allows for the transfer of more information to the radiation model *MicroEnv* (formerly 3dCLIP) than before. This data format replaces the non-simplified *.kub* format described in Kurth (1994) on page 132. (The “simplified” format remains unchanged.)

The data files in the new *.kub* format for the GROGRA structures contain the following additional informations:

- Needle surface areas per cell, separately for 4 needle age classes (referring to needles grown in the years t, $t-1$, $t-2$ and older needles when t is the last year simulated (all surface values are given in mm^2)),
- Needle numbers per cell, also separately for the 4 age classes explained above,
- a direction information, given in the form of a vector obtained as the sum of all *normal* vectors of shoots in the cell under consideration (each shoot normal vector is weighted with the shootlength). Attention: this direction information differs from the direction-vector d described in (Kurth 1994, p. 132)!

The header lines of the file are the same as in the old format.
The line specifying a single cell has the following content:

$$m_x\ m_y\ m_z\ V\ vol\ a_0\ a_1\ a_2\ a_3\ n_0\ n_1\ n_2\ n_3\ dx\ dy\ dz\ \ell\ e$$

with
$m_x = x$ coordinate of cell midpoint (see old format)
$V =$ upper case letter V (marker)
$vol =$ sum of shoot (i.e., wood) volumes [mm^3] in the cell
$a_0 =$ sum of whole needle surface of current year (year t) needles in the cell,
$a_1 =$ sum of whole needle surface of year $t-1$ needles in the cell,
$a_2 =$ sum of whole needle surface of two-years old (year $t-2$) needles in the cell,
$a_3 =$ sum of whole needle surface of needles older than two years (i.e. $\leq t-3$),
$n_0 =$ total number of current year needles in the cell,
n_1 etc. analogously,
$dx = x$-component of sum of all shoot normal vectors in the cell
(shoot normal vector = orthogonal to direction vector of shoot),
$\ell =$ sum of all shoot lengths in the cell,
$e =$ exposition angle of the cell (cf. old format).

8. Analysis options

8.1. Stem analysis (option E)

There is now the possibility to choose one tree out of a simulated tree stand for the stem analysis. The user is asked for a number which identifies the tree to be selected, the number referring to the order in which the trees were initialized in the grammar.

8.2. Extensions to options F and G

Under menu option F, "numbers of daughter shoots" (Kurth 1994, p. 103 f.), the produced table was extended by two new columns.

Column 10: The sum of the squares of the diameters d_b of all daughter units of the elementary unit considered in that line,
column 11: The sum of the diameters d_b of all daughter units.

Likewise, under the menu option G, "lengths and angles" (Kurth 1994, p. 104 ff.), the table was extended by three new columns:

Column 14: The number of internodes u of the unit (cf. 1.1 above),
column 15: The number of internodes u_m of the mother unit,
column 16: The number of internodes u_b of the "big brother" of the unit, if it exists; otherwise -1.

8.3. Analysis option H

A new option H was added, producing a table in a file with extension *.dat*. The table contains a data line for each elementary unit like in the options F and G, and has the following entries:

Column 1–4: identical with column 1–4 in option G,
column 5: the relative position q of the unit,
column 6: the branching angle of the unit,
column 7: the length ℓ of the unit,
column 8: the diameter d_b of the unit,
column 9: the number of internodes u of the unit,
column 10: the number of daughter units of the unit (incl. axis-prolonging daughters),
column 11: the number of "proventive" daughter units, i.e. those having a generative distance g greater than that of the unit itself $+1$,
column 12: the length of the big brother of the unit, if it exists; otherwise: -1,
column 13: the length of the mother unit,
column 14: the colour number p of the unit,
column 15: the difference $g - g_m$ between the generative distances of the unit and its mother unit,
column 16: an index indicating the position of the unit on its axis, i.e. a lateral shoot has index 1, the prolonging apical shoot of a lateral shoot index 2, etc.

8.4. Analysis option I

A new option I was added, producing in a file with extension *.dat* a table containing a data line for each elementary unit with the following entries:

Column 1: the letter "s", followed by the number of the elementary unit,
column 2: the number n of daughter units, except the eventually existing axis-prolonging daughter unit,
columns 3 to $(n+2)$: for each daughter unit, the relative position q ($q = 1$ corresponds to the basis of the unit, $q = 0$ to the apex).
(The number of daughter units is limited to a maximum of 40 here.)

