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

Winfried Kurth

Berichte des Forschungszentrums Waldökosysteme Göttingen
Ser. B, Vol. 38

Göttingen 1994

Contents

Preface

The software system which is described in this manual is still in development, and also the underlying research activities have not come to an end. Thus only a momentary state-of-the-art report can be given which will, besides describing accurately the current implementation of GROGRA, also indicate gaps and future research fields. The interested GROGRA user is requested to make his own experiences and to tell the author about any misbehaviour, uncertainties, open problems or suggestions which come up.

The main part of GROGRA was developed during the DFG project `Ku 847/1-1`, “Formal description and computer based simulation of branching habit and growth of some trees using L systems and fractals” (Postdoc program, duration 1. 9. 1991 – 31. 8. 1993). Additional support was given by the Federal Ministry of Research and Technology (BMFT) in the frame of the joint research project “Veränderungsdynamik von Waldökosystemen” (Dynamics of forest ecosystem change) of the Forest Ecosystems Research Centre at the University of Göttingen (since 1. 1. 1994), namely in subproject A7–e2, “Morphological model of tree growth”, as well as in the predecessor project “Stability conditions of forest ecosystems”, subproject PM–3.2, “Investigations concerning crown architecture” (1. 10. 1990 – 31. 12. 1993). The author gratefully acknowledges the support from DFG and BMFT.

The research activities and the program development were carried out at the Department of Forest Biometry and Informatics of the University of Göttingen, where a workgroup on plant modelling could be established. The author expresses his heartly thanks to Professor B. SLOBODA for his never-ending engagement for the benefit of this workgroup as well as for his constant encouragement. Special thanks are due to Dipl.-Biol. TH. FRÜH and Dipl.-Forstw. D. LANWERT for many fruitful discussions and for the motivating atmosphere they helped maintaining. The basic ideas and plannings for the interface to HYDRA (described in Section 5.5) originated from TH. FRÜH.

Valuable impulses came also from the AMAP research team at the “Unité de Modélisation” of the CIRAD (Montpellier), where the author had a two-months working stay. Thanks to PH. DE REFFYE, M. JAEGER, F. BLAISE and all the other members of AMAP at Montpellier for their hospitality and support.

Preface to the second edition

Since the first edition is sold out now, it was necessary to create a second edition, which will, however, not go to be printed, but will exclusively be available online in the form of a Postscript file.

No attempt was made to update the manual to the extensions of the GROGRA software which were implemented since September 1994, when the first edition was printed. The basic structure of the program remained unchanged. For the extensions until September 1996 (Version 2.7), the reader is asked to consult the appendix of the paper

> Winfried Kurth: Some new formalisms for modelling the interactions between plant architecture, competition and carbon allocation. *Bayreuther Forum Ökologie* (in press),

which will also be made accessible online and which gives a complete documentation. For the later extensions, more recent publications of the author and the file "readme", which is delivered with GROGRA, should be consulted.

The author wishes to express his thanks to all persons who have helped to improve the software by using it, asking questions and giving critical comments. Special thanks are due to G. Buck-Sorlin (formerly at the University of Wales, Bangor, now at the IPK, Gatersleben).

Winfried Kurth
Göttingen, December 19, 1997.

Introduction

GROGRA is a software, written in C language, which is designed primarily for the creation of time series of three-dimensional, plant-like branching structures. The acronym stands for **Gro**wth **Gra**mmar interpreter. GROGRA must *not* be considered as a growth model for a specific plant species, but instead as a model shell, able to create a great variety of structures, depending on its input. Its input data are mainly given in the form of *growth grammar files*, following a specific syntax and semantics, extending the possibilities of the so-called *Lindenmayer systems* (L systems; see [98]). In its nature as a model shell, GROGRA is comparable to other software tools which are universal for a specific domain, like STELLA for differential equation systems, SAS for statistical data analysis, and MACSYMA or MATHEMATICA for formal calculus and geometry (though GROGRA is not yet so advanced, of course).

GROGRA 2.4 is currently implemented in two versions: On an IBM-compatible PC (with Intel 80286 or higher processor) under MS-DOS, and on a Silicon Graphics Workstation (Iris Indigo R4000 with 24 bit XS graphics) under IRIX, a UNIX derivative. The reader of this manual is assumed to have access to one of these versions. The software GROGRA 2.4 will be distributed freely by the author for scientific purposes (not for commercial use).

GROGRA 2.4 consists of about 394 KB (DOS), resp. 766 KB (UNIX) executable code; the program source is about 15 000 lines long. Graphical output is directed immediately to the screen, or to files in HPGL, Postscript or AUTOCAD format. Several data interfaces to other software tools (amongst them HYDRA, GROBOL and 3dCLIP) are included, thus preparing the development of a carbon production and allocation supermodel together with other research groups at the Forest Ecosystems Research Centre.

However, in its current version GROGRA is mainly devoted to architectural plant models and does not take carbon, energy or nutrient economy of plants into account. In Chapter 1 of this treatise this structural approach is briefly motivated, some references to related work are given, and the position of the approach in the context of the modelling strategy of the Research Centre is indicated.

Chapter 2 will contain first an informal overview and then rigorous mathematical definitions concerning the rule language which is interpreted by GRO-

GRA. It has mainly the purpose to show that the "sensitive grammars" of GROGRA can be embedded in the framework of formal language theory (a part of theoretical computer science and, at the same time, of algebra). There will be no attempt to prove any theorems about sensitive growth grammars here. The mathematical exploration of the subject would be a separate issue, deserving its own elaboration. Section 2.2 requires some basic acquaintance in algebraic issues (see, e.g., [71] or [109]). It can be omitted by readers not interested in mathematical precision.

Chapter 3 contains the basic informations necessary to get GROGRA started, in an informal style. Together with the examples in Chapter 6, the reader unfamiliar with GROGRA can use these explanations to get first experiences before going into detail more deeply.

In Chapter 4, the rule language (stochastic, sensitive growth grammars) of GROGRA is specified — not in a mathematical framework, as in Section 2.2, but in words describing the actions which GROGRA will perform. Three levels of description have to be distinguished: 1. The so-called turtle commands, specifying a momentary spatial branching structure, 2. the L systems, giving the development of structures, and 3. meta-information, governing the choice of L system rules according to probability distributions (stochastic grammars) or to the global context in the just generated structure (globally sensitive grammars).

Chapter 5 explains all menu items, graphical options and data interfaces of the GROGRA software in an exhausting manner and gives also some informations about its internal behaviour and way of representing geometrical data. This part can also be used as a reference manual in the case of troubles with the software. The examples in Chapter 6 have the purpose to help the reader in understanding the somewhat abstract notions and explanations of the foregoing chapters.

Chapter 1

The role of growth grammars in modelling forest ecosystems

1.1 General purposes of modelling

For designing mathematical models and implementing simulation systems, two aims can be distinguished:

I. The design and application of models as an instrument of scientific research,

II. models as tools for planning and decision-making in practice.

These two purposes are not independent of each other. Before a model reaches the level of practical applicability, there is normally a phase in which it is subject of scientific research, including tests under controlled conditions. On the other hand, even purely conceptual models without concrete applications in the field can bring indirect benefits for practice if they enhance understanding.

Under aim I we collect such modelling purposes as the connection and integration of scientific results, their explanation, their further use for purposes of experimental design, hypotheses testing, dependence and sensitivity analyses, finding knowledge gaps, as well as the development of appropriate description languages (this last item being a great part of the current work on GROGRA). There are also situations in research when field measurements or experiments would be too expensive, destructive or even impossible and where appropriate simulations can overcome these difficulties to a certain extent. (This is e.g. true for measurements of total leaf area of trees, for determining their fractal dimension, or for complete matter exchange monitoring in forests.) Such stopgap models can also be subsumed under aim I. Aim II, on the other hand, embraces extrapolation, scenario making and prognosis (let us think of climate change models) as well as efforts to make scientific knowledge accessible to the man of practice (e.g. in the shape of computer-based forestry management information systems).

The GROGRA system is mainly devoted to aim I. It is primarily meant as an integrative research tool referring to botanical knowledge, forest yield, the radiation and water regime of tree crowns, competition, and (in the future) forest damage symptoms. However, the development of GROGRA is part of an integrated modelling strategy at the Forest Ecosystems Research Centre in Göttingen where aim-II-models like Geographical and Forestry Information Systems are also included (see Fig. 3 below). Experiences from the work with GROGRA will find their way into such close-to-application models. Other structural plant models similar to GROGRA were even more directly motivated by practical needs like yield calculation of coffee [99], cotton [102] or fruit trees [23].

1.2 Systematisation of approaches

To understand and / or predict tree growth and forest dynamics, a lot of different mathematical models and simulation systems have been developed, reaching from classical yield tables to physiologically based single-tree models. To bring some order into the different approaches, it is useful to arrange them in a *triangle of plant models* (Fig. 1, [70]).

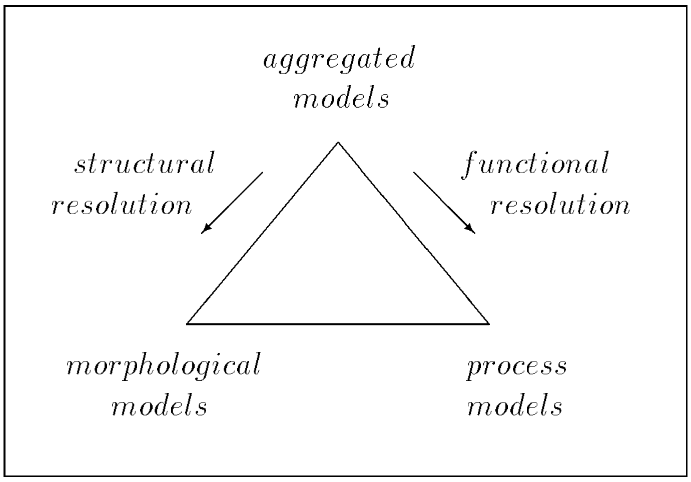


Fig. 1: A triangle of plant models

At the extreme top, one finds *aggregated models* which deal with whole plant populations (e.g. forest stands) in a statistical manner, e.g. utilizing regression analysis. When biological processes are modelled on a deeper level of causal and functional relationships, we speak of *process models* (lower right corner of the triangle). These are nowadays very common in ecology and are often expressed in the formalism of differential equations. Typical examples are the TREEDYN system of Bossel [16] as well as the single tree models for beech *(Fagus sylvatica)* of Stickan et al. [114] and of Hoffmann [56].

But there exists still another type of models: *Morphological models*, i.e. descriptions of a plant's structure and development *in space* (lower left corner of the triangle). This kind of modelling utilizes concepts like *modularity* (which is well-known in botany, see [50], [51], [119], [95]), geometrical *grammars* (which are at the heart of GROGRA) and statistics on a high-resolution level (e.g. concerning the fate of meristems in different shoot positions and different ages, see e.g. [81], [104], [9]).

Several intermediate forms of models exist ("inside the triangle"), but in most of the common process models of plant growth there is only a very small amount of three-dimensional structure present, and it is one of the aims of GROGRA to establish links between the different approaches and to show the usefulness and manageability of models with higher spatial resolution.

The "downward" dimension in the triangle of Fig. 1 can be identified with spatial or temporal resolution as well as with levels of process hierarchy in the sense of hierarchy theory ([89], [116]). Hierarchy theory considers ecosystems as determined by a complex net of interwoven processes which can be arranged in units called *holons* which are grouped in hierarchical levels distinguished by different rates and velocities of process dynamics. E.g., forest succession takes place on a higher level than the matter exchange of forest stands, which is itself on a higher level than the biochemical metabolism in the cells. In hierarchy theory, the processes on the higher levels exert a filtering and ordering influence on the lower levels, if the system is not in an instable situation which is characterized by a "hierarchy break" in the sense of a determining influence of low-level processes on the whole system's behaviour.

Hierarchy theory justifies a "bottom-up approach" in understanding and modelling plants and plant communities. A conceptual and even quantitative understanding of low-level processes is considered essential to an understanding of system behaviour, at least in system transition phases and unstable situations like those which are assumed to prevail in contemporary middle-European forest ecosystems. Past failures of this approach can be ascribed to an insufficient description of what is going on in the lower hierarchical levels, especially with respect to the *spatial structures* underlying the processes (see [6]). It is the promise of systems like GROGRA that by a better model of the complex organism-environment boundary as well as of the branching system and its development, some missing items can be added to the process models and that thereby the predictive capabilities of models as far as higher levels (forest dynamics) are concerned can be enhanced. This two-step procedure is visualized in Fig. 2: First, structures and functions have to be brought together at the basis, i.e. on the low level of plant growth processes (double arrow at the bottom), and in a second phase, by extracting condensed information and rescaling models, propositions for the higher hierarchy levels shall be made (arrow aiming at the top).

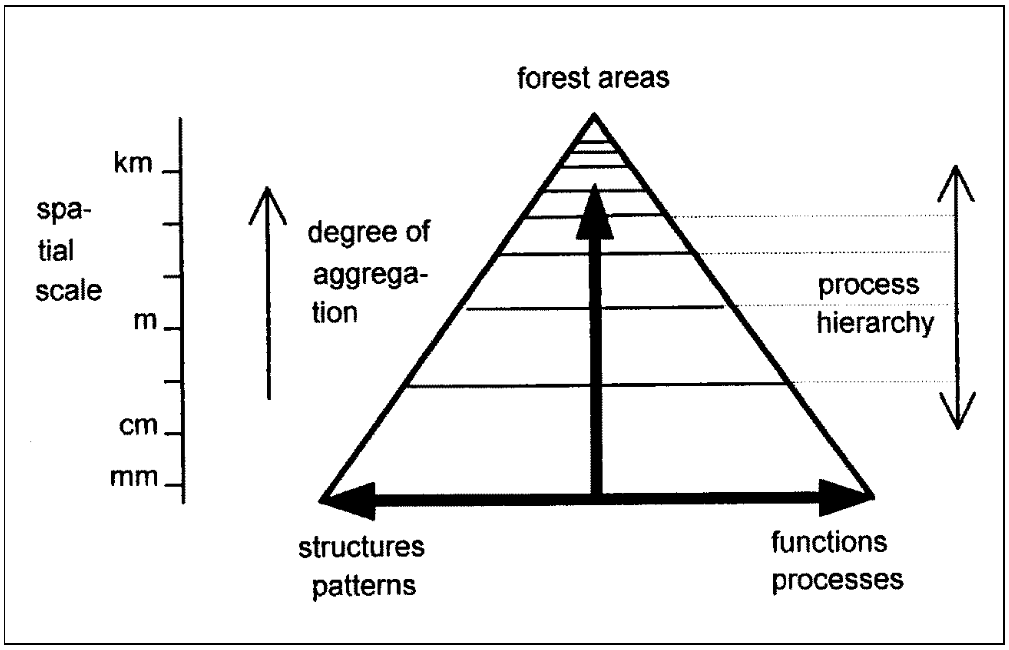


Fig. 2: Main research directions in plant modelling (bottom-up approach)

The different mathematical and simulation models currently at work in the Forest Ecosystems Research Centre Göttingen, together with their interconnections, can be inscribed into this triangle (Fig. 3). GROGRA appears clearly on the "structural" side of this picture. The meaning of the other acronyms, as well as short descriptions of the models, can be found in the Göttingen Research Centre joint application for 1994–1998 [34], Chapter "Modellbildung im Rahmen des Forschungsansatzes des FZW" (p. 302 ff).

Other researchers neglect the success chances of the bottom-up approach and favorize independent higher-level system models which are assumed to possess their own simple rules [52]. Past failures are ascribed to the deficiency of not seeing these rules and to confusing the situation by including inappropriate levels of description due to the bottom-up approach. But in the effort to deduce appropriate abstract rules (especially concerning the connection between nutrient flow and height growth strategy of trees), representatives of this research strategy have designed models which include also some 3-dimensional plant structure and development [53]. Thus the GROGRA morphological modelling approach seems not to be confined to a certain research philosophy, but to be of some general interest.

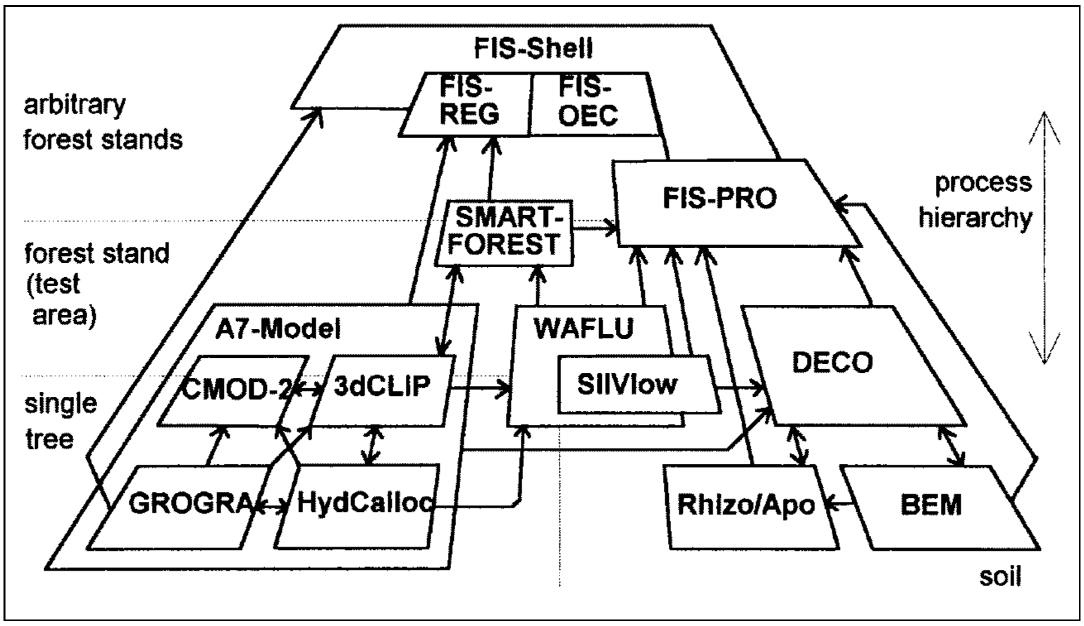


Fig. 3: The models of the Forest Ecosystems Research Centre, Göttingen

1.3 Purposes of morphological models

The attempt to construct three-dimensional models of tree structure and development is motivated by several demands:

- To quantify photosynthesis, a canopy model for the simulation of light interception is required. There is good evidence that such a model should be three-dimensionally structured to give realistic results ([111], [26]). Especially for conifers, the localization of needles of different age-classes in the canopy plays an important role. Furthermore, LANWERT [74] has found some influence of morphological trends like acrotony and branching order on needle characteristics.

- Calculations of the mechanics of trees have to take some essential features of the branching patterns into account ([85], [36]). Wind damages and the influence of stand structure on mechanical stability are important ecological items, especially for conifer stands.

- Three-dimensional structure is essential for investigations concerning the mutual mechanical obstruction of plants (a phenomenon which is important in competition and was called "phytosadism" by some botanists, see [37]).

There exist already morphological models of plant development which can simulate such effects [11].

- There are some reasons that also for a thorough understanding of the hydraulic architecture of trees the branching structure has to be taken into account (for references see [40], [41]). Research in this direction could lead to a better explanation of some damage patterns of forest decline, e.g. needle losses in certain crown sections, which can be caused by embolisms in the water conducting system.

- Understanding the developmental dynamics of mixed species stands, which are of increasing importance in contemporary forestry, requires to take spatial structure and spatial reactions into account (see e.g. [96]).

- The modelling of matter exchanges in forest ecosystems can be improved if the 3-dimensional arrangement of the interface between plant and environment is included.

- The morphological approach allows the inclusion of botanical phenomena like the sprouting of sylleptic shoots, needle losses or crown damage patterns into mathematical models. This can establish a first bridge between the classical, descriptive world of botany and ecosystemic process models in the sense of Section 1.2 above. The term "ecomorphological model" was coined for this future perspective [107].

There is also an important methodological reason: There are strong demands for integrated tree models which allow predictions concerning such different topics like stability against wind, competition strength, drought risk, timber production and CO_2 gas exchange. But if one considers the different spatial simplifications of trees which are today in use within the models concerning these topics, it becomes probable that a future common integration of all these approaches will suffer from severe inconsistencies. This problem can be avoided if a common basis is found for all the different specialized models. And this basis will inevitably be a morphological model, because morphology is common to all aspects of plant structure (Fig. 4).

From a more philosophical viewpoint, the algorithmic approach of GROGRA represents a step from *description* to *construction*, from *analysis* to *synthesis*, and can be subsumed under the new research direction of "Artificial Life" ([72], [73]). As in the parallel case of "Artificial Intelligence", the main purpose of these constructive efforts lies in the effect to get a better understanding (of life / of the human mind) by the attempt to construct artificial life (resp., intelligence), although some motivating drive which lies in the claim of being a creator can certainly not be neglected. In fact, realistic plant models have their place in modern computer graphics ([80], [103]), and in the branch of man-machine-communication which is called "virtual reality". Consequently, PRUSINKIEWICZ and LINDENMAYER attach to their book on L systems the subtitle "The virtual laboratory" [98].

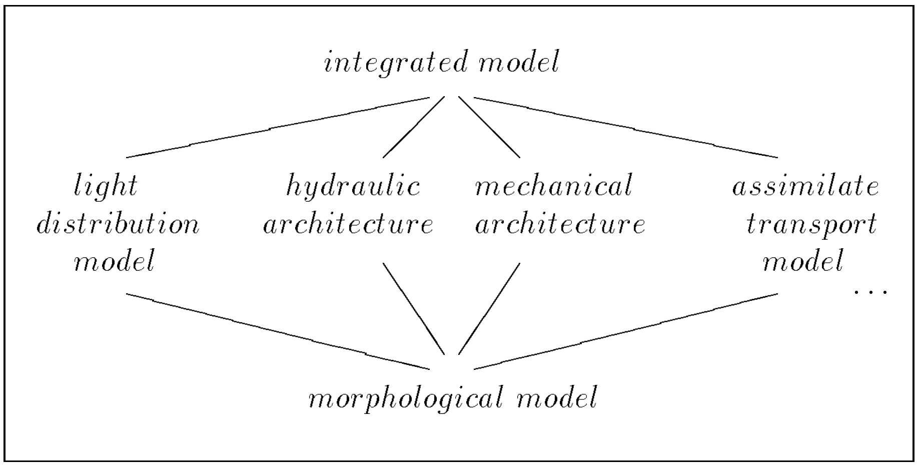


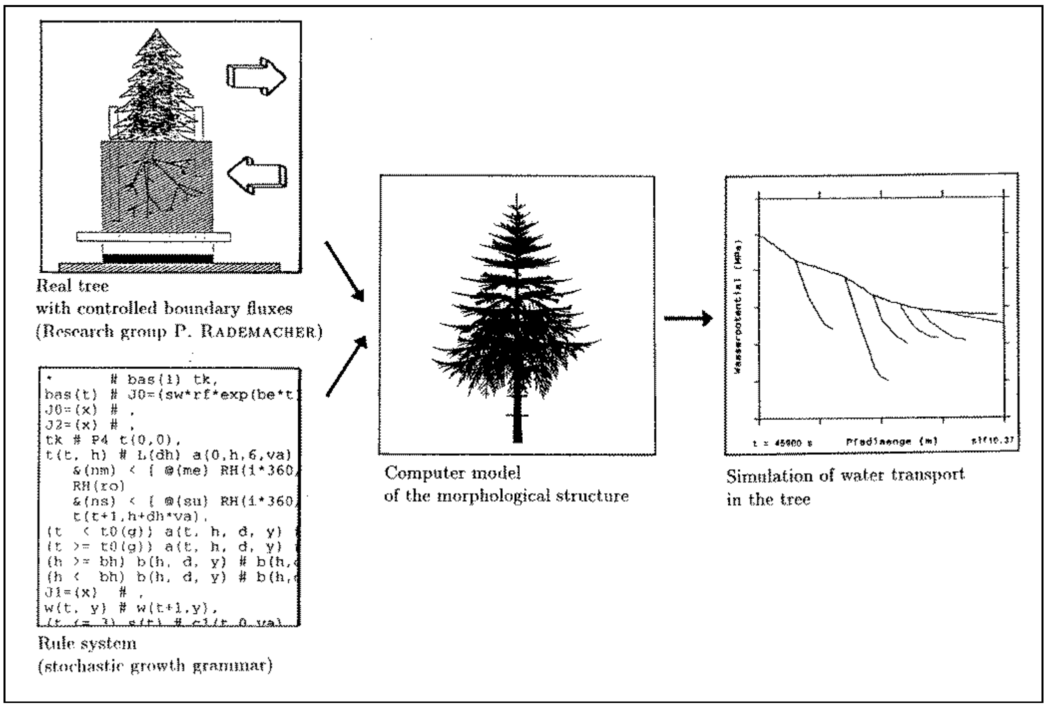
Fig. 4: Modelling different aspects of plant structure

1.4 The role of GROGRA

For the modelling activities of the Forest Ecosystems Research Centre at Göttingen, the current main topics are, according to [34]:

- the relations between *structure and function* in forest ecosystems, including the interpretation of *patterns* as signals indicating changes, and the recording and modelization of *spatial heterogeneity*,
- the *carbon relations* in forest ecosystems,
- the matter turnover in the soil,
- the elaboration of an area-based *forest ecology information system* and of models for ecological forest enterprise management.

Besides that, some activity concerning the *water relations* of trees and forest stands complement these main themes.


Real tree
with controlled boundary fluxes
(Research group P. RADEMACHER)
Computer model
of the morphological structure
Simulation of water transport
in the tree
Wasserpotential (MPa)
t = 45900 s
Pfadlaenge (m)
Rule system
(stochastic growth grammar)

```
*        # bas(1) tk,
bas(t) # J0=(sw*rf*exp(be*t
J0=(x) # ,
J2=(x) # ,
tk # P4 t(0,0),
t(t, h) # L(dh) a(0,h,6,va)
    &(nm) < [ @(me) RH(i*360
    RH(ro)
    &(ns) < [ @(su) RH(i*360
    t(t+1,h+dh*va),
(t  < t0(g)) a(t, h, d, y)
(t >= t0(g)) a(t, h, d, y)
(h >= bh) b(h, d, y) # b(h,
(h <  bh) b(h, d, y) # b(h,
J1=(x)  # ,
w(t, y) # w(t+1,y),
```

Fig. 5: Currently implemented data flow GROGRA — HYDRA

The above-mentioned topics are motivated by the relationships between the Research Centre and the Forestry Faculty at Göttingen and by their practical importance in forestry. E.g., each forestry operation changes the structural setup of the ecosystem and thereby also the processes. Here lies the main future potential of model approaches like GROGRA. Especially for the evaluation of operations on the single-tree level (e.g. limbing) there is a need for models which take into account the three-dimensional structure of stands and offer at the same time connections to process models reflecting the influences of climate, nutrient availability, water status, CO_2 etc. Such combined models will help the decision maker in planning forestry operations, selecting the appropriate tree species for a given stand and choosing optimal planting patterns (cf. [66]).

To achieve these goals, several connections between GROGRA and other models have to be built up. The way how this will be realized will become clear in more details in the section about interfaces and data formats (5.5). At present, the connection is a "one way" data interface — e.g., from GROGRA to the tree-internal water flow simulation software HYDRA (Fig. 5). Here, GROGRA can interprete structural informations, either directly from measured trees (upper left branch of the diagram) or (and this is the main purpose of GROGRA) from growth grammar rules encorporating developmental and morphological laws, and gives the resulting three-dimensional tree structure (central picture of the diagram) via a data interface to HYDRA which simulates the daily courses of the water potential and other hydraulic parameters in the branching system (an example profile seen on the right).

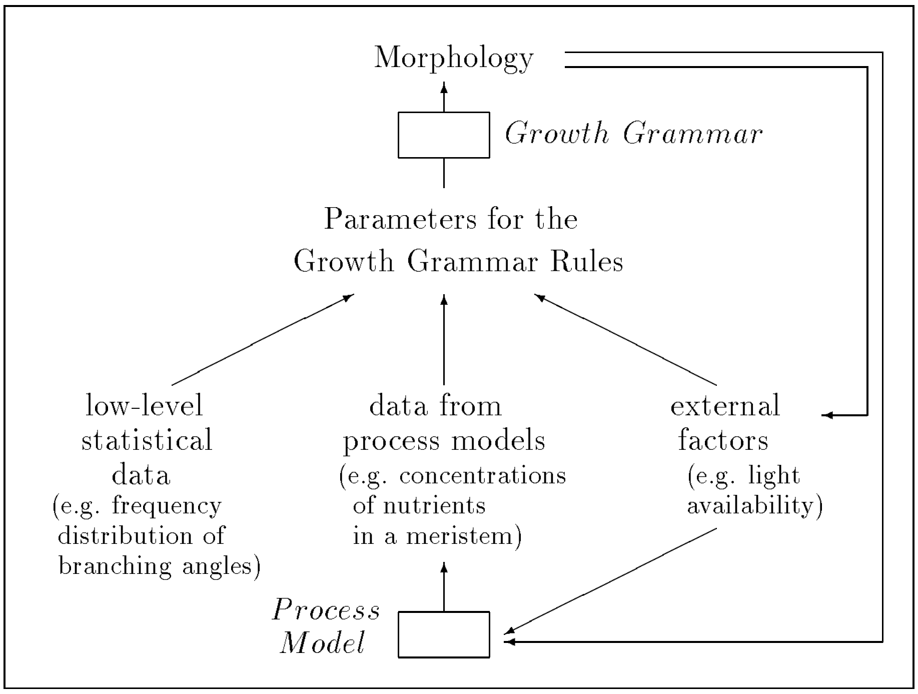

Morphology
Growth Grammar
Parameters for the Growth Grammar Rules
low-level statistical data (e.g. frequency distribution of branching angles)
data from process models (e.g. concentrations of nutrients in a meristem)
external factors (e.g. light availability)
Process Model

Fig. 6: Structure of an integrated plant growth model

For future integrated models, a feedback loop will be necessary which will enable GROGRA to take results of the process models (here: results of HYDRA concerning the water supply of meristems) into account while simulating growth. Then we arrive at a cyclic model structure as sketched in Fig. 6. Presently, such a feedback is possible in GROGRA only in a very restricted sense (see Section 4.7 on "sensitivity" and the examples 6.9 – 6.11). Future extensions of GROGRA will concentrate on this item.

1.5 Related approaches

Some of the earlier attempts to create three-dimensional tree structures exploited the notion of *self-similarity*, which has its origin in geometry. Roughly spoken, it means that inside an object a smaller copy of the whole object can be found. A weaker version of self-similarity is *self-affinity*, where the mapping relating the object with its smaller copy is allowed to be an affine mapping instead of a similarity. An example of a self-similar pattern often found in nature is the logarithmic spiral.

For *branching systems*, the presence of self-similarity implies a high regularity, a constant branching angle and constant length ratios between the branch orders — properties which are seldom to be found in a pure manner in natural trees. Simulations of trees based on strict self-similarity like some of those in [2], [14], [44] and [87] appear therefore somehow artificial and lack a profound botanical basis. — With GROGRA, it is an easy exercise to produce self-similar structures, as will be seen in the examples section.

Fractality, often falsely mixed up with self-similarity, is more an analytic than a constructive notion and comes from dimension theory. Essentially, a *fractal* is an object which is space-filling to a certain extent, describable by a dimension between 2 and 3 (or between 1 and 2, if figures in a plane are considered). Self-similar constructions often lead to fractal structures, but not in every case. In nature, fractal dimensions can be assigned to many different objects, including coastlines [82], landscapes (mountains), dust particles, vascular systems, leaves [64], and also tree crowns and root systems. For the crowns of different Rocky Mountains conifers, Zeide and Pfeifer [120] have obtained fractal dimensions between 2.13 and 2.76. However, as Rigaut has shown [106], as far as natural objects are involved, the fractal description is often, like self-similarity, an over-idealization and should better be replaced by semi-fractal models.

A tool for the fractal analysis of structures generated by GROGRA is planned as a supplement to GROGRA, but at the time it is not possible to make sound propositions about fractal dimensions of trees created by GROGRA. The term "fractal" should therefore be avoided when speaking about those structures.

Refining the self-similar approach, Barnsley [4] and his co-workers have obtained beautiful pictures of natural scenes, including forest landscapes, utilizing *iterated function systems* (IFS), that means, the iterated application of a set of carefully selected affine transformations on a start figure, resulting in a limit set with (calculable) fractal characteristics. The advantages of this approach are the high degree of condensation of the relevant information for the picture, the generality (whole stands can be modelled in the same manner as single trees) and the close connection to fractal theory. However, there are also certain earnest disadvantages, namely, the global effect of changes (to mimic the removal of a single branch, it is necessary to compute the whole picture once again), a certain vagueness in the details (demonstrating the inadequacy of purely self-similar constructions), and, most serious, the missing reference to botanical knowledge.

Some modellers of plant structures use *aggregation techniques* or *cellular automata*. The first step is the discretization of space in quadratic *pixels* or their three-dimensional analogues, *voxels* (i.e. cubic cells) [46]. The *state* of each cell tells us whether it is occupied by a phytoelement or not (it may also contain additional information) and is changed in discrete time intervals according to certain rules, the new state depending on the states of the neighbour cells. The rules can simulate certain physical or biological processes like diffusion, aggregation or growth. This technique was especially used for the simulation of root competition in a growing forest stand [53]. The cellular automata (CA) approach is quite different from that of GROGRA — the spatial discretization of the GROGRA structures is associated with the branching structure (shoots or internodes as units) and not with an external cubic grid. However, there is an interface which transforms GROGRA structures into files containing grid occupation data (see Section 5.5 for details), thus permitting a comparison with CA-generated structures. — Discretization with respect to a cubic grid ("voxelization") is also utilized as an organizational or calculation speed enhancing tool in some of the advanced extensions of the AMAP software mentioned below ([11], [27]).

Another approach in modelling tree structure which has some theoretical background — similar to fractality, but more specifically related to branching structures — is the *matrix method* of VIENNOT et al. [117], [118]. It is based on a refined form of *Horton-Strahler analysis* of branching patterns (determination of *bifurcation ratios*), a method first introduced by hydrogeologists for the analysis of river networks. The formalization of the method is carried out with notions from combinatorial mathematics, and its use has led to seemingly realistic graphical images of trees and leaves. However, the same criticism as for the methods discussed above must be applied here, namely, that the connection to botany is not very close. Especially, the ontogeny of branch development is ignored. Furthermore, there is considerable evidence in the literature that branching ratios can vary significantly between trees of the same species and even between branches of the same order in a single tree crown ([119], [57]) and therefore seem to be not very well suited for modelling purposes.

A lot of *ad-hoc methods* for generating three-dimensional plant models have been developed by different authors to simulate special species, often restricting themselves to a momentary picture of the plant, and seldom going beyond some elementary statistical methods on the theoretical side. As far as detailed field measurements precede these modelling efforts, they can partially serve as a data source for GROGRA-interpretable rule systems. As an example we mention the statistics on Sitka spruce branching and development by COCHRANE and FORD [22]. There are also several *root system simulations*, among them the model of HENDERSON et al. for Sitka spruce roots ([54], [55]), which was included in a former, preliminary version of GROGRA as a separate root model. The increased capabilities of the currently implemented type of stochastic growth grammars made it unnecessary to include such restricted models any longer. — The HENDERSON root model was extended and exploited for the purposes of the NAPAP project in the USA (National Acidic Precipitation Assessment Program), see [65]. Other root simulations include that of DIGGLE [28], PAGÈS and ARIÈS [90] and PAGÈS and KERVELLA [91], which are partially of a more general character. It is stated here (without a proof, because there was not enough time to concentrate on this subject) that these simulations could essentially be encoded by stochastic growth grammars and could then be done by GROGRA. Perhaps, in some cases some minor extensions to the growth grammar language would have to be made. — The experimental root systems generated by GROGRA until now and shown in the examples section do not pretend to mimic a certain real species.

An *ad hoc*-model is also the architectural simulation of Norway spruce trees (Picea abies (L.) Karst.) by KRANIGK [68]. This model was developed at the Forest Ecosystems Research Centre in Göttingen like GROGRA and is parametrized for a spruce stand in the Lange Bramke research area (Harz mountains). It is primarily based on botanical observations of GRUBER ([47], [48]) and was used for calculations of the light regime in the forest stand [45]. It is a static model, i.e. the simulated trees do not grow, and it is restricted to one species. On the other hand, it shows clearly the usefulness of high-resolution structural models for radiation simulations and the realistic results obtainable thereby. GROGRA will give more general and dynamic structural input for these simulations when the already implemented interface between GROGRA and the 3D climate and physiology model (3dCLIP) of the Institute for Bioclimatology in Göttingen (see Section 5.5) has passed its test phase and when growth grammars for more complex tree arrangements are designed and validated. (For another *ad hoc*-model of spruce trees see [75], for one of poplar trees see [20].)

When *dynamical models*, i.e. models not only reflecting the architecture of a plant at a given moment in time, but also simulating the development of the structure, are demanded, one must have a clear concept of how growth of plants and plant parts takes place. Botanical insight leads to the concept of *meristem-based modelling*: The shape of the plant is the result of the activity of its meristems (i.e., of its sprouting buds, responsible for primary shoot growth, and of its cambial zone, responsible for growth in diameter). Other factors like branch bending and mechanical stress [83] as well as branch shedding have also influence on the shape, but primarily the shape is seen as the trajectory of the meristems and is therefore determined by the basic processes governing the “fates of buds” [81], namely, *death*, *pause*, *growth*, *ramification* and *(de-) differentiation* (leading eventually to the phenomenon of *reiteration*) — see [8], [100], [60], [9].

The meristem-based approach was used for simulation models of plant structures by DE REFFYE ([99], [100]) and by BELL ([7], [51]). Later on, the method of DE REFFYE, which was first applied to coffee trees, was further elaborated ([60], [11], [25], [104], [61], [12] and further work) and extended to a great variety of plant species, including tropical ([24]) and temperate forest trees ([35], [19], [104]). A research group was formed at the CIRAD (Centre de Coopération Internationale en Recherche Agronomique pour le Développement) in Montpellier (France) — namely, PH. DE REFFYE, M. JAEGER, E. COSTES, F. BLAISE, Y. GUEDON, D. BARTHÉLEMY and others — which initiated cooperations with researchers at Strasbourg, Nancy, Bordeaux and Tokyo and adopted the name *AMAP*, which stands for “Atelier for Modelization of the Architecture of Plants”. Besides producing marvellous synthetic images of plants, they made valuable contributions to the theoretical basis of plant growth and architecture modelling:

- On the meristem level, stochastic processes are postulated, governing e.g. the production of internodes, which can be analyzed by renewal theory [105].
- Meristems can adapt discrete states, their branching behaviour can be simulated by Markov chains [5].
- Physiological age serves as a main organizational parameter governing meristem potential and metamorphosis. This approach is comprised in the notion of “reference axis” ([104], [61]).
- Recent extensions concern the secondary growth of trees and the inner structure of the stem [101].