8.5. Analysis option J

The new analysis option J assumes that a single tree was generated and writes into a file with extension *.ana* some basic tree parameters for each non-empty developmental step, namely:

the number of the developmental step,
the minimal z coordinate of a unit with order 0,
the maximal z coordinate of a unit with order 0 (height of the tree),
the last height increment, i.e. the length of the stem unit with the maximal z coordinate,
the height of the crown basis, i.e. the minimal z value of an order 1 lateral unit,

the crown radius, defined as the maximal distance from the z axis,
the diameter at the stem basis,
the diameter at breast height, i.e. the diameter of the stem unit at height 1300,
the branch diameter at the crown basis,
the maximal branch diameter,
the total number of elementary units,
the sum of all n_values,
the maximal length of an order 1 axis,
the z coordinate of the node where the largest order 1 axis emerges,
the number of elementary units of this axis,
a list of all elementary unit lengths of this axis.

(All lengths are understood to be interpreted in mm.)

8.6. Fractal analysis

The "fractal analysis" option writes a file with extension *.vox* ("voxel file") which serves as input for a special software named GROFANT ("GROGRA Fractal ANalysis Tool"), designed for the determination of box-counting dimensions of GROGRA-generated structures. Similar to the "cubic grid"-analysis (options C and D), the model space is divided into cubic cells (voxels). However, in the case of the fractal analysis this is done several times with different sidelengths, which results in several datasets. A *.vox*-file has the following entries:

$\# \; n$
@1
D_1
@2
D_2
$\vdots$
@n
D_n

Here, n is the number of datasets, @j is the marker for the beginning of the j-th dataset. A dataset D_j has the following structure:

$k_t \; k_x \; k_y \; k_z \; s$
$a_x \; a_y \; a_z \; b_x \; b_y \; b_z \; c_x \; c_y \; c_z$
$L_{0,0,0}$
$L_{0,0,1}$
$\vdots$
L_{k_x-1,k_y-1,k_z-1}

where

$k_t =$ total number of voxels of the dataset $(= k_x \cdot k_y \cdot k_z)$
$k_x =$ number of voxels along the x axis (analogously: k_y, k_z),

$s =$ side length of the (cubical) voxels (floating point number),
$a_x =$ x-component of the first basis vector specifying the grid
(analogously: a_y, a_z, b_x etc.),
$L_{p,q,r} =$ data line for one voxel.

A data line $L_{p,q,r}$ has the following structure:

$p \quad q \quad r \quad v_{p,q,r}$

with $v_{p,q,r} = 0$ if the voxel is empty, and 1 otherwise.

8.7. Level analysis (option L)

In this new analysis option, which writes results to a file with extension *.dat*, shoot lengths and n_values are summed in horizontal layers. There is the alternative to work with an arbitrary number of equidistant layers or with 4 layers with arbitrary thickness. In the latter case, the heights of the 3 boundaries are asked from the user. In the output file, the data for each layer are written in one line.

Column 1: layer number,
column 2: height of the lower layer boundary,
column 3: length sum of all units in the layer,
column 4: sum of n_values of all units in the layer.

8.8. Height growth curves (option S)

This analysis option is only supported by the SGI version of GROGRA. A diagram with the height growth curves of all trees actually simulated is displayed on the screen. The x axis corresponds to time, the y axis to height. Interpolation is done linearly between the consecutive developmental steps.

9. Graphics

An interpolation modus was included in the graphical part of the SGI version of GROGRA. This made necessary several extensions:

9.1. tsp *modus*

By the command line "*tsp*" (= "treat symbols as points") in an *lsy*- or *ssy*-file, the grammar interpreter is told to create an (invisible) zero growth unit (of length and diameter zero) for each symbol which is not identical to one of the standard turtle commands (normally, such symbols are not interpreted by the turtle). These zero growth units are used as a basis for growth in the interpolation routine. (The interpolation works also without the zero units, but in this case the interpolation is done between a new shoot and its mother shoot, which is less realistic for botanical applications.)
The *tsp*-modus is also activated when before reading of the grammar file the menu item "change the treatment of symbols" in the submenu "service functions and explanations" is chosen.