Other aspects, like the mechanics of branches or the reactions of the plant to light and obstacles, are also represented in the more recent versions of the AMAP software [11].

As was essentially already recognized by FRANÇON [38], the AMAP model can be formalized within the frame of grammars. It is subject of a planned common research program of CIRAD (Montpellier) and Forest Ecosystems Research Centre (Göttingen) to demonstrate this possibility at a non-trivial example. This cooperation will at the same time establish an interface between GROGRA and AMAP. Until this common project is carried out, it can only be stated that the principal capabilities of the growth grammar approach can keep up with the basic features of AMAP (stochastic modelling of meristematic production, reference axis, sensitivity to overshadowing and obstacles...) as will be indicated by some of the examples in Chapter 6.

Lindenmayer systems (L systems) are named after the biologist Aristid Lindenmayer (1925–1989) who introduced them as a formal tool to describe the development of filamentous and branching organisms, first (in 1968) in the context of automata theory (see [77] and [98] for further early references). L systems are parallel rewriting systems operating on words (strings) which can describe geometrical structures. As replacement rule systems on discrete objects, they are subject of *formal language theory* as a branch of theoretical computer science, and a huge number of papers have since the early 70s investigated them from this point of view.

However, the "pure" classical L systems have revealed themselves as somewhat too restricted to manage the great variety of possible plant architectures and growth behaviour. Therefore, several extensions have been introduced, the stochastic sensitive growth grammars of GROGRA, which are defined in detail in Chapter 2, being one of them. Typical extensions were

- *context sensitive L systems*, where the applicability of a replacement rule depends not only on the single symbol to which it is to be applied but also on its neighbours (see [98] for more detailed definitions),
- *stochastic L systems* (named "*kakuritsuteki* L systems" by Nishida [88] in his simulation of cypress shoots), where one of several applicable rules is chosen in a random process with given probability,
- *parametric L systems* where each symbol may be complemented by a list of numerical (i.e., real-valued) parameters. The rules may contain arithmetical expressions made up of those parameters, and *conditions* restricting their applicability. These extensions were proposed rather early [76] and exploited further by Prusinkiewicz and Lindenmayer [98] for pictures of herbaceous plants.

The growth grammars of GROGRA incorporate all these extensions and go even further. (Context sensitivity is replaced by global sensitivity.) Further variants of L systems which are at the time not yet accepted by GROGRA are

- *timed L systems* [98] where a continuous time axis replaces the discrete simulation steps of the classical L systems,
- *map L systems* and cellwork L systems ([98], [79]) which generalize the rule based approach to planar, resp. space dividing structures like cell tissues.

(For the construction of surface structures like leaves, there exist also other theoretical approaches, e.g. *modular maps* [21]. See also [87] for a discussion of related topics.)

L systems were used for the description of primitive organisms like algae [86] and for the simulation of the flowering behaviour of herbaceous plants ([39], [98]).

L-system-created plant pictures were combined with IFS techniques [113] and popularized, especially by SMITH [112], under the name "*graphtals*" [115]. GOEL, KNOX and NORMAN used them as part of an integrated biological model of corn plants [42]. Implicitely, they introduced repetition operators, parametrization by functions and equivalents of the rotation operators "RG" and "RV" of GROGRA (see Sections 4.1, 4.3 and 4.5 below). However, their use of grammars differs in one essential point from that which is intended primarily with GROGRA and demonstrated in our examples: They use the dynamic character of an L system derivation, i.e. the generation of subsequent "developmental steps" only as an auxiliary tool for geometric definition; in their simulations the time steps implicit to the grammar application process do not reflect real development and aging like in our examples. The developmental characteristics of the plant are in their approach completely excluded from the rule part of the simulation system and externalized into specific "growth profile" tables — a way of handling the dynamic aspect of structure which may be appropriate in the case of smaller plants, but which is not useful in the case of trees where the development of the branching structure over a longer time is an essential part of growth.

Besides that, L systems were used for a lot of non-biological applications ([97], [43]), among them repeated tilings, weaving patterns, fractal music, computer hardware configurations, architecture of buildings, and robotics. A forerunner of our two-phase growth grammars (Section 4.2 below) was used in this context [43].

L systems are closely related to *graph grammars*, which are of growing interest in theoretical computer science and in applications including topics like pattern recognition, CAD and software engineering [13].

With few exceptions ([53], [65], [42]), the models mentioned so far are rather confined to plant architecture and do not take the *physiological basis* of growth (i.e. processes like photosynthesis, respiration, carbon gain and losses, nutrient uptake) into account. This is also to a great extent true for the current version of GROGRA which will be extended in this respect in the future. However, there are some structural models already offering such a bridge to physiology.

The model of PFREUNDT [94], which was developed at the Department of Forest Biometry and Informatics, Göttingen (like GROGRA), simulated the light interception, photosynthesis and growth of a spruce stand. It was, however, not a structural model in the strict sense: The tree crowns were modelled as hollow paraboloids with the phytoelements distributed randomly inside. No branching structure was included. But it allowed a simulation-based analysis of a practical concept, that of "shadowing biomass", for estimating relative photosynthetic capacity at given positions in the canopy — a concept that can easily be transferred to sensitive growth grammars (cf. Chapter 6, examples 6.10 and 6.11).

A model more similar to GROGRA in its structural component is that of

BERGER [10] for *Ficus elastica* and *Ficus benjamina*. Despite of its simplicity and its flaws in issues of tree morphology, it offers interesting connections between growth, light availability and optimization assumptions.

A model for young poplar trees, ECOPHYS, was developed by HOST et al. [58], taking light, temperature, carbon exchange and translocation into account. The model is based on field measurements on short-rotation poplar plantations under near-optimal water and nutrient availability and gives quite realistic results. However, the architecture of the simulated juvenile trees is very simple (no branching occurs).

A more ambitious modelling project is that of E. D. FORD, R. FORD, BASSOW and KIESTER ([30], [33], [32], [6], [31]) which was first restricted to single conifer branches and later extended to the "Simple Whole Tree" compound model in the frame of the NAPAP project [65] concentrated on forest damage caused by air pollution. FORD and BASSOW criticize the traditional regression models in forest ecosystem research (e.g. the "pipe model") for their lack in causality and propose a "model structure based on branch units" [30]. "A more precise description of the morphological construction of the growing tree and its competitive environment, and the implicit positive and negative feedbacks, is required" [6]. Morphology and phenology are seen as fundamental organizing principles, and the prediction of the correct *shape* of the tree by the model is appreciated as a valuable controlling criterion [31]. The structural model for conifer branches takes leaf display, photosynthesis, phenology, carbon export to the trunk and requirements for mechanical support of the branch (calculated according to the approach of MCMAHON and KRONAUER, [84]) into account [33]. Sensitivity and non-sensitivity assumptions for branching behaviour are investigated in computer experiments [32]. This research is in its intentions very close to the GROGRA usage. Like the FORD model in the larger context of the "Simple Whole Tree" compound model, GROGRA has to be coupled with other, specialized models to fulfill the requirements of ecosystem research.

Advantages of GROGRA in comparison with the above-mentioned models lie in its mathematical foundation in formal language theory, and especially in its generic character. However, to take account for physiological processes like the FORD model does, it will be necessary to extend GROGRA, probably into the direction of *object-oriented modelling* as it is utilized in recent plant growth models of PERTTUNEN et al. ([93], [108]) and BRECKLING [18].

Chapter 2

Stochastic sensitive growth grammars: Mathematical foundations

2.1 Overview

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

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

  (e.g.: $b \quad \longrightarrow \quad cca$), 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 $t + 1$.

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, which forms the heart of the L system modelling approach, is visualized in Fig. 7, where each arrow stands for an application of the L system rules. We refer to the numbers or timesteps 1, 2, 3... as *generations* and call the above-mentioned replacement rules also *generative rules*.

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

Fig. 7: Sequence of strings defined by an L system

However, we are not interested in strings, but in *geometrical structures* resembling three-dimensional plant architecture. Therefore we will include in our definition of a growth grammar, additionally to the above-mentioned three ingredients,

- a *geometrical interpretation* of the strings (i.e. a *semantics*) translating strings into spatial structures (subsets of $\mathbb{R}^3$).

More specifically, our geometrical interpretation will be a variant of the so-called *turtle geometry* (see [1], [98]): Some symbols will be interpreted as commands for a drawing, resp. branch-constructing device, the "turtle", which can be told by these commands e.g. to move forward, to produce a cylindrical branch, to change its direction or to change the length of the next forward moves. (The possible turtle commands in GROGRA will be explained in full detail in Section 4.1).

This interpretation is applied to each string of the generated sequence σ_1, σ_2, σ_3 ... of Fig. 7, yielding a *sequence of geometrical structures* S_1, S_2, S_3 .... Hence we arrive at the situation visualized in Fig. 8. Here, the horizontal arrows stand for the development governed by the generative rules as in Fig. 7, whereas the vertical arrows correspond to the geometrical interpretation.

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

Fig. 8: Structure generation by a simple growth grammar (L system with interpretation)

Only the structures S_1, S_2, S_3 ... will be output of the GROGRA software. The strings σ_1, σ_2, σ_3 ... used for their creation are only written temporarily into an auxiliary file and are (in the current GROGRA version) not subject of further analysis once they have been used to construct the structures.

In a *parametric L system* [98], to each symbol there can be attached a finite list of real-valued parameters (we speak of *modules* instead of symbols in this case), e.g. $a(7,\ 42,\ -0.5)$ instead of a. 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 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. must sum up to 1.) In the repeated application of rules, the choice is done randomly with the given probabilities.

The inclusion of parameters and probabilities does not principally alter the fundamental scheme of Fig. 8. The extensions imply that parameters appearing in the strings σ_1, σ_2 ... can now have influence on the geometrical interpretation (e.g., they can determine lengths, angles or thicknesses of geometrical objects being part of the structures S_1, S_2, ...), and that the generative process is no longer necessarily deterministic (random choice of rules can lead to different sequences σ_1, σ_2, σ_3 ... from the same start string α).

A further extension of the L system concept leads to the notion of *two-phase growth grammar*, standing in a certain analogy to the "two-level grammars" of VAN WIJNGAARDEN [69]: One class of rules ("metarules" in [69], corresponding to our generative rules) is used for abstract construction of objects, whereas another class of rules ("hyperrules" in [69]) encodes further, special transformations of these objects. Thus we introduce a second set of rules (possibly empty), named *interpretative rules* or *rules of the second phase* in our terminology, which do not differ in their syntax from the generative rules, but which have another function: Their application will intervene between the generation of a string σ_k and its interpretation. So the creation of the structure S_k will be 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, we will assume $\sigma'_k = \sigma_k$). Fig. 9 shows this modified structure generation process, with the vertical double arrows indicating the application of interpretative rules.

$$\begin{tikzcd}
\alpha \arrow[r] & \sigma_1 \arrow[r] \arrow[d, Rightarrow] & \sigma_2 \arrow[r] \arrow[d, Rightarrow] & \sigma_3 \arrow[r] \arrow[d, Rightarrow] & \cdots \\
 & \sigma'_1 \arrow[d] & \sigma'_2 \arrow[d] & \sigma'_3 \arrow[d] & \cdots \\
 & S_1 & S_2 & S_3 & \cdots
\end{tikzcd}$$

Fig. 9: Structure generation by a two-phase growth grammar

The introduction of two-phase grammars has 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 or 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.

Another complication arises when interactions between parts of a structure have some influence on the further development of the structure. We can have local interactions, e.g. between successive segments of a branch (Fig. 10 a), which can be taken into account by context-sensitive grammars (see [98]), or far-reaching interactions, e.g. overshadowing (Fig. 10 b), which require *global sensitivity.*

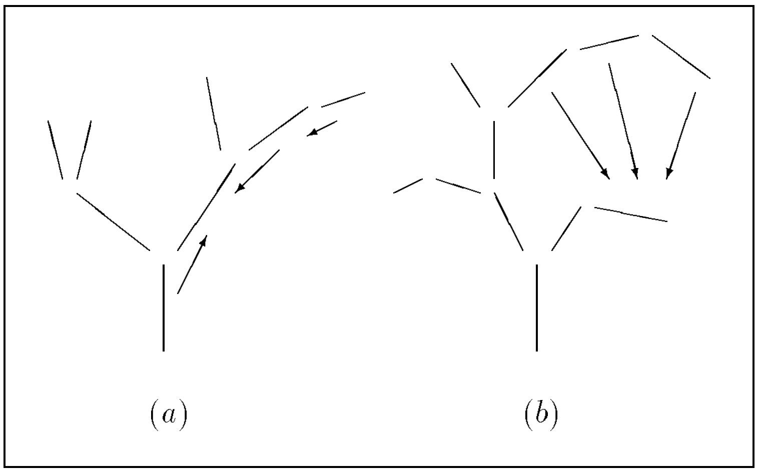


Fig. 10: Local (a) and global (b) interactions influencing the development of a structure representing a plant

In globally sensitive growth grammars, functional parameters are allowed in the generative rules which depend on the last generated geometrical structure. Hence we have an influence from S_k on the creation of σ_{k+1} (and thereby on S_{k+1}), indicated by dotted arrows in Fig. 11.

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

Fig. 11: Structure generation by a (globally)
sensitive growth grammar

Of course, sensitivity can be combined with the two-phase scheme of Fig. 9, leading to the final scheme of Fig. 12.

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

Fig. 12: Structure generation by a sensitive,
two-phase growth grammar

As the reference to the structures S_k during rule application requires additional computing time and memory, GROGRA handles sensitivity as a special extension to be required by the user explicitely, but this is not reflected in the theoretical definitions.

2.2 Mathematical definitions

If Σ is a nonempty set (an *alphabet*), let Σ^* denote the set of all *finite strings* (*words*, finite sequences) over Σ (i.e. $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$), where the *empty string*, denoted by $\Box$, is included. The *length* i of a word $w \in \Sigma^*$ is denoted by $\ell(w)$.

Let $\mathbb{N}_0$ denote the set of non-negative integers, $\mathbb{N}$ the positive integers, $\mathbb{Z}$ all integers, and $\mathbb{R}$ all real numbers. *id* stands for the identical mapping, $\mathcal{P}(A)$ for the set of all subsets of a set A.
If Σ is an alphabet, an *arity function* on Σ is a function

$$\mu : \quad \Sigma \longrightarrow \mathbb{N}_0 .$$

We call (Σ, μ) a *parametric alphabet* if μ is an arity function on Σ.

Definition 1: Let (Σ, μ) be a parametric alphabet. A *module* m over (Σ, μ) is an element $m = (a, \rho) \in \Sigma \times \mathbb{R}^*$ with $\ell(\rho) = \mu(a)$. We write m in the form

$$a(r_1, r_2, \ldots, r_k),$$

if $\rho = (r_1, r_2, \ldots, r_k)$ and $k = \ell(\rho) \geq 1$, and simply as a, if $k = 0$, and we call $r_1, \ldots, r_k$ the *actual parameters* of the symbol a.

Let $M(\Sigma, \mu)$ be the set of all modules over (Σ, μ). A *parametric string* σ over (Σ, μ) is an element of the set

$$M(\Sigma, \mu)^*,$$

i.e. a finite sequence of modules. (Cf. [98], p. 41–42.)

Definition 2: Let Γ be an alphabet, called the set of *formal variables*, let (F, ν) be a parametric alphabet, called the set of *formal functions*, and let the finite sets U, B, P, C and L be defined as follows:

$$\begin{aligned}
U &= \{-;\ \mathrm{log};\ \mathrm{exp};\ \mathrm{sqrt};\ \mathrm{atan};\ \mathrm{atg};\ \_\},\\
B &= \{+;\ -;\ *;\ /;\ \hat{}\ \},\\
P &= \{(;\ );\ ,\ \},\\
C &= \{<;\ >;\ <=;\ >=;\ =;\ ==\ ;\ !=\},\\
L &= \{\&\&\ ;\ ||\ ;\ !\ \}.
\end{aligned}$$

The set of *arithmetical expressions* over Γ (with respect to (F, ν)) is the smallest subset $\mathcal{E}$ of $(\Gamma \cup \mathbb{R} \cup U \cup B \cup F \cup P)^*$ fulfilling

(a) $\Gamma \subseteq \mathcal{E}$,
(b) $\mathbb{R} \subseteq \mathcal{E}$,
(c) $e \in \mathcal{E} \implies (e) \in \mathcal{E}$,
(d) $e \in \mathcal{E},\ u \in U \implies u(e) \in \mathcal{E}$,
(e) $e_1, e_2 \in \mathcal{E},\ b \in B \implies e_1\, b\, e_2 \in \mathcal{E}$,
(f) $e_1, \ldots, e_k \in \mathcal{E},\ f \in F,\ \nu(f) = k \implies f(e_1, \ldots, e_k) \in \mathcal{E}$.

The set of *conditional expressions* over Γ (w.r.t. (F, ν)) is the smallest subset $\mathcal{C}$ of $(\Gamma \cup \mathbb{R} \cup U \cup B \cup F \cup P \cup C \cup L)^*$ fulfilling

(g) $\square \in \mathcal{C}$,
(h) $e_1, e_2 \in \mathcal{E},\ c \in C \implies e_1\, c\, e_2 \in \mathcal{C}$,
(i) $g \in \mathcal{C} \implies (g) \in \mathcal{C}$,
(j) $g \in \mathcal{C} \implies\ !g \in \mathcal{C}$,
(k) $g, h \in \mathcal{C} \implies g\ \&\&\ h \in \mathcal{C}$ and $g\ ||\ h \in \mathcal{C}$.

Thus the syntax of arithmetical and conditional expressions resembles the usual notational conventions in the programming language C, with some extensions.

Remark. In GROGRA, the unary oparator _ (underline) is written after its argument instead of writing it in front. The unary operators $-$ and _ are also allowed without their argument being enclosed in parentheses (as required in (d) above) if this argument is an atomic expression, i.e. an element of Γ or $\mathbb{R}$. In the case of a nullary function symbol, f is written instead of $f()$. For $\mathbb{R}$ one has, of course, to substitute the set of floating point numbers representable by the available machine.

Definition 3: Let Γ be an alphabet, $\mathcal{E}(\Gamma)$ the set of arithmetical and $\mathcal{C}(\Gamma)$ the set of conditional expressions over Γ. Let h be a mapping which assigns to each $f \in F$ a function

$$h(f):\ \mathbb{R}^{\nu(f)} \longrightarrow \mathbb{R} \cup \{\text{undef}\}.$$

Let b be a function

$$b:\ \Gamma \longrightarrow \mathbb{R},$$

a so-called *instantiation* (or *substitution*). Then the *induced evaluation* $\bar{b}$ on $\mathcal{E}(\Gamma)$ (with respect to h) is the function

$$\bar{b}:\ \mathcal{E}(\Gamma) \longrightarrow \mathbb{R} \cup \{\text{undef}\}$$

defined as the homomorphism on the term algebra $\mathcal{E}(\Gamma)$ fulfilling $\bar{b}|\Gamma = b$, $\bar{b}|\mathbb{R} = id$, $\bar{b}(f(e_1, \ldots, e_k)) = h(f)(\bar{b}(e_1), \ldots, \bar{b}(e_k))$, and interpreting the unary and binary operators from U and B as their counterparts on $\mathbb{R}$, i.e.

$$\bar{b}(e_1 + e_2) = \bar{b}(e_1) + \bar{b}(e_2),$$

etc. (The binary operator ˆ corresponds to exponentiation, the unary operator *atan* to the arcustangens function, $atg(x)$ is evaluated as $arctan(\pi \cdot x/180)$, and _ corresponds to *id*. The usual priority rules are obeyed.) Expressions without meaning in $\mathbb{R}$, like log(-1), are sent to *undef* by $\bar{b}$.

The induced evaluation $\hat{b}$ on $\mathcal{C}(\Gamma)$ (with respect to h) is defined as

$$\begin{aligned} \hat{b} \ &:\ \mathcal{C}(\Gamma) \longrightarrow \{true,\ false,\ undef\}, \\ \hat{b}(\square) \ &=\ true, \end{aligned}$$

$$\begin{aligned}
\hat{b}(e_1\;c\;e_2) &= \mathit{true} \text{ if } \bar{b}(e_1)\,c\,\bar{b}(e_2) \text{ is valid in } \mathrm{I\!R}, \\
\hat{b}(e_1\;c\;e_2) &= \mathit{false} \text{ if } \bar{b}(e_1)\,c\,\bar{b}(e_2) \text{ is invalid in } \mathrm{I\!R}, \\
\hat{b}(e_1\;c\;e_2) &= \mathit{undef} \text{ iff } \bar{b}(e_1) = \mathit{undef} \text{ or } \bar{b}(e_2) = \mathit{undef},
\end{aligned}$$

and as a homomorphism on the boolean terms in $\mathcal{C}(\Gamma)$, interpreting ! as *not*, && as *and*, and || as *or* (where *undef* is everywhere treated as an absorbing element). The comparison operators = and == are synonymous, and != stands for $\neq$, <= for $\leq$, >= for $\geq$.

If (Σ, μ) is a parametric alphabet, $\lambda = (a, \tau) \in \Sigma \times \Gamma^*$ with $\ell(\tau) = \mu(a)$ (also written in the form $a(x_1, x_2, \ldots, x_{\mu(a)})$) with τ consisting of pairwise distinct variables $x_i \in \Gamma$, and $m = a(r_1, r_2, \ldots, r_{\mu(a)}) \in M(\Sigma, \mu)$ is a module with the same $a \in \Sigma$, then we speak of a *matching* between λ and m. The instantiation $b: \Gamma \longrightarrow \mathrm{I\!R}$ defined by

$$b(x) = \begin{cases} r_i & \text{if } x = x_i, \\ 0 & \text{if } x \text{ does not appear in } \lambda \end{cases}$$

is called *the instantiation defined by the matching of λ and m.*

Definition 4: An *attributed geometrical structure* (with p attributes) is a finite set $\{U_1, U_2, \ldots, U_n\}$ with

$$U_k \in \mathcal{P}(\mathrm{I\!R}^3) \times \mathrm{I\!R}^p \qquad (k = 1, 2, \ldots, n)$$

$(p, n \in \mathrm{I\!N}_0)$. Its elements U_k are called *elementary units*. The set of all attributed geometrical structures with p attributes is denoted by G_p .

The elementary units are to be interpreted in the applications of GROGRA as plant parts like shoots (growth units), internodes, leaves or flowers, which are not differentiated further in the model at interest. A typical attribute is, e.g., the needle surface of a conifer shoot.

Definition 5: Let (Σ, μ) be a parametric alphabet and $p \in \mathrm{I\!N}_0$. A *geometrical interpretation* is a mapping

$$I: \; M(\Sigma, \mu)^* \times \mathrm{I\!N} \longrightarrow \mathcal{P}(\mathrm{I\!R}^3) \times \mathrm{I\!R}^p$$

with $I(\sigma, j) = (\emptyset, 0, \ldots, 0)$ for all $j > \ell(\sigma)$. (The integer argument j is meant to mark a position in the string σ.) The *induced interpretation* of I on $M(\Sigma, \mu)^*$ is the mapping

$$\tilde{I}: \; M(\Sigma, \mu)^* \longrightarrow G_p$$

defined by $\tilde{I}(\sigma) = \{I(\sigma, j) \mid 1 \leq j \leq \ell(\sigma)\}$. The *elementary unit associated to $m_s \in M(\Sigma, \mu)$ in the string $\sigma = m_1 \ldots m_n \in M(\Sigma, \mu)^*$* $(m, s \in \mathrm{I\!N}, s \leq n)$ is given by

$$I(\sigma, s).$$

That means, the geometrical interpretation of a string is the attributed geometrical structure built up from the interpretations of all of its modules. Note that the string σ itself is part of the argument of I — hence the interpretation of a module is in general not independent of the context in which it stands.

However, we can restrict this dependence to some extent.

Definition 6: A geometrical interpretation I is called *undisturbing* if for all k, $s \in \mathbb{N}$ and all $m_1, \ldots, m_{k+s} \in M(\Sigma, \mu)$,

$$I(m_1 \ldots m_k m_{k+1} \ldots m_{k+s},\ k) = I(m_1 \ldots m_k,\ k),$$

that is, if the interpretation of a module is independent of the modules coming after it in the string.

To specify an undisturbing interpretation, it is sufficient to specify

$$i(\sigma) \ := \ I(\sigma,\ \ell(\sigma))$$

for all $\sigma \in M(\Sigma,\ \mu)^*$. The induced interpretation is then

$$\tilde{i}(\sigma) = \{ i(\pi) \mid \pi \text{ prefix of the string } \sigma \ \} \in G_p \quad . \tag{$*$}$$

The geometrical interpretation used by GROGRA is undisturbing if the usage of method calls (see Section 4.6, p. 74) is excluded.

More specifically, we can define *turtle geometry* as a special undisturbing geometrical interpretation with the help of an infinite pushdown automaton (cf. [109]), the *turtle*.

Definition 7: Define the *turtle command vocabulary* $(\Sigma,\ \mu)$, $\Sigma = \Sigma_0 \cup \Sigma_1$,

$$\Sigma_0 \ := \ \{L_0,\ D_0,\ V_0,\ F_0,\ f_0,\ RG,\ RV_0,\ +,\ -,\ \$,\ [,\ ],\ \%\}$$

and

$$\begin{aligned} \Sigma_1 \quad := \{ \quad & L,\ L+,\ L*,\ Ll,\ Ll+,\ Ll*,\ D,\ D+,\ D*,\ Dl,\ Dl+,\ Dl*, \\ & V,\ V+,\ V*,\ Vl,\ Vl+,\ Vl*,\ F,\ F+,\ F*,\ f,\ f+,\ f*,\ @, \\ & RH,\ RL,\ RU,\ RV,\ RV+,\ RV* \ \}, \end{aligned}$$

with $\mu(a) = 0$ if $a \in \Sigma_0$ and $\mu(a) = 1$ if $a \in \Sigma_1$. Let $p \in \mathbb{N}$, $p \geq 3$. Let $\mathcal{S}$ be the set of *states* of the turtle, defined as

$$\mathcal{S} \ := \ \Big(((\mathbb{R}^3)^4 \times \mathbb{R}^p)^* \setminus \{\square\} \Big) \times ((\mathbb{R}^3)^4 \times \mathbb{R}^p) \times \mathbb{Z}\,.$$

We refer to the elements of $\mathcal{S}$ in the form

$$(\sigma s,\ s_L,\ r),$$

where $s \in (\mathbb{R}^3)^4 \times \mathbb{R}^p$ is the last element of the non-empty string in the first component, called the *actual state*, $s_L \in (\mathbb{R}^3)^4 \times \mathbb{R}^p$ is the second component, called the *local state*, and $r \in \mathbb{Z}$ is named the *relevance counter*. σ is called the *actual stack content*. The actual state is written $s = (P,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$ with $P,\ H,\ L,\ U\ \in \mathbb{R}^3$, $\ell,\ d,\ v,\ \dots \in \mathbb{R}$, and the local state analogously $s_L = (P_L,\ H_L,\ L_L,\ U_L,\ \ell_L,\ d_L,\ v_L,\ \dots)$. (Note that, as $p \geq 3$, at least the three components ℓ, d, v from $\mathbb{R}$ must exist.) In the following, let $(s)_L$ denote the state obtained from s by replacing all the real-valued attributes ℓ, d, v, ... by their analogues from s_L, namely ℓ_L, d_L, v_L, ...

We refer to P as the turtle *position*, to H as its *head direction*, to L as its *left direction*, to U as its *upward direction*, to ℓ as its *steplength*, to d as its *diameter*, and to v as its *vertical tendency*. The vectors H, L, U form an orthonormal basis of $\mathbb{R}^3$. It is assumed that $P_L = P$, $H_L = H$, $L_L = L$, $U_L = U$.

The *turtle* is defined as the quintuplet

$$((\Sigma,\ \mu),\ \mathcal{S},\ s_0,\ t,\ w),$$

where $s_0 \in (\mathbb{R}^3)^4 \times \mathbb{R}^p$ is the *initial state*,

$$t: \quad M(\Sigma,\ \mu) \times \mathcal{S} \longrightarrow \mathcal{S}$$

the *transition function*, and

$$w: \quad M(\Sigma,\ \mu) \times \mathcal{S} \longrightarrow \mathcal{P}(\mathbb{R}^3) \times \mathbb{R}^p$$

the *output function* (see, e.g., [109] or [17] for background on automata theory), which are defined in detail in the following.

$$s_0 := ((0,\ 0,\ 0),\ (0,\ 0,\ 1)\,(-1,\ 0,\ 0),\ (0,\ -1,\ 0),\ \ell_g,\ d_g,\ v_g,\ 0,\ \dots),$$

where ℓ_g, d_g, $v_g \in \mathbb{R}$ are some fixed values.

For $r \leq 0$:

$$\begin{aligned} t(a,\ (\sigma s,\ s_L,\ r)) &= (\sigma s,\ s_L,\ r) \quad \text{if } a \neq\ [\ ,\ ]\ , \\ t(\ [\ ,\ (\sigma s,\ s_L,\ r)) &= (\sigma s,\ s_L,\ r-1), \\ t(\ ]\ ,\ (\sigma s,\ s_L,\ r)) &= (\sigma s,\ s_L,\ r+1). \end{aligned}$$

For $r > 0$:

$$\begin{aligned} t(\ [\ ,\ (\sigma s,\ s_L,\ r)) &= (\sigma s s,\ s_L,\ r), \\ t(\ ]\ ,\ (\sigma s,\ s_L,\ r)) &= \begin{cases} (\sigma' s_1,\ s_1,\ r) & \text{if } \sigma = \sigma' s_1, \\ (s_0,\ s_0,\ r) & \text{if } \sigma = \square\ , \end{cases} \\ t(\%,\ (\sigma s,\ s_L,\ r)) &= (\sigma s,\ s_L,\ 0). \end{aligned}$$

For $r > 0$, $a \in \Sigma_0$, $a \neq [,\ ],\ \%$:

$$t(a,\ (\sigma s,\ s_L,\ r)) = (\sigma s',\ s'_L,\ r), \quad \text{where}$$

for $a = L_0$:	$s' = (P,\ H,\ L,\ U,\ \ell_g,\ d,\ v,\ \dots)$,
	$s'_L = (P,\ H,\ L,\ U,\ \ell_g,\ d_L,\ v_L,\ \dots)$,
for $a = D_0,\ V_0$:	analogously with d_g, v_g substituted for d, resp. v,
for $a = F_0$:	$s' = (P + \ell_L \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
	$s'_L = (P + \ell_L \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v_L,\ \dots)$,
for $a = f_0$:	$s' = (P + \ell \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
	$s'_L = (s')_L$,
for $a = RG$:	let $H = (h_x,\ h_y,\ h_z)$.
	$s' = \begin{cases} (P,\ (0,\ 0,\ -1),\ (h_x^2 + h_y^2)^{-1/2} \cdot (-h_y,\ h_x,\ 0), & \\ \quad (h_x^2 + h_y^2)^{-1/2} \cdot (h_x,\ h_y,\ 0),\ \ell,\ d,\ v,\ \dots) & \text{if } h_x^2 + h_y^2 \neq 0, \\ (P,\ (0,\ 0,\ -1),\ L,\ -U,\ \ell,\ d,\ v,\ \dots) & \text{if } H = (0,\ 0,\ 1), \\ (P,\ (0,\ 0,\ -1),\ L,\ U,\ \ell,\ d,\ v,\ \dots) & \text{if } H = (0,\ 0,\ -1), \end{cases}$
	$s'_L = (s')_L$,
for $a = RV_0$:	let $J = H - v_L \cdot (0,\ 0,\ 1)$, $J^0 = J/\text{norm}(J)$.
	(Here, $\text{norm}(x_1,\ x_2,\ x_3) = (x_1^2 + x_2^2 + x_3^2)^{1/2}$.)
	$s' = \begin{cases} (P,\ J^0,\ L,\ J^0 \times L,\ \ell,\ d,\ v,\ \dots) & \text{if } J \neq 0, \\ s & \text{if } J = 0, \end{cases}$
	$s'_L = \begin{cases} (P,\ J^0,\ L,\ J^0 \times L,\ \ell_L,\ d_L,\ v,\ \dots) & \text{if } J \neq 0, \\ (P,\ H,\ L,\ U,\ \ell_L,\ d_L,\ v,\ \dots) & \text{if } J = 0. \end{cases}$
for $a = +$:	same result as for $a = RU$ with a fixed argument $w_g \in \mathbb{R}$ (see below)
for $a = -$:	same result as for $a = RU$ with the argument $-w_g$,
for $a = \$$:	let $H = (h_x,\ h_y,\ h_z)$, $J = (h_x^2 + h_y^2)^{-1/2} \cdot (-h_y,\ h_x,\ 0)$.
	$s' = \begin{cases} (P,\ H,\ J,\ H \times J,\ \ell,\ d,\ v,\ \dots) & \text{if } h_x^2 + h_y^2 \neq 0, \\ s & \text{otherwise,} \end{cases}$
	$s'_L = (s')_L$.

For $r > 0$, $a \in \Sigma_1$, $p \in \mathbb{R}$:

$$t(a(p),\ (\sigma s,\ s_L,\ r)) = (\sigma s',\ s'_L,\ r), \quad \text{where}$$

for $a = L$: $s' = (P,\ H,\ L,\ U,\ p,\ d,\ v,\ \dots)$,
$s'_L = (P,\ H,\ L,\ U,\ p,\ d_L,\ v_L,\ \dots)$,
for $a = L+$: $s' = (P,\ H,\ L,\ U,\ \ell + p,\ d,\ v,\ \dots)$,
$s'_L = (P,\ H,\ L,\ U,\ \ell + p,\ d_L,\ v_L,\ \dots)$,
for $a = L*$: $s' = (P,\ H,\ L,\ U,\ \ell \cdot p,\ d,\ v,\ \dots)$,
$s'_L = (P,\ H,\ L,\ U,\ \ell \cdot p,\ d_L,\ v_L,\ \dots)$,
for $a = Ll$: $s' = s$,
$s'_L = (P,\ H,\ L,\ U,\ p,\ d_L,\ v_L,\ \dots)$,
for $a = Ll+$: $s' = s$,
$s'_L = (P,\ H,\ L,\ U,\ \ell + p,\ d_L,\ v_L,\ \dots)$,
for $a = Ll*$: $s' = s$,
$s'_L = (P,\ H,\ L,\ U,\ \ell \cdot p,\ d_L,\ v_L,\ \dots)$,
for a beginning with D or V:
analogously, with d (resp., v) as the component to be replaced,
for $a = F$: $s' = (P + p \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (P + p \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v_L,\ \dots)$,
for $a = F+$: $s' = (P + (p + \ell_L) \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (P + (p + \ell_L) \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v_L,\ \dots)$,
for $a = F*$: $s' = (P + p \cdot \ell_L \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (P + p \cdot \ell_L \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v_L,\ \dots)$,
for $a = f$: $s' = (P + p \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (s')_L$,
for $a = f+$: $s' = (P + (p + \ell) \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (s')_L$,
for $a = f*$: $s' = (P + p \cdot \ell \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (s')_L$,
for $a = @$: $s' = (P + (p - 1) \cdot \ell \cdot H,\ H,\ L,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (s')_L$,
for $a = RH$: let $\gamma = \pi \cdot p/180$,
$s' = (P,\ H,\ L \cdot \cos\gamma + U \cdot \sin\gamma,\ U \cdot \cos\gamma - L \cdot \sin\gamma,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (s')_L$,
for $a = RL$: let $\gamma = \pi \cdot p/180$,
$s' = (P,\ H \cdot \cos\gamma + U \cdot \sin\gamma,\ L,\ U \cdot \cos\gamma - H \cdot \sin\gamma,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (s')_L$,
for $a = RU$: let $\gamma = \pi \cdot p/180$,
$s' = (P,\ H \cdot \cos\gamma - L \cdot \sin\gamma,\ H \cdot \sin\gamma + L \cdot \cos\gamma,\ U,\ \ell,\ d,\ v,\ \dots)$,
$s'_L = (s')_L$,
for $a = RV$: let $J = H - p \cdot (0,\ 0,\ 1)$, $J^0 = J/\mathrm{norm}(J)$,

$$s' = \begin{cases} (P,\ J^0,\ L,\ J^0 \times L,\ \ell,\ d,\ v,\ \dots) & \text{if } J \neq 0, \\ s & \text{if } J = 0, \end{cases}$$

$$s'_L = \begin{cases} (P,\ J^0,\ L,\ J^0 \times L,\ \ell_L,\ d_L,\ v,\ \dots) & \text{if } J \neq 0, \\ (P,\ H,\ L,\ U,\ \ell_L,\ d_L,\ v,\ \dots) & \text{if } J = 0, \end{cases}$$

for $a = RV+$ or $a = RV*$: Analogously to $a = RV$ with
$J = H - (v_L + p) \cdot (0,\ 0,\ 1)$, resp. $J = H - v_L \cdot p \cdot (0,\ 0,\ 1)$.

The output function is defined by

$$w(a,\ (\sigma s,\ s_L,\ r)) = (\emptyset,\ 0,\ 0,\ \ldots\ ) \ \text{ if } \ r \le 0 \ \text{ or } \ a \notin \{F_0,\ F,\ F+,\ F*\ \},$$

and, if $r > 0$, let

$$u = \begin{cases} \ell_L & \text{if } a = F_0, \\ p & \text{if } a = F \text{ with argument } p, \\ p + \ell_L & \text{if } a = F+ \text{ with argument } p, \\ p \cdot \ell_L & \text{if } a = F* \text{ with argument } p. \end{cases}$$

Furthermore, $Q \ := \ P + u \cdot H$.
Then $w(a,\ (\sigma s,\ s_L,\ r))$ is the *closed cylinder* with middle axis PQ and diameter d_L.

(The restriction to cylindrical elementary units is not inherent in the turtle geometry approach and could easily be overcome by the introduction of other geometrical elements, e.g. for leaves, flowers, fruits etc. — the current GROGRA version produces for the sake of simplicity only “pure” branching structures made up of cylindrical elements representing “shoots”.)