9.2. Backward references

In the SGI version of GROGRA, the user is asked whether "backward references" are to be created before the interpretation of a growth grammar is started. These references are pointers connecting an elementary unit at step $t+1$ with its predecessor at step t, and they are used in the interpolation modus. There are four possible answers to this questions: "n" (or simply the return key) = "no", "y" = "yes", "e" (= "even") creates references only between the structures created at time steps with even number, "o" (= "odd") creates references only between the structures created at time steps with odd number. The appropriate choice depends on the interpreted grammar (e.g. the grammar *shoot.lsy* (see Kurth 1994, p. 145 f.) requires even interpolation).

9.3. Slider and interpolation

In the graphics mode of the SGI version of GROGRA, the key "s" activates a slider which has to be positioned by the mouse (use the left mouse button to fix it on the screen) and which enables the control of the shown timestep by clicks of the left mouse button when the mouse cursor is inside the slider window.
The key "i" activates the interpolation mode (normally, "i" should be used after "s"). To have a smooth interpolation of the growing structure, one has to move the mouse cursor from the left to the right end of the slider while pressing the left mouse button.
The keys "s" and "i" work as toggles, i.e. slider and interpolation modus can be de-activated by pressing them again.

9.4. Movie modus

A modus for automatic growth animation was installed for the graphical display of the SGI version of GROGRA. It is activated and de-activated by the "m" key. The speed of the animation can be influenced by the cursor keys (cursor up = faster, cursor down = slower). The movie modus can be used with or without the slider.

10. Other extensions

10.1. Comments

Comments can be written into grammar files in C language style, i.e. $/*$ comment $*/$.

10.2. Randomseed

By the command "*randomseed* n," where n is an integer, in the declaration part of a grammar file, the random number generator can be initialized, such that exactly identical simulation runs can be reproduced if desired. With "*askrandomseed*", the seed value for the random number generator will be asked from the user.

10.3. Start word from file

The start word for a grammar can be read from a file. To achieve this, the input at the start word request must be the character "#", immediately followed by the name of the file from where the start word is to be read. The format of the file must be that of the file "*lstri.str*" produced by GROGRA. The file *lstri.str* from a former GROGRA run can be renamed and used for this purpose, thus enabling the continuation of a former developmental sequence. When the start word is read from a file, GROGRA will first interprete this start word ("step zero"), i.e. an interpretative step will be carried out first, differing from the normal case when GROGRA starts with a generative step.

10.4. Filename extension for stored structures

The former extension *.dta* was replaced by *.dtg* (to avoid a confusion with the AMAP *dta*-format).

10.5. XWindows-version

Besides the SGI- (GL-) version and the MSDOS-version of GROGRA, there exists now an XWindows-based version for various Unix workstations (IBM, DEC-Alpha, Sun). The only difference in the control is that the exit from the graphics mode is done there by the "q" key instead of the "return" key.

11. Command line call of GROGRA

An abbreviated call of GROGRA on the system level, avoiding interactive menu handling, is possible now. If the grammar file "*name.lsy*" is to be executed, one can type the command

grogra name

instead of *grogra*. The first menu choices will be skipped then. If a sensitive grammar, "name.ssy", is to be executed, one can type

grogra name s

Several command line arguments are possible to specify certain options. Each command line argument must be a string without internal blanks. (If blanks shall be included in an argument, the whole argument is to be enclosed in "...".) The first argument must always be a filename without extension (normally, the grammar file name to be worked with). In the following, we provide a list of the possible command line arguments following the first filename (here, "name" stands for a string and "17" for some arbitrary nonnegative integer):

s	sensitive mode, an *ssy*-file is read
d	a *dtd*-file is read (the first command line argument must be the name of the *dtd*-file (without *.dtd*) in this case)
h	run GROGRA in harddisk mode
n17	specification of the number of steps (here, 17 steps) and no user requests during grammar file reading
na17	like n17, but only the last developmental step is translated into a geometrical structure ("n alone")
b	create backward references
be	create backward references for even steps
bo	create backward references for odd steps
tsp	treat symbols as points
w[name]	use the specified start word "name" (instead of default "$*$")
g[name]	generate *dtg* file after reading and grammar evaluation, filename "*name.dtg*"
ax[name]	make analysis after reading and grammar evaluation. Here, x stands for a character, corresponding to the respective menu item in the submenu "analyze the actual structure". "*name*" is used for the result-file name.