Now that we have defined the transition function t and the output function w, we can define recursively (as it is common for automata) the *induced transition function* on strings,

$$\tilde{t}: \quad M(\Sigma,\ \mu)^* \ \longrightarrow \ \mathcal{S},$$

by

$$\begin{aligned} \tilde{t}(\Box\,) &= (s_0,\ s_0,\ 1), \\ \tilde{t}(\sigma m) &= t(m,\ \tilde{t}(\sigma)) \quad (m \in M(\Sigma,\ \mu),\ \ \sigma \in M(\Sigma,\ \mu)^*), \end{aligned}$$

and the *geometrical interpretation* (in the simplified version for the undisturbing case)

$$i: \ \ M(\Sigma,\ \mu)^* \ \longrightarrow \ \mathcal{P}(\mathbb{R}^3) \times \mathbb{R}^p$$

by

$$\begin{aligned} i(\Box\,) &= (\emptyset,\ 0,\ 0,\ \ldots\ ), \\ i(\sigma m) &= w(m,\ \tilde{t}(\sigma)) \quad (m \in M(\Sigma,\ \mu),\ \ \sigma \in M(\Sigma,\ \mu)^*). \end{aligned}$$

The induced interpretation is now yielded by $(*)$ above (p. 27).

Remark. In GROGRA, the index "0" in the nullary symbols L_0, D_0, V_0, F_0, f_0, RV_0 is left away. This indexing was only made to distinguish them from the corresponding unary symbols to avoid a formal confusion. GROGRA registrates automatically whether an L, D etc. is followed by an argument or not.

Besides L, D, V there can be arbitrarily many further commands to change additional attributes in the same manner as ℓ, d and v. In GROGRA, there is N (in the variants N without argument and N, $N+$, $N*$, Nl, $Nl+$, $Nl*$ with argument) to manipulate a leaf surface attribute and P (only in the variants P without arguments and P as well as Pl with an integer argument) to specify a colour. Furthermore, there are global index-manipulating commands ($In =$, $In+ =$, $In* =$, $Jn =$, $Jn+ =$, $Jn* =$) with an integer n and a further argument, and method calls (Mn), which were excluded in this formal definition but could be included without great difficulty.
See Section 4.1 (p. 50) for a more intuitive explanation of the complete turtle language.

To ascertain the greatest possible degree of generality, we do not refer to turtle geometry in the following definitions of growth grammars. Any kind of geometrical interpretation of strings in the sense of Definition 5 will be permitted.

Definition 8: Let Γ be a fixed alphabet, called the set of *formal variables*, and (F, ν) a fixed parametric alphabet, called the set of *formal functions*.
A (non-sensitive, 1-phase, stochastic, conditional) *growth grammar* (with respect to Γ and (F, ν)) is a quintuplet

$$\mathcal{G} = ((\Sigma, \mu), \alpha, \mathcal{R}, I, \psi),$$

where (Σ, μ) is a parametric alphabet, $\alpha \in M(\Sigma, \mu)^*$ is a parametric string, called the *start string* (or *axiom*),

$$\mathcal{R} \subseteq \mathcal{C}(\Gamma) \times (\Sigma \times \Gamma^*) \times (\Sigma \times \mathcal{E}(\Gamma)^*)^* \times \mathcal{E}(\Gamma)$$

is a totally ordered, finite set of *rules* ($\mathcal{E}(\Gamma)$ and $\mathcal{C}(\Gamma)$ being the set of arithmetical, resp. conditional expressions over Γ w.r.t. (F, ν), see Def. 2, p. 24),

$$I : \quad M(\Sigma, \mu)^* \times \mathbb{N} \longrightarrow \mathcal{P}(\mathbb{R}^3) \times \mathbb{R}^p$$

($p \in \mathbb{N}_0$) is a geometrical interpretation (see Def. 5), and $\psi : \mathbb{N} \times \mathbb{N} \longrightarrow [0, 1]$ is an infinite matrix of random numbers, uniformly distributed in the interval $[0, 1]$. Each rule is written in the form

$$(\gamma) \;\; \lambda \longrightarrow \rho \;\; ?\pi,$$

where $\gamma \in \mathcal{C}(\Gamma)$ is the (possibly empty) *condition*, $\lambda \in \Sigma \times \Gamma^*$ the *left-hand side*, $\rho \in (\Sigma \times \Sigma(\Gamma)^*)^*$ the *right-hand side* and $\pi \in \mathcal{E}(\Gamma)$ the *probability expression*. Herein, the following restrictions must be fulfilled:

(1) No variable $x \in \Gamma$ appears twice in λ,

(2) if $\lambda = (a,\ \tau)$, $a \in \Sigma$, $\tau \in \Gamma^*$, and $\rho = (a_1,\ \varphi_1)\ (a_2,\ \varphi_2)\ \cdots\ (a_k,\ \varphi_k)$, $a_i \in \Sigma$, $\varphi_i \in \mathcal{E}(\Gamma)^*$, then $\ell(\tau) = \mu(a)$ and $\ell(\varphi_i) = \mu(a_i)$ $(i = 1,\ 2,\ \ldots\ k)$, i.e. the arities have to be respected,

(3) each formal variable $x \in \Gamma$ appearing in the expressions γ, ρ or π must also appear in λ, i.e. no *free variables* are permitted. (They come, however, indirectly into play via the formal function alphabet, $(F,\ \nu)$.)

$\lambda \in \Sigma \times \Gamma^*$ is written in the form $a(x_1,\ x_2,\ \ldots,\ x_{\mu(a)})$, and analogously each component of ρ in the form $a_i(e_{1,i},\ e_{2,i},\ \ldots,\ e_{\mu(a_i),i})$, where $a_i \in \Sigma$ and $e_{j,i} \in \mathcal{E}(\Gamma)$.

A rule is *potentially applicable* to a module $m = a(r_1,\ \ldots,\ r_{\mu(a)}) \in M(\Sigma,\ \mu)$ iff the following three conditions are fulfilled:

(a) λ matches with m (see Def. 3), defining the instantiation $b:\ \Gamma \longrightarrow \mathbb{R}$,

(b) $\hat{b}(\gamma) = true$,

(c) $\bar{b}(\pi) > 0$.

(Herein, the specification of a fixed function instantiation h on $(F,\ \nu)$ like in Def. 3, p. 25, is presupposed.)

Let $R_1,\ R_2,\ \ldots,\ R_n$ be the list of all rules which are potentially applicable to m, ordered according to the total ordering on $\mathcal{R}$. The rule R_k is *actually applicable* to m in the t-th step and in the s-th position $(s,\ t \in \mathbb{N}\,)$ if

$$(\mathrm{c}') \qquad \sum_{i=1}^{k-1} \bar{b}_k(\pi_k) < \psi(t,\ s) \leq \sum_{i=1}^{k} \bar{b}_k(\pi_k).$$

(Note that a potentially applicable rule is excluded from actual applicability if the sum of the evaluated probabilities of the potentially applicable rules coming "earlier" in the list exceeds 1. This implies especially that in a situation when several rules with probability 1 are potentially applicable, the first one is taken for actual application. — Note further that at most 1 rule can be actually applicable.)

The *result* of the *application* of the actually applicable *rule* $(\gamma)\ \lambda\ \longrightarrow\ \rho\ ?\pi$, where $\rho = a_1(e_{11},\ \ldots,\ e_{\mu(a_1)1})\ \cdots\ a_k(e_{1k},\ \ldots,\ e_{\mu(a_k)k})$, to the module $m = a(r_1,\ \ldots,\ r_k)$ is the parametric string

$$\rho_m\ :=\ a_1(\bar{b}(e_{11}),\ \ldots,\ \bar{b}(e_{\mu(a_1)1}))\cdots a_k(\bar{b}(e_{1k}),\ \ldots,\ \bar{b}(e_{\mu(a_k)k})),$$

where b is again the instantiation defined by the matching of λ and m.
The result of the *application of $\mathcal{G}$ to m* in the t-th step and in the s-th position $(s,\ t \in \mathbb{N}\,)$ is

$$\mathcal{G}_{t,s}(m)\ :=\ \begin{cases} \rho_m, & \text{if there is a rule } (\gamma)\ \lambda\ \longrightarrow\ \rho\ ?\pi \text{ actually applicable} \\ & \text{to } m \text{ in the } t\text{-th step and in the } s\text{-th position,} \\ m, & \text{if no such rule exists.} \end{cases}$$

That means that modules to which no rule is applicable remain unchanged.
The result of the *application of $\mathcal{G}$ to the parametric string* $\sigma = m_1 m_2 \cdots m_n \in M(\Sigma, \mu)^*$ in the t-th step ($t \in \mathbb{N}$) is the parametric string

$$\mathcal{G}_t(\sigma) \;:=\; \mathcal{G}_{t,1}(m_1)\,\mathcal{G}_{t,2}(m_2)\;\cdots\;\mathcal{G}_{t,n}(m_n) \in M(\Sigma,\ \mu)^*.$$

The *sequence of strings generated by* $\mathcal{G}$ is the infinite sequence of parametric strings $(\sigma_0,\ \sigma_1,\ \sigma_2,\ \ldots) \in (M(\Sigma,\ \mu)^*)^{\mathbb{N}_0}$ with

$$\begin{aligned} \sigma_0 &= \alpha, \\ \sigma_t &= \mathcal{G}_t(\sigma_{t-1}) \quad (t \in \mathbb{N}). \end{aligned}$$

The *sequence of structures generated by* $\mathcal{G}$ is the infinite sequence of attributed geometrical structures

$$(\tilde{I}(\sigma_1),\ \tilde{I}(\sigma_2),\ \tilde{I}(\sigma_3),\ \ldots) \in G_p^{\mathbb{N}}$$

where $\tilde{I}$ is the interpretation on $M(\Sigma,\ \mu)^*$ induced by I.

Remark. Nondeterminism is excluded from Definition 8 by the requirement of an ordering of the rule set and by the inclusion of the fixed random number table ψ. The rule choice process could be controlled by another strategy if these requirements are dropped. In the actual GROGRA implementation, the reference to the number table ψ is replaced by a runtime call to a pseudorandom number generator each time (c′) is evaluated.

Definition 9: Let Γ and $(F,\ \nu)$ be fixed alphabets like in Definition 8. A (non-sensitive, stochastic, conditional) *2-phase growth grammar* (with respect to Γ and $(F,\ \nu)$) is a 7-tuplet

$$\mathcal{G} = ((\Sigma,\ \mu),\ \alpha,\ \mathcal{R}_1,\ \mathcal{R}_2,\ I,\ \psi_1,\ \psi_2)$$

such that $\mathcal{H} \;:=\; ((\Sigma,\ \mu),\ \alpha,\ \mathcal{R}_1,\ I,\ \psi_1)$ is a 1-phase growth grammar, $\mathcal{R}_2 \subseteq \mathcal{C}(\Gamma)\times(\Sigma\times\Gamma^*)\times(\Sigma\times\mathcal{E}(\Gamma)^*)^*\times\mathcal{E}(\Gamma)$ is a totally ordered, finite set of rules fulfilling the same restrictions as $\mathcal{R}_1$ (see Definition 8), and $\psi_2 :\ \mathbb{N}\times\mathbb{N} \longrightarrow [0,\ 1]$ is another random matrix like ψ_1.
The elements of $\mathcal{R}_1$ are called *generative rules* (or first phase rules), the elements of $\mathcal{R}_2$ *interpretative rules* (or second phase rules).
Potential and actual applicability of an interpretative rule to a module m and the result of its application are defined in the same manner as for generative rules, taken ψ_2 substituted for ψ_1.
The result of the *interpretative application* of $\mathcal{G}$ to m in the t-th step and in the s-th position ($s,\ t \in \mathbb{N}$) is

$$\tilde{\mathcal{G}}_{t,s}(m) \;:=\; \begin{cases} \rho_m & \text{(see Def. 8), if there is an interpretative rule} \\ & (\gamma)\ \lambda \ \longrightarrow\ \rho\ ?\pi \text{ actually applicable to } m \\ & \text{in the } t\text{-th step and in the } s\text{-th position,} \\ m & \text{if no such interpretative rule exists.} \end{cases}$$

The result of the interpretative application of $\mathcal{G}$ to a parametric string $\sigma = m_1\, m_2 \cdots m_n \in M(\Sigma,\ \mu)^*$ in the t-th step ($t \in \mathbb{N}$) is the parametric string

$$\tilde{\mathcal{G}}_t(\sigma) \ :=\ \tilde{\mathcal{G}}_{t,1}(m_1)\,\tilde{\mathcal{G}}_{t,2}(m_2)\cdots\tilde{\mathcal{G}}_{t,n}(m_n) \in M(\Sigma,\ \mu)^*.$$

The *sequence of structures generated by* $\mathcal{G}$ is the infinite sequence of attributed geometrical structures

$$(S_1,\ S_2,\ S_3,\ \dots) \in G_p^{\mathbb{N}}$$

derived from the sequence of strings generated by the corresponding 1-phase grammar $\mathcal{H}$ through $\sigma'_t \ :=\ \tilde{\mathcal{G}}_t(\sigma_t)$ and $S_t \ :=\ \tilde{I}(\sigma'_t)$ for $t = 1,\ 2,\ 3,\ \dots$ (cf. Fig. 9, page 21).

Definition 10: Let Γ be a fixed alphabet, called the set of formal variables. Let $(F,\ \nu)$ be a parametric alphabet, the set of formal functions, and let F be splitted into $F = F_N \cup F_S$ ($F_N \cap F_S = \emptyset$). F_N is called the set of *nonsensitive function symbols*, F_S the set of *sensitive function symbols*. Let h_N be a mapping on F_N, associating to each $f \in F_N$ a function

$$h_N(f):\ \ \mathbb{R}^{\nu(f)} \longrightarrow \mathbb{R} \cup \{undef\},$$

and h_S a mapping on F_S, associating to each $f \in F_S$ a function

$$h_S(f):\ \ (\mathcal{P}(\mathbb{R}^3) \times \mathbb{R}^p) \times G_p \times \mathbb{R}^{\nu(f)} \longrightarrow \mathbb{R} \cup \{undef\}.$$

A (1-phase) *sensitive growth grammar* (with respect to Γ and $(F,\ \nu)$) is a 7-tuplet

$$\mathcal{G} = ((\Sigma,\ \mu),\ h_N,\ h_S,\ \alpha,\ \mathcal{R},\ I,\ \psi),$$

such that $((\Sigma,\ \mu),\ \alpha,\ \mathcal{R},\ I,\ \psi)$ fulfills the formal requirements of a non-sensitive growth grammar (see Def. 8). The specification of the interpretation of function symbols from F_S, necessary to define potential / actual applicability of rules and application results, is given recursively. Let $f(e_1,\ \dots,\ e_k)$ appear in an expression on the r.h.s. of a rule, $f \in F_S$, $e_1,\ \dots,\ e_k \in \mathcal{E}(\Gamma)$. The *evaluation of* f *in the* t*-th step with respect to the module* $m_s \in M(\Sigma,\ \mu)$ *in the string* $\sigma_{t-1} = m_1 \cdots m_n \in M(\Sigma,\ \mu)^*$ ($t,\ s,\ n \in \mathbb{N}$, $s \le n$) is

$$\bar{b}(f(e_1,\ \dots,\ e_k)) = \begin{cases} h_S(f)\ ((\emptyset,\ 0,\ 0,\ \dots),\ \emptyset,\ \bar{b}(e_1),\ \dots,\ \bar{b}(e_k)) & \text{if } t = 1, \\ h_S(f)\ (I(\sigma_{t-1},\ s),\ \tilde{I}(\sigma_{t-1}),\ \bar{b}(e_1),\ \dots,\ \bar{b}(e_k)) & \text{if } t > 1. \end{cases}$$

For $f \in F_N$, the evaluation is done in the standard manner (see Def. 3, p. 25). In this recursive definition, $(\sigma_0,\ \sigma_1,\ \sigma_2,\ \dots)$ is the sequence of strings generated by $\mathcal{G}$, defined in the same way as in Definition 8.

Note that the calculation of the result of the application of a rule to a module appearing in string σ_t depends on the precedent structure $\tilde{I}(\sigma_{t-1})$ and on the elementary unit $I(\sigma_{t-1},\ s)$ associated to the module in this structure (Fig. 13; see also Fig. 11 on p. 23).

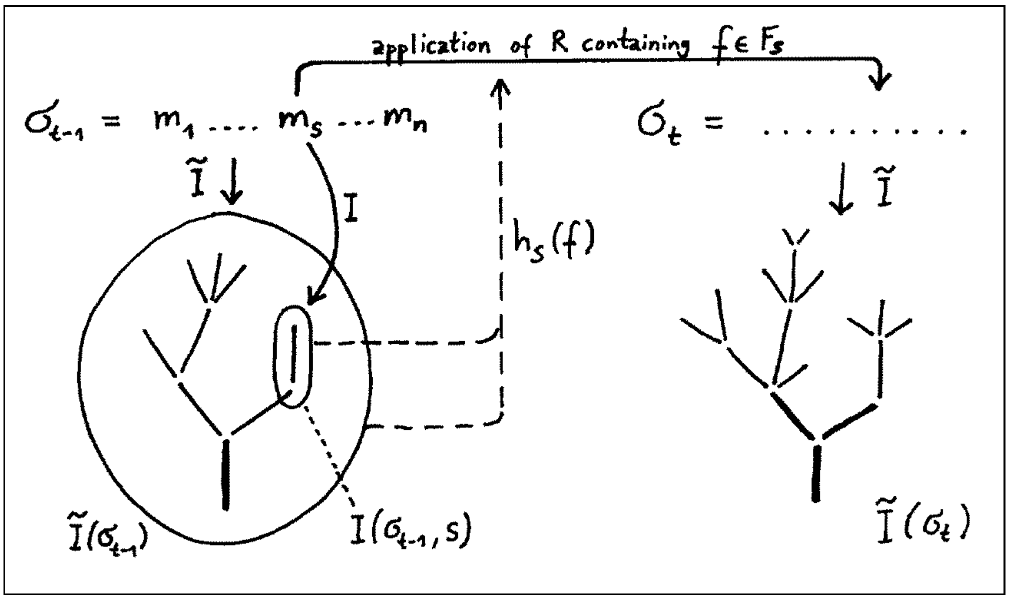

application of R containing f ∈ F_S
σ_{t-1} = m_1 .... m_s ... m_n
σ_t = ...
Ĩ
I
h_S(f)
Ĩ
Ĩ(σ_{t-1})
I(σ_{t-1}, s)
Ĩ(σ_t)

Fig. 13: Intervention of a sensitive function during the rewriting step leading from string σ_{t-1} to string σ_t

In GROGRA, some fixed sensitive functions are implemented; see Section 4.3 (p. 68) for details.

The combination of sensitivity with the 2-phase process leads to the most general definition:

Definition 11. Let Γ, $(F,\ \nu)$, F_N, F_S, h_N, h_S be specified as in Def. 10. A *sensitive 2-phase growth grammar* (with respect to Γ and $(F,\ \nu)$) is a 9-tuplet

$$\mathcal{G} = ((\Sigma,\ \mu),\ h_N,\ h_S,\ \alpha,\ \mathcal{R}_1,\ \mathcal{R}_2,\ I,\ \psi_1,\ \psi_2),$$

such that $((\Sigma,\ \mu),\ \alpha,\ \mathcal{R}_1,\ \mathcal{R}_2,\ I,\ \psi_1,\ \psi_2)$ fulfills the formal requirements of a non-sensitive 2-phase growth grammar (see Def. 9). Furthermore, it is required that no function symbol from F_S appears in the r.h.s. of an interpretative rule (i.e., a rule from $\mathcal{R}_2$). Applicability and application of interpretative rules are defined as in Def. 9 and are independent from the generated structure sequence.

Let $\tilde{\mathcal{G}}_{t-1,s}(m_s)$ be the result of the application of the interpretative rules of $\mathcal{G}$ to the module $m_s \in M(\Sigma,\ \mu)$ in the $(t-1)$-th step, and let the last module of the string $\tilde{\mathcal{G}}_{t-1,s}(m_s)$ have position q in the string

$$\sigma'_{t-1} = \tilde{\mathcal{G}}_{t-1}(\sigma_{t-1}) = \tilde{\mathcal{G}}_{t-1,1}(m_1) \cdots \tilde{\mathcal{G}}_{t-1,n}(m_n).$$

(Set q to the position of the nearest preceding module if $\tilde{\mathcal{G}}_{t-1,s}(m_s)$ is empty.) Then we define the evaluation of an $f \in F_S$ in the t-th step w.r.t. the module m_s in the string σ_{t-1} in the same way as in Def. 10 for 1-phase sensitive grammars, but we replace in that definition $I(\sigma_{t-1},\ s)$ by $I(\sigma'_{t-1},\ q)$, and $\tilde{I}(\sigma_{t-1})$ by $\tilde{I}(\sigma'_{t-1})$.

This section had the purpose to show that a precise mathematical definition of the grammars forming the basis of GROGRA is possible, and to provide such a definition. It had not the aim to prove general mathematical statements about these grammars or about the structures they generate. This would be subject of another work. (See, e.g., [62] or [63] for some related work.)

Furthermore, it is not the aim of this general treatise to discuss special, botanically motivated concepts of plant behaviour, of stochastic growth processes etc. The introduced growth grammars offer a rather general framework for such models. Analogously, in a textbook on the foundations of calculus, one will find a lot about ϵ-neighbourhoods, continuity, differentiability and related stuff, but not necessarily the applications to physics, celestial mechanics or other sciences which motivated the development of infinitesimal calculus. — The examples in Chapter 6 will give some hints how the formal framework of growth grammars can be filled with special modelling concepts more closely related to botany.

In the definitions given above, the possible use of the *repetition operator* on the r.h.s. of a rule was excluded to avoid a too complex syntax specification of rules. See Section 4.5 (p. 73) for a detailed description of the repetition operator.

Chapter 3

Quick guide for using GROGRA

3.1 Installation

There exist two versions of GROGRA 2.4, one running on an IBM-compatible microcomputer, the other on a Silicon Graphics workstation (Iris Indigo). (A third version for Sun and other workstations is currently being developed.) The executable file is grogra.exe for the micro version, grogra for the workstation version. Both versions require additionally the file lexpla.msg, containing some short explanation texts on the growth grammar syntax.

Further requirements for the micro version are: EGA- or VGA-card (or other graphics card supported by the Borland Graphics Interface, see [15]), mouse (optional), printer at LPT1 (optional), MS-DOS 3.3 or higher, Borland Graphics Interface file (egavga.bgi or other driver software, depending on the graphics card). The bgi file is expected in the subdirectory \tc\bgi, the file lexpla.msg in the subdirectory where GROGRA is called. There is some amount of free space on the disk required, because GROGRA creates some auxiliary files during work. Their size depends on the application for which GROGRA is used. The same holds for the RAM requirements.

Grammar files (having the file name suffix .lsy or .ssy) and any other input files to be used by GROGRA are normally expected in the same subdirectory where GROGRA is called.

If, e.g., GROGRA is to be installed from a floppy disk together with the example files koch.lsy, examp.lsy and dichomur.ssy on a PC with VGA card, the following files should be created on the harddisk by copying them (the subdirectory tc\gg could be exchanged by another path):

```
\tc\bgi\egavga.bgi
\tc\gg\grogra.exe
\tc\gg\lexpla.msg
\tc\gg\koch.lsy
\tc\gg\examp.lsy
\tc\gg\dichomur.ssy
```

(On the UNIX workstation, \ has of course to be replaced by / , and the file egavga.bgi is not needed.)
To manipulate the input files, additionally some text editor is necessary.

3.2 Usage

This section will explain the most important features of GROGRA 2.4 and how they are invoked. A detailed description of all possible options will be given in Chapter 4 and 5.

The first thing to do is normally to write a file containing the grammar that GROGRA is meant to interprete. This must be a simple ASCII textfile without additional formatting (ensure that the option "text only" or something similar is chosen when manipulating the file with WORD or another editor). The filename must end with .lsy in the case of a grammar not using sensitive functions or parameters, and .ssy in the case of a sensitive grammar.

Let us assume we have written with a text editor a file named base.lsy consisting of the following 6 lines:

```
\var lg length,
\angle 50,
* #  L50  a(1,100),
(t<m)  a(t,m) #  s  [  @0.8  +  L*0.5   a(1, min(t+1,m-t))  ]
    [  @0.9  –  L*0.6  a(1, min(t+1,m-t))  ]  L*0.9  a(t+1,m),
s ##  N(lg* 80)  F
```

The first two of these lines contain directives — a variable declaration and the initialization of the angle associated to the symbols + and − in the rules — , the other lines contain rules, where the symbol # stands for the arrow $\longrightarrow$ separating l.h.s. and r.h.s. of the rule. Line 5 continues the r.h.s. of the rule beginning in line 4. The last rule is an interpretative rule, indicated by the double rule sign ##.

No probabilities are specified in this grammar — they are all assumed to be 1 —, but the second rule has a condition, $t < m$.

Note that the directives and the rules must be separated by commas, and that all symbols must be separated by blanks. Especially, the blanks preceding and following the signs [and] must not be omitted. — A complete description of the growth grammar syntax will be given in Chapter 4.

To see what sequence of geometrical structures is generated by this grammar, we will start GROGRA.

We type grogra at the DOS (or UNIX) prompt. When the software is installed correctly, there will now appear a display with the heading

```
******** G R O G R A ********
G R O W T H   G R A M M A R S
```

and a small text, ending with the invitation

```
Start of program:
      O.K.
```

The box containing "O.K." will become green if we move the mouse cursor inside it. Thereby we can test whether the mouse is working. If not, GROGRA is also capable to execute most tasks without the mouse. To start GROGRA, we can either press the left mouse button when the cursor is inside the box or press any key on the keyboard.

Now we arrive at the *main menu* of GROGRA, which is entitled "Please make a choice". It contains 8 items which can be selected by moving the mouse (thereby underlying the selected item with red colour) and pressing the left mouse button, or by pressing the key with the letter indicated in the menu.
Normally, the first choice is the first item,

```
a   Generation of a new branching structure.
```

When we choose this, there appears a submenu "Way of making the structure:", listing several input possibilities, the most important of which being the third and fourth one. As our file base.lsy contains no sensitive functions or variables (and has already the suffix .lsy), we choose the third:

```
non-sensitive growth grammar  <N>,
```

either by using the mouse or by typing "n".
After that, there will appear a list of the available .lsy-files in the current subdirectory. If there are more than 15 such files, we get a request "Press any key" to see the next part of the list (possibly several times, if the list is very long). Our file "base.lsy" should appear in this list. Next, there is the request

```
Name of the L-system file (without extension):
```

Here, we have to type base and to finish the text input with the return-key.

If we have made a mistake, or if the file is not there or erroneous, there will appear the information

```
L-system file not existing or not accessible.
Press <return> to continue.
```

and we come back to the main menu.
However, if everything is alright, there appears the message

```
L-system successfully read
```

and we are asked for the *start word*. As in the other examples, this is the word consisting of the single symbol $*$ here (look for the $*$ character on the keyboard). We have to give in this start symbol and a <return>. Then we are asked

```
How many steps are to be executed?
```

Here, the number of developmental steps has to be specified (corresponding to t in Section 2.2). One has generally to be careful not to choose a number too high because several example grammars imply some exponential growth behaviour, leading to long calculation times and memory shortcuts if the number of steps is too high. Let us assume we choose 9 steps here.

The next request is

```
Execution only of specified developmental steps (give <s> and <return>)
or of all steps (any other input):
```

When we want to create all developmental steps from 1 to 9, we can simply press the return key at this request.

This will start the structure generating process, consisting of (generative) grammar application, interpretative application and geometrical interpretation. The geometrical structures corresponding to the 9 developmental steps will be created in the memory, as it will be indicated by "Structure is generated" after each step. However, we see no graphical result in this moment, because the graphics display mode is not activated.
The creation of the structures can take considerable time in the case of grammars describing very complicated and large plants, or when a high number of steps is chosen. It is finished when the message

```
Prescribed number of steps is done.
Execution of this L-system finished.
Press <return> to continue.
```

appears.
Afterwards, we will be back in the main menu, but the sequence of structures is now present in the background and can be watched, listed, stored, transformed or analyzed.

To get a visual impression of our structures, we have to choose

e Show the actual structure graphically.

We arrive at a small submenu "Direction of view". Let us choose the first item, "side view <A>". On the workstation, there is also the possibility to select the display size ("standard format" / "full screen"). Next, we get some informations, entitled "Now there will be the graphical output". The numerical values of the screen limitations are not important for us; the visual area is automatically adapted to the maximal extensions in x- and y-direction appearing in the generated structure sequence.
When we press the return key, we come into the *graphical display mode*. In our case, with the structure generated from base.lsy still loaded, we will first see an empty screen with a small number "1" in the upper left corner. This is the structure derived from applying the first rule to the start symbol $*$ — namely, the empty structure, because the r.h.s. of this rule contained no F command which would generate a visible unit.

With the <space> key, we can proceed to the second, third... developmental step, which correspond to non-empty structures now. The number of the step is always indicated in the upper left corner. After step 9 — the structure with the highest number created —, we come back to the main menu. We can also interrupt the display sequence before by pressing <return>. Or, by pressing the "c" key, we can tell GROGRA to proceed automatically through the sequence of structures, showing each one for a fixed interval of one second and beginning again with the first one when the last is shown. (We can stop this automatic display by pressing the space key or, if we want to leave the graphics display completely, the return key.)
The structures generated from base.lsy should look like those in Fig. 14.

When one structure is displayed in the graphical display mode, we can decide to create a Postscript file to *print* this structure on a remote printer, to which the file must be sent after we have left GROGRA. This is simply done by typing "p". (GROGRA asks for the paper format and for the file name afterwards.) Or we can produce an online hardcopy from the screen on a directly connected printer by typing "d" (currently only in the PC version possible). There is also the possibility to create a HPGL file for plotting by typing "h" (see Section 5.3 for further informations).

In the graphical display mode, we can also decide to look more closely at certain parts of a structure by "zooming" into that structure. This is done by typing "z". After that, one has to click at two positions with the mouse — the first one specifying the lower left corner of the new visible part, the second the upper right corner. (This feature does only work with the mouse.) The zooming can be iterated several times. To re-establish the original scaling, it suffices to press "e".

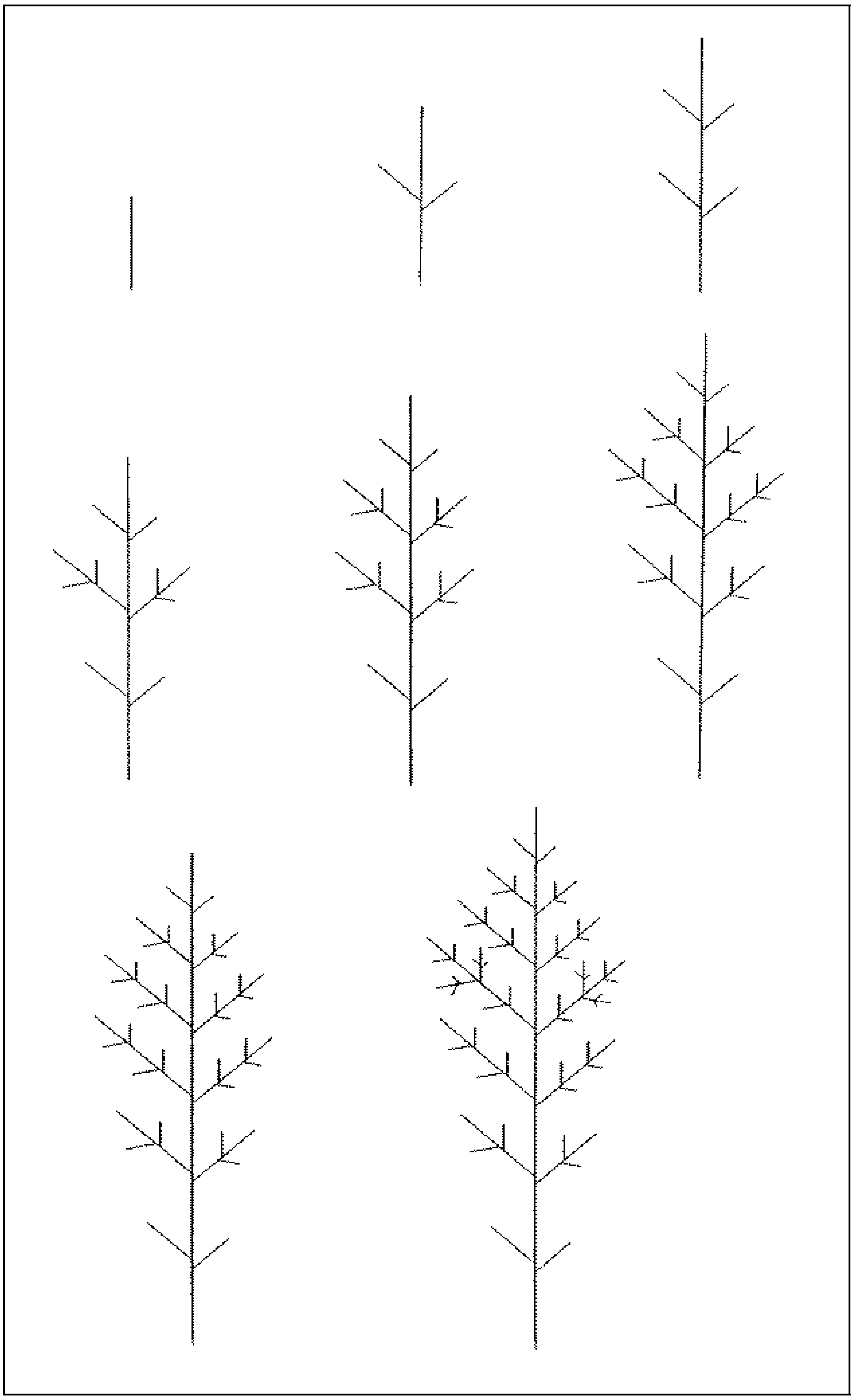


Fig. 14: Sequence of structures generated from base.lsy

Our growth grammar base.lsy does only work in two dimensions, so it doesn't make much sense to look at its results from *different spatial positions*. But for other grammars, this possibility can be useful. We can choose in the submenu "Direction of view" the third item, "arbitrary direction of view <C>". Then we are asked for an angle in the xy plane (y-direction = 0), and afterwards for another angle, specifying the slope of view relative to the horizontal direction. Both values have to be given in degrees and finished with <return>. Thereafter, the whole developmental sequence will be seen in the changed perspective. (The projection is always a parallel projection on a plane orthogonal to the direction of view. See Section 5.3 for further details.)

However, not all informations contained in the created structures are visible in the graphics display. There are *attributes* which can be determined by the grammar (e.g. the N command in the last rule of base.lsy controls the "leaf parameter" n for each elementary unit building up the structure). To look at these attributes, and also to get known the exact values of the lengths, positions, diameters etc. of all units, there is the possibility to "List the actual structure" (item c in the main menu).

Having chosen this item, we are first asked for the output device, which can be the screen, a connected printer, or a file. Because the generated structures can be rather large, it is useful to have first a look at the list on the screen before writing to the printer or to a file.
The next submenu is entitled "Output of which structures / shoots:", and one can choose between "all <A>" and "beginning from a specified structure / shoot <S>". (Here and in the following, "shoot" is used synonymously to "elementary unit" in the sense of Section 2.2.) Let us choose the second alternative. Then we are asked for a number indicating the developmental step to start with. This can be an integer between 1 and 9, in our case.
Let us, e.g., type "3" (finished with return). Next, we have to specify a shoot number, let us say, "1". After that, we obtain a screen page with a lot of numerical informations, entitled "Structure number 3: Shoot number 1, emerging from shoot number -2:". All informations on this page refer to the first unit (the basic stem section) of the third structure in our developmental sequence. (Remind that the first structure is empty, so Structure number 3 corresponds in fact to the second time step where something can be seen.) All information about this unit which is laid down in the memory is given on this screen page. We mention only the length (50.000000 length units), which corresponds to the $L50$-command in the first rule of base.lsy, the diameter, which is zero because no D-command appeared in base.lsy, and the leaf parameter (n), which is put to the value 4000 by the last rule of base.lsy ($50 * 80 = 4000$). This parameter could, e.g., stand for leaf surface in mm^2. By the way, the irritating information *"emerging from shoot number -2"* will only tell us that the basic shoot has no mother shoot at all. Further details about the informations on this screen page can be found in Section 5.1, p. 79.
To control the next elementary unit of structure number 3 in the same way,

one has only to press <return>. This shoot has number 2 and emerges from number 1. It establishes a side branch with length 25 and was created from $a(1,\ \min(t+1,\ m-t))$ in the second rule of base.lsy. The length is 50/2 because this module was preceded by the turtle command $L * 0.5$.
The structure has two further elementary units which can be shown by pressing <return> twice. Then, when we press <return> once more, we arrive at structure number 4, which begins again with a basic shoot of length 50. At each shoot, we can leave the list display by simply typing "q <return>", which brings us back to the main menu.
When we choose to list "all" structures, the list will begin with structure number 1, which is empty in our case.

GROGRA has several *analysis* options which can provide further informations about the actual structure. Let us choose the item

f Analyze the actual structure

in the main menu. There appears a submenu "Options of analysis:" with again 8 items. A detailed discussion will be found in Section 5.2 (p. 98). We start only the "elementary analysis" here (first item). Again, the output medium is to be specified — let us choose the screen.

There appears a text screen entitled "Basic Data" where we find informations about Structure number 1. All numerical values (with the exception of the last one) will be zero here, because our structure number 1 is empty. By pressing <return>, we get the corresponding informations for the next structure, and so on. From structure 2 on, also the increments in relation to the preceding structure are given. Thus we can e.g. see that our "tree" has reached a height (maximal z coordinate) of 284.766418 length units in step 9, and that this is an increment of 23.914856 length units compared to step 8. Also, the n-parameters (leaf surfaces) and the shoot volumes are summed up — the latter sum being always zero here, because we have excluded diameter growth from our grammar. (See Chapter 6 for other examples, including such with real thicknesses.)

Once a sequence of structures is created, there arises possibly the wish to *store* it in a file instead of creating it again from the .lsy-file when it is needed once more. This can be done under the main menu item

d Store or transform the actual structure.

There are several data formats (see Section 5.5) which can be selected in the appearing submenu. The standard format is invoked by the first menu item, "Save in standard format (.dta) <A>". This option takes care for saving the whole developmental sequence (steps 1 to 9 in our case). Instead, we can also decide to save only one developmental step, e.g. the most complicated, number 9. Then we would have to choose the second menu item, "Save in .dta format,

only one developmental step <E>", and afterwards to specify the number of the structure to be saved, here 9.
In either case, we have afterwards to give in a filename, which can be arbitrary (but not longer than 8 characters under DOS) and which is automatically extended by the suffix ".dta".
Attention: If a file with the specified name does already exist, it is overwritten without warning!
When GROGRA has finished saving, we are informed by the message "Saving finished, press <return> to continue." Then we are in the main menu again, and the structure just saved is still present in the memory. It will be deleted only when we create a new structure (item a), when we read another structure from a file (item b), when we change the internal mode of memory (see below), or when we leave GROGRA (item q: Quit the program).

The complementary process to saving is *reading*. If we have saved the last developmental step, using option <E> above, in a file named base9.dta, we want perhaps to read from this file. To this purpose, we have to choose the main menu item

b   read a structure from a file.

Again, we find ourselves confronted with a submenu specifying several data formats. Here, we must choose the first item, "Standard format (format .dta) <A>". (Another format is, e.g., the descriptive format .dtd (fourth item), which is used to read data from morphological measurements to analyze and compare them with artificially produced structures. dtd-files can not be created by GROGRA, they must be written "by hand" and can only be read and interpreted.) Having chosen <A>, we get a list of available .dta-files, like in the case of the .lsy-files above. Then we have to type in our "base9".
In the case of large structures, the reading process can cost considerable time. (In fact, there are sometimes situations when it is more economical to create a structure once more from the lsy- or ssy-file instead of reading it from a dta-file.) The completion of reading is signalled by the message

"reading process finished.
Press <return> to continue."

When we now invoke the graphical display, we will see our former structure number 9 now as the single number 1 — as was to be expected, because we didn't save the other developmental steps. We can work with this structure in the same way as with a structure generated directly from a grammar file.
The .dta-files can also be read by a text editor (they are simple ASCII files). They contain essentially the same information that is listed in the list-option explained above, but in a more comprimed manner.

When we try to generate on the microcomputer a very large structure, let us say, base.lsy with 15 steps, there can possibly occur the error message

> Not enough space for new segment. Structure cannot be finished. Press <return> to continue.

during the grammar evaluation and structure creation. The structure will then be created up to the point where the memory deficiency occurred. (In the display, this will probably cause "holes" and missing parts.)

To avoid this error, there is a possibility to *change the internal mode* how structures are stored. Normally, this is done in "RAM mode", i.e. the structures exist as chained lists of data records in the main memory of the computer. But there is also a "HD mode" where structures are hold as sequential lists in a file on the hard disk. This method provides much more space for large structures. However, it has the disadvantages that the computation time is longer and that not all GROGRA features work in this mode (see Section 5.4, p. 108, for details.) E.g., it is not possible to use sensitive grammars in HD mode.
To change to HD mode, we have to choose the main menu item

> w Service functions and explanations.

(Eventually, we have to save our structure first in a dta-file, because it will be deleted when we change the mode of memory.) The submenu "Explanations and Service" contains 7 items. The 6th of them, "Change the internal mode of memory <S>", acts like a switch: When we are in RAM mode, we come into HD mode, and vice versa. We are requested to confirm this change with an "O.K." like in the start display of GROGRA. When we are in HD mode, we can again invoke the "Generation of a new branching structure", and there will be no apparent difference to the default RAM mode, except perhaps a certain slowness and the loss of some features in the menues.

The "Service functions and explanations"-menu can also provide us with some short informations about turtle commands, expression syntax and grammars (first three menu items) if, for example, this manual is not at hand. There is also the possibility to switch between an English and a German version of the program (menu item "Change the language <A>"). The default is English.

Chapter 4

Growth grammar syntax and semantics

This chapter contains all informations necessary to write an .lsy- (non-sensitive growth grammar-) or an .ssy- (sensitive growth grammar-) file. Although most semantical features were principally already defined in Chapter 2, the complete syntax and semantics will be given here independently of the mathematical formalism of Chapter 2.

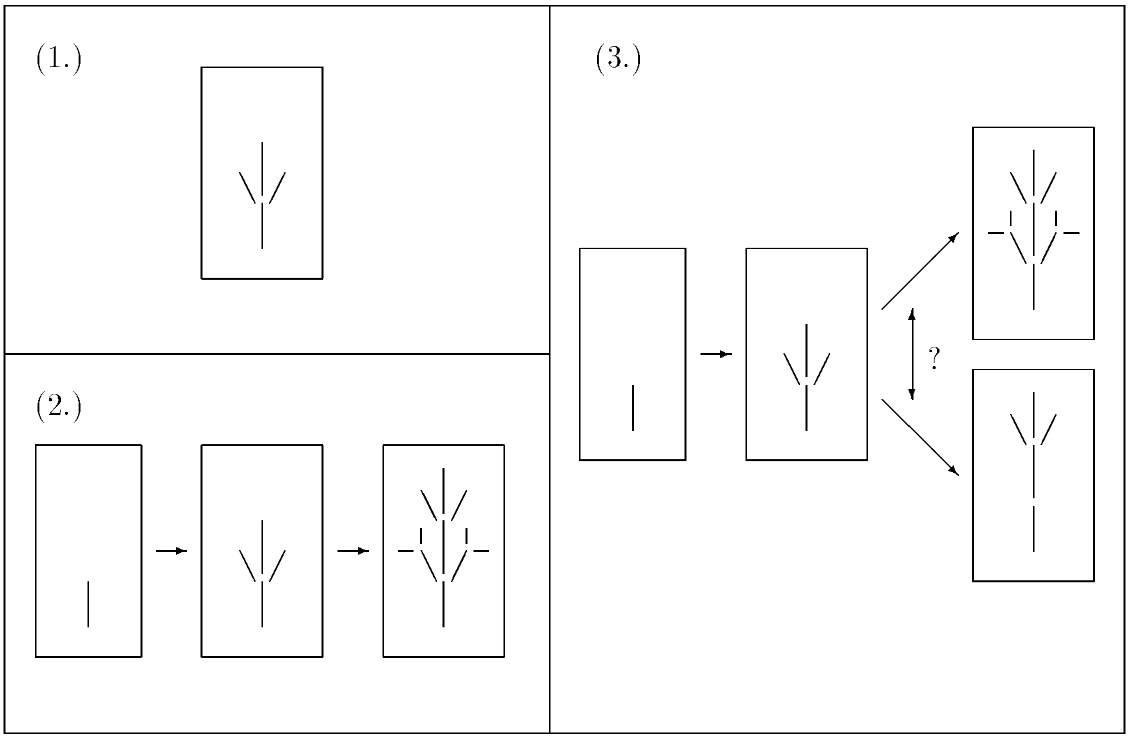


Fig. 15: Three levels of formal description

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 (see Fig. 15):

(1.) The specification of a geometrical structure (possibly with additional, non-geometric attributes) at a fixed moment in time. This is achieved in the GROGRA system by *turtle geometry*.

(2.) The specification how these structures develop in time. This is done by deterministic *rule systems* (L systems).

(3.) The control which of several applicable rules is actually applied at a given instance. This control is carried out in the framework of *growth grammars* with the help of features like *stochastic rules*, *random variables*, *parametrization*, *conditions* and *sensitivity*.

In our explanation of growth grammars as a formal tool for the specification of structural development, we will roughly follow this classification.

4.1 Turtle commands

When we neglect the dynamical aspect, the objects which GROGRA creates are simply *branching structures* in 3-dimensional space which are built up of *cylindrical units* (possibly of diameter 0), each of which can bear additional, non-geometric *attributes* (Fig. 16). We speak of (*attributed*) *geometrical structures*. The cylinders (rectangles in Fig. 16) are called *elementary units* or *shoots*. Each unit has at most one *mother unit* from which it emerges.

Turtle geometry [1] is a formal tool to describe such structures. It could be extended to include further geometrical possibilities like surface constructions ([98], [79]) or simply other fixed geometrical objects additional to cylinders, but these possibilities are actually not implemented in GROGRA 2.4.

Turtle geometry is based on the conception that there is a device, called the *turtle*, which can be told by *commands* to move, to change its orientation in space, and to *construct elementary units with given attributes* while moving.

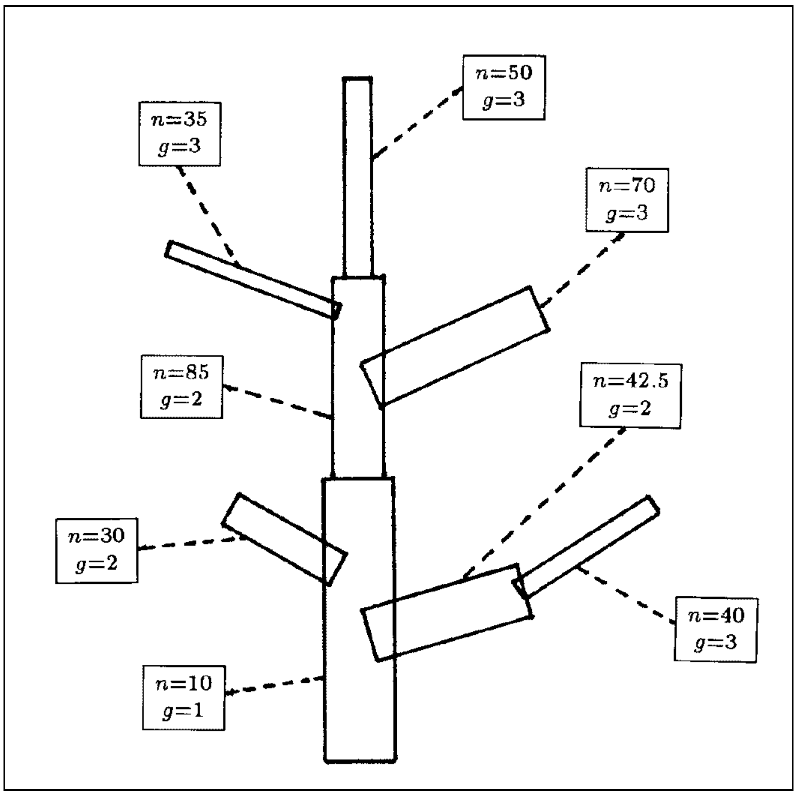


Fig. 16: An attributed geometrical structure

The *state* of the turtle is a data record containing all relevant information on its position, orientation, attributes to be used in the next construction step, etc. More precisely, the *state variables* of the turtle are:

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

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

$L = (l_x,\ l_y,\ l_z)$ (= *left*), a vector of unit length, orthogonal to H, specifying a direction in three-dimensional space to be considered as "left" with respect to the turtle,

$U = (u_x,\ u_y,\ u_z)$ (= *up*), a vector of unit length, orthogonal to H and L, specifying a direction in three-dimensional space to be considered as "upwards" 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 non-negative 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, see below) is given,

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

p, an integer specifying the *colour* of the next elementary unit (in the current GROGRA version, the colour is encoded according to the EGA colour table with 16 colours, see Appendix 1),

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

m, a reference to the *mother unit* of the next elementary unit which will be constructed,

b, an integer, giving the branching *order* of the next unit which will be constructed,

g, an integer, called the *generative distance*, indicating the number of units between the next unit and some mother-less "root unit",

r, an integer, called the *relevance counter*, which prevents the creation of new units when it is smaller or equal to zero.

Furthermore, there are some "shadow variables", called the *local state variables*, namely, ℓ_L, d_L, v_L, n_L and p_L. In fact, it is this local variable set which is used when a new elementary unit is constructed. These variables are normally identical to their counterparts ℓ, d, v, n, p. The only way to change their values independently is to use the so-called *local commands* (bearing the specifyer ℓ) explained below. After each usage, the local variables are re-identified with their ordinary counterparts, such that their intentional alteration has only consequences which are restricted to the next constructed elementary unit (or to the next *RV*-command, in the case of v_L). Therefore the name "local state variables". Some of the state variables can be changed by specific commands, others are changed automatically during the action of the turtle.

Before discussing the possible commands, let us specify the **default values** of the state variables which are assumed before any command is given (and which can be re-installed by some commands):

$$P = (0,\ 0,\ 0),$$
$$H = (0,\ 0,\ 1),\ \ L = (-1,\ 0,\ 0),\ \ U = (0,\ -1,\ 0),$$
$$\ell = \ell_g,\ \ d = d_g,\ \ v = v_g,\ \ n = n_g,\ \ p = p_g,$$
$$\ell_L = \ell_g,\ \ d_L = d_g,\ \ v_L = v_g,\ \ n_L = n_g,\ \ p_L = p_g,$$
$$q = 0,$$
$$m = \text{NULL (no mother unit)},$$

$b = 0$,
$g = 1$,
$r = 1$.

The "global" default variables ℓ_g, d_g, v_g, n_g, p_g can — together with a further variable, w_g, specifying an angle in degrees — be defined in the *directive part* of a grammar file (i.e., preceding the rules) by the \set *statement*:

\set L x,
\set D x,
\set V x,
\set N x,
\set P i,
\set W x,

Here, x stands for a real number to be specified in decimal notation, and i for an integer. (Note that the upper-case letter must be preceded and followed by exactly one blank, and that the statement must be finished by a comma. Upper and lower case letters have generally to be distinguished, like in the programming language C.) After \set L x, the variable ℓ_g has the value x, and analogously for the other variables. Synonymous to \set L x, is the statement \length x, or \laenge x, and synonymous to \set W x, is \angle x, or \winkel x,. An alternative form of assignment is provided by the \ask *statement*, which enforces a request to the user at runtime:

\ask L *question text*,

(and, analogously, with D, V, N, P or W). Here, the *question text* is an arbitrary text which may contain blanks, but which must not contain any comma.
If no \set or \ask statement provides a value, the global variables (and with them, also the turtle's state variables) have the following default values:

$\ell_g = 100$,
$d_g = 0$,
$v_g = 0$,
$n_g = 0$,
$p_g = 14$ (= yellow),
$w_g = 90$.

Let us now consider the turtle commands which change the value of one (or more) state variable(s) while the turtle is at work. An overview is given in Table 1. For x, one has to substitute some floating point number. The parentheses ( ) enclosing the argument can in all cases be omitted. (This is no longer true if formal expressions are substituted for x in the context of parametric rules, see Section 4.3 below.)

Table 1: Standard turtle commands

	Command							influenced variables
	without arg.	with one argument x						
		with sustained effect			with local effect			
Group 1	L	L(x)	L+(x)	L∗(x)	Ll(x)	Ll+(x)	Ll∗(x)	ℓ
	D	D(x)	D+(x)	D∗(x)	Dl(x)	Dl+(x)	Dl∗(x)	d
	V	V(x)	V+(x)	V∗(x)	Vl(x)	Vl+(x)	Vl∗(x)	v
	N	N(x)	N+(x)	N∗(x)	Nl(x)	Nl+(x)	Nl∗(x)	n
	P	P(x)			Pl(x)			p
Group 2	F	F(x)	F+(x)	F∗(x)				P, q, m, g
	f	f(x)	f+(x)	f∗(x)				P, q
		@(x)						P, q
Group 3		RH(x)						L, U
		RL(x)						H, U
		RU(x)						H, L
	RV	RV(x)	RV+(x)	RV∗(x)				H, U
	RG							H, L, U
	+							H, L
	−							H, L
	$							L, U
Group 4	[							b, r
	]							*all*
	%							r

Some of the commands (especially F, f, +, −, \$, [,], %) were taken from [98], but the command language is extended and more systematically organized here.

Turtle commands of group 1: Assignments

The syntax and meanings of the group 1 commands L, D, V, N, P are rather systematic. All these commands are possible without argument (they set their corresponding state variable back to the default value in this case) or with one floating point number x as argument. The upper case leading symbol indicates the state variable which is changed. The symbols constituting a command, like L, l, +, (, must follow each other without blanks between them. The correct usage of upper- and lower case letters is to be ascertained. Table 2 summarizes the effects of the L-commands on the state variables ℓ and ℓ_L; the other commands of this group act in an analogous manner.

Table 2: Effects of L-commands

command	new value of ℓ	new value of ℓ_L
L	ℓ_g	ℓ_g
L(x)	x	x
L+(x)	$\ell + x$	$\ell + x$
L*(x)	$\ell \cdot x$	$\ell \cdot x$
Ll(x)	ℓ	x
Ll+(x)	ℓ	$\ell + x$
Ll*(x)	ℓ	$\ell \cdot x$

Turtle commands of group 2: Movement and unit construction

These are the most important commands, because they cause the turtle to move and to create new cylindrical units. The letter "f" stands for "forward". The f command does only move the turtle (i.e. the position P is changed), while the F command makes the turtle additionally construct an elementary unit along the straight line where it moves, thereby using the state variables ℓ_L, d_L, n_L, p_L to fix the extensions and attributes of the new unit. The new position P' is at the midpoint of the upper circle of the newly created cylinder after an F command, while the old position P marks the midpoint of its basal circle (Fig. 17). The direction of movement is in both cases, for f and F, the current head direction (H) of the turtle.

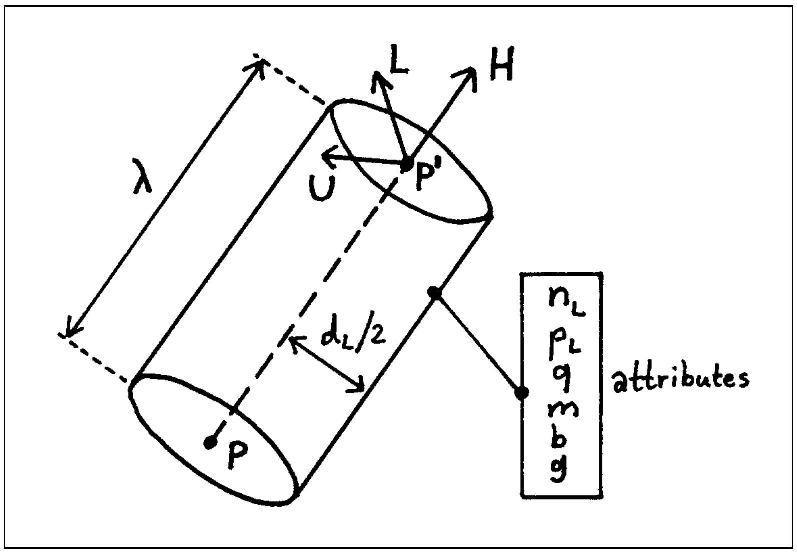


Fig. 17: Result of an F command

The several variants of the F command differ only in the length λ (of the constructed unit and of the movement) which is used. This length λ is

$$\begin{array}{rll} \ell_L & \text{for} & F, \\ x & \text{for} & F(x), \\ \ell_L + x & \text{for} & F{+}(x), \\ \ell_L \cdot x & \text{for} & F*(x). \end{array}$$

The variants of the f command work analogously, but without unit construction.

The command @(x) performs the same movement as $f*(x-1)$, or, equivalently, $f((x-1)\cdot\ell)$. Here, the argument x will normally be a number between 0 and 1. Thus, if the command @(x) comes directly after an F command, it will cause the turtle first to move back along the just created unit ($f(-\ell)$) and then again to move forward for a specified part x of the length ℓ of the unit (Fig. 18).

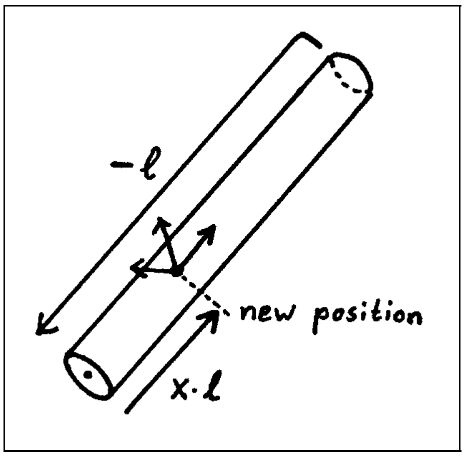


Fig. 18: Turtle movement after the @(x)-command

Besides changing the turtle position, the commands F, f and @ have also some influence on other state variables: If the used length is λ, the state variable q, which is meant to specify the *relative position* on the mother unit where a side branch emerges (but measured inversely to Fig. 18, i.e. identifying a shoot tip with 0 and the basis with 1), is set to

$$\begin{array}{rl} 0 & \text{by the } F \text{ command,} \\ q - \lambda/\ell & \text{by the } f \text{ command,} \\ 1 - x & \text{by the @}(x) \text{ command.} \end{array}$$

The mother unit m is actualized by the F command to be the newly created unit, and the generative distance counter g is incremented by 1.

Turtle commands of group 3: Rotation

With the exception of the abbreviated commands +, −, $, all rotation commands begin with an upper case *R*. The argument — if there is one required — is a floating-point number which is interpreted (except for the *RV* command) as an angle in degrees. Negative values and values exceeding 360 are allowed.

The commands $RH(x)$, $RL(x)$ and $RU(x)$ induce the turtle to rotate around the *H*-, *L*-, or *U*-axis, respectively, by x degrees. The position remains unchanged. After the command $RH(x)$, the direction vectors *L* and *U* have rotated by x degrees, while the *H* vector remains unchanged (Fig. 19). The situation after the $RL(x)$- and $RU(x)$-command is analogous. The relative situation of the vectors *H*, *L* and *U* to each other — forming an orthonormal system — remains unchanged during each rotation operation.

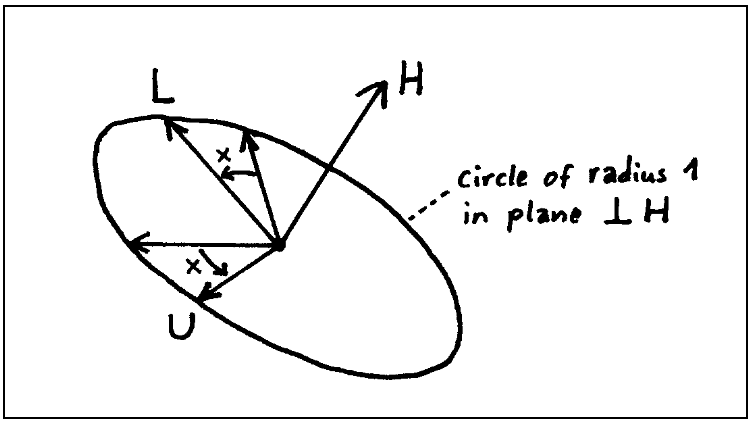


Fig. 19: Effect of the command $RH(x)$

Note that the moving direction of the turtle remains unchanged after a $RH(x)$-command, but is changed after $RL(x)$ and $RU(x)$.
The command + is equivalent to $RU(w_g)$, where w_g is the global variable mentioned above in the context of the \set-statement. Similarly, − stands for $RU(-w_g)$. The symbols + and − must — like all other turtle commands — be separated from the preceding and succeeding symbols by blanks to be interpreted correctly.

The $ command stands for $RH(\varphi)$, where φ is the rotation angle to be used for the *U* vector to minimize the angle between *U* and the vertical direction (i.e., the z axis of the global coordinate system). This is a “correction” of orientation which is often necessary after a sequence of rotations around different axes to ensure that the *U*-vector points as far as possible “upwards”.

RG is a command without argument that enforces an orientation strictly “downwards” (geotropic). After *RG*, the *H* vector is (0, 0, −1). (See Section

2.2 for the exact effect on the vectors L and U, which is normally not of interest after this command.)

The RV command makes use of a value s which is determined as

v_L	by RV without argument,
x	by $RV(x)$,
$v_L + x$	by $RV{+}(x)$,
$v_L \cdot x$	by $RV * (x)$.

This s quantifies a "vertical tendency" (downwards), or a "geotropism strength", in the following sense: RV adds the vector $(0,\ 0,\ -s)$ to H and enforces the new H to be the (normalized) sum by performing a rotation around the L axis (Fig. 20). The greater the value s, the greater is the "deviation" from the original H direction and the inclination of the new H direction downwards. Note that also negative values of s are possible, leading to an upward tendency.

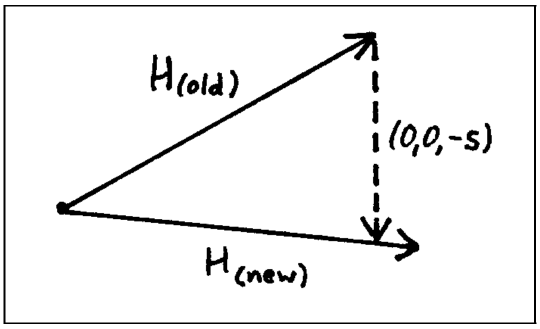


Fig. 20: Orientation change by the RV command

A sequence of alternating commands F and RV leads to a "bending down" of the generated branch. — The RV command is inactive if H is pointing directly upwards or downwards.

Turtle commands of group 4: Control of branching behaviour

By enclosing a sequence of commands in brackets [,], the turtle is induced to construct a *side branch.* This is done by storing the actual state of the turtle in a pushdown memory or *stack* once an open bracket "[" is read. When the corresponding closed bracket "]" is reached, the turtle stops the construction of the branch, forgets its current state and adopts the old state that 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 Fig. 21 (a)). Another interpretation — which leads to the same result — says 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 parallel (Fig. 21 (b)). This interpretation is certainly closer to the reality of plant growth, where meristems can divide itself and work in parallel, but due to the lack of properly parallel machines, the current implementation of GROGRA works according to the first, sequential interpretation.

Brackets can be nested arbitrarily often, but the usual syntax requirements for parentheses systems must be fulfilled, i.e. each open bracket must be balanced by a closed bracket on the correct level. Be careful that each "[" and each "]" is surrounded by blanks.

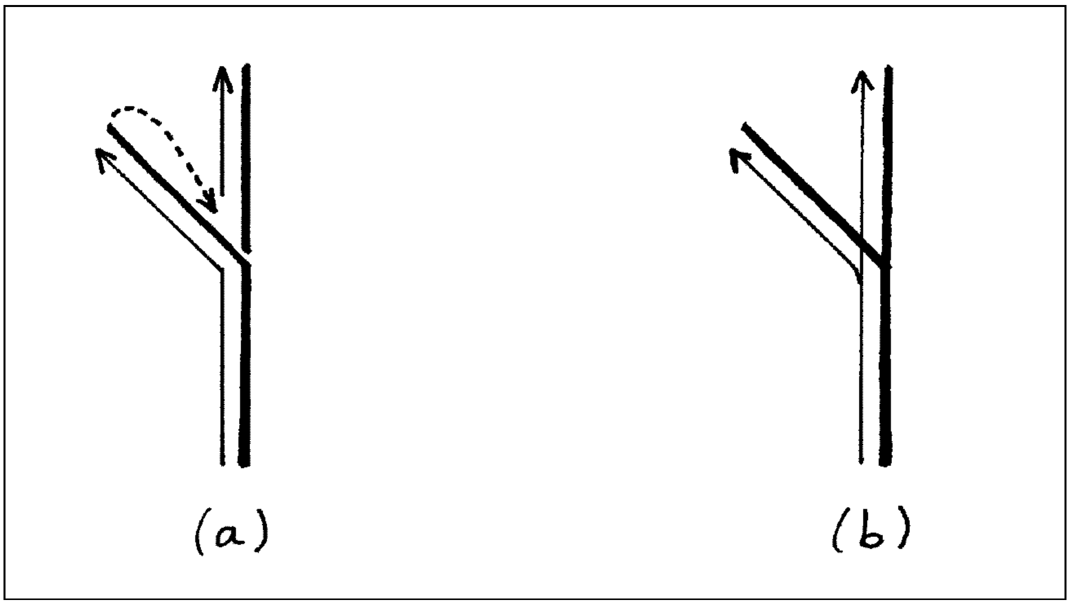


Fig. 21: Sequential (a) and parallel (b) interpretation of the turtle command sequence F [RU-45 F] F.

When the symbol "[" is passed, the branching order b is normally incremented by 1. The program takes care that in a construction like ... [[...]] ..., where no F commands occur between the [[pair, the order is also incremented only by 1, not by 2.

The symbol % works as a **cut operator**. It stops the construction of a branch *before* the appropriate closed bracket "]" is reached. This is often useful in developmental sequences when branch shedding is to be modelled. When we consider, as in this subsection, only momentary structures, the % command is superfluous.

Technically, the effect of the cut operator is performed by manipulation of the relevance counter r, which is part of the state (but which is not put on the stack when a bracket is reached). Each command is only performed when r is greater than 0. r is put to 0 by % , an r smaller or equal to 0 is decremented by [and incremented by].

Additional turtle commands: Register manipulation, method calls

Independently from the turtle state, there are (in the current version of GROGRA) 10 global "register variables" $r_0, r_1, \ldots, r_9$. They can carry additional information of a global character which is required during the execution of a growth grammar. Their current values can be obtained by declared register variables, see Section 4.3, p. 66. For the assignment and manipulation of their values, 6 turtle commands are available (let n be a decimal digit specifying the register number and x an arbitrary floating point number):

I$n = (x)$	assigns x to r_n,
I$n+ = (x)$	increments r_n by x,
I$n* = (x)$	multiplicates r_n by x.

The three commands

J$n = (x)$
J$n+ = (x)$
J$n* = (x)$

do essentially the same like the corresponding I commands, but at another moment in the generation-interpretation-cycle of GROGRA (see Section 4.2, p. 61). The I commands are executed — like all other turtle commands — during (geometrical) interpretation, the J commands during generative rule application.

The register handling is not very elegantly solved and will certainly be reorganized in later versions of GROGRA. However, the reference to global registers is often indispensable for realistic simulations (see the examples section), and the above declared commands are nothing but a first attempt to meet this need in the framework of turtle geometry and grammars.

Another feature which has even more *ad hoc* character but which is also necessary in some cases are **method calls**. Currently, there are three procedures (methods) implemented in GROGRA which perform certain calculations and manipulations on attributed geometrical structures which cannot be done by elementary turtle geometry commands (see Section 4.6, p. 74, for details). The execution of one of these methods is enforced when the turtle arrives at the command

$$\mathsf{M}n$$

where n in an integer (currently restricted to 1, 2 or 3) specifying the desired method.

Appendix 2 gives again a list of all 61 different turtle commands and their effects.

The execution of turtle commands to create a single attributed geometrical structure can be done directly by GROGRA when in the submenu evoked by "Generation of a new branching structure" the item "Turtle geometry without grammar <T>" is chosen. The turtle commands are then to be typed in by hand. However, this is a test modus working only for small strings because of the restriction to at most 127 characters of input. The created structure can be watched and analyzed as described in Section 3.2 for structures generated from grammars.

4.2 Generative rules and interpretative rules

Now we come to the second level of description, the description of how the attributed geometrical structures develop in time. This is done by systems of rules (L systems) which are to be applied to the strings describing the structures in turtle language. We refer to Section 2.1 for the general outline how the two rule classes, *generative rules* (rules of the first phase) and *interpretative rules* (rules of the second phase) work together. We call the generation of a new string by generative rules, the subsequent application of interpretative rules and the geometrical interpretation by the turtle the *generation-interpretation cycle* of GROGRA (Fig. 22).

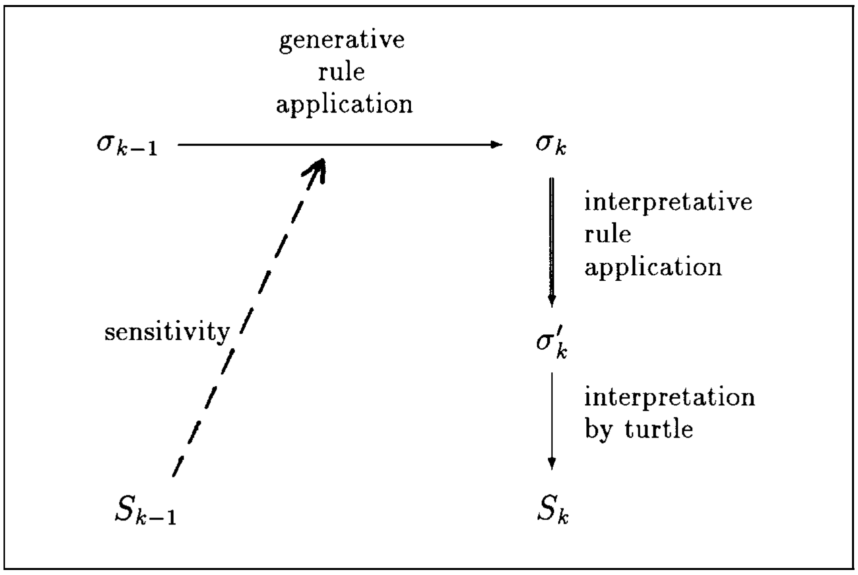


Fig. 22: The generation – interpretation cycle

All turtle commands are executed in the step "interpretation by turtle", with the exception of the J-commands, which are executed during the "generative rule application". Note that the execution of J-commands is always delayed by 1 timestep, because the J-commands appearing in σ_k can only be interpreted in the next cycle.

For the sake of rule application, we allow the strings to contain *additional symbols* (possibly carrying parameters) besides the turtle command symbols. These symbols may consist of one or several letters, digits or special characters, possibly followed by a parameter list (the single real-valued parameters separated by commas) enclosed in parentheses. *They are not allowed to begin with one of the characters* D, F, f, I, J, L, M, N, P, R, V, @, +, −, $, [,], % which are reserved for turtle command symbols. Furthermore, no comma or blank is allowed as part of a symbol. The most secure way to avoid conflicts with these restrictions is to use exclusively lower case letters and additionally to avoid the *f*. The length of symbols must not exceed 10 characters. Some examples for legal symbols are:

```
a(42, 17, -0.5)
root(1000, 0)
z14~(1E−6)
b
leaf_0
```

When combined to form strings, these symbols must be separated from each other by blanks. By the turtle, the additional symbols are treated like blanks, i.e. no action is performed.

A **generative rule** has in general the form

(*condition*) *l.h.s.* # *r.h.s.* ?(*probability*),

The parts (*condition*) and ?(*probability*) may be omitted. The rule ends with a comma. (This is not required for the last rule in a file.) The constituting parts, (*condition*), *l.h.s.*, #, *r.h.s.* and ?(*probability*), must be separated by blanks from each other. An **interpretative rule** differs in its syntax only by the usage of the "double double-cross" ## instead of #. For interpretative rules, sensitivity is excluded (see Section 4.7).

The condition, when one occurs, must be a conditional expression, see Section 4.4 (p. 72). The l.h.s. is always a single symbol, possibly equipped with *formal parameters*, see Section 4.3 (p. 68). *No actual real-valued parameters are allowed on the left-hand side of a rule.* Hence, of the above-mentioned example symbols, only "b" and "leaf_0" would be allowed as l.h.s. We will see in Section 4.4 how these restriction can be overcome by the usage of conditions. — Turtle commands, however, are allowed to form the l.h.s., as far as they have no actual parameters. Thus, it is e.g. permitted to formulate a rule

```
F # % ?0.5,
```

which "kills" each created unit in the next timestep with a probability of $\frac{1}{2}$. (However, the usage of turtle commands like [or] on the l.h.s. is not recommended because it may lead to syntactically incorrect strings as results.) The length of a l.h.s. must not exceed 70 characters.

The r.h.s. of a rule consists of an arbitrary (possibly empty) string, made up of symbols (turtle command symbols and additional symbols mixed together) which are now allowed to bear actual parameters (and also arithmetical expressions instead of parameters, see Section 4.3). A possible rule would be

```
b # a(42, 1, 0) [ RU45 L*0.5 F b ] c(0),
```

where no condition is given and the probability is automatically assumed to be 1. An example for a rule with an empty r.h.s. is

```
c(t) # ,
```

The total length of a r.h.s. is bounded to at most 600 characters (including all blanks and parentheses), and the length of its constituents (symbols with parameter lists) to 70 characters.

The application of a system of rules on a string (which is the start string in the beginning) is principally performed in parallel to each symbol of the string, replacing the symbol by the appropriate r.h.s., if there is a rule with this symbol as its l.h.s. (GROGRA, however, performs this task sequentially, passing the string from left to right.) If no applicable rule exists, the symbol *remains unchanged*. If several rules apply, and if no conditions and probabilities are involved, the order in which the rules are written down in the grammar file dictates which rule is applied: It is the first one.

The **probability** must be specified immediately following the ? sign. It can be given by an arithmetical expression (Section 4.4) which is to be enclosed in parentheses, or directly by a floating-point value between 0 and 1, for which the parentheses can then be omitted. When for all rules with the same l.h.s. the sum of the probabilities is 1, the choice between these rules is done randomly with the given probabilities. If the sum is smaller than 1, let us say: s, then the symbol will remain unchanged with probability $1 - s$.

If a rule is too long to fit in a single line of the grammar file, it may be split up by carriage-return-signs between its symbols, like the second rule in the example system base.lsy in Section 3.2.

4.3 Parametrization and variables

GROGRA 2.4 allows three fundamental types of numerical variables:

- *constants*, which just stand for single numerical values and which have to be defined in the directive part of a grammar file,
- *declared variables*, which have to be declared in the directive part (e.g., as random variables),
- *bounded variables*, also called *formal parameters*, which appear on the l.h.s. of some rule and which are replaced by numerical values from the actual string before the rule is applied.

All variable names (identifiers) are restricted in their length to at most 10 characters and can consist of arbitrary letters and other characters, excluding commas, blanks, parentheses and operator symbols like +, −, ∗, /, ^. They should not end with an underline character (_), and should not be identical to one of the strings atan, atg, exp, log, max, min, sqrt. Upper and lower case letters are distinguished. Because GROGRA can distinguish variable identifiers by their syntactical position in the rules from symbol names, it is allowed that some of them coincide with symbol names. Thus, a rule like

```
t(t) # t(t+1),
```

would be possible, where the "t" outside the parentheses is a symbol (e.g. standing for some terminal meristem) and the "t" inside the parentheses is a formal parameter, e.g. standing for time.

All variables may appear in arithmetical *expressions* (the "t+1" in the above rule being a simple example), see Section 4.4 for details.

The *evaluation* of a variable, i.e. its replacement by a numerical value, takes place when GROGRA evaluates the expression where it appears during the application of a generative or interpretative rule.

Constants

A constant definition has the form

\const *name value*,

where *value* is a floating-point decimal number in one of the usual notations allowed in the programming language C. Like all other directives beginning with a backslash character (\), a constant definition must precede all rules in the grammar file and must be finished by a comma.
The effect is that every appearance of *name* in a position in the rules where a numerical value is permitted is replaced by *value*. Note that constant names are not allowed to appear as part of the l.h.s. of a rule.

Declared variables

Variable declarations are part of the directive part, i.e. the first part of a growth grammar file, and can be mixed with constant definitions as well as with \set- and \ask-statements. Their general syntax is

\var *name specification parameterlist*,

where each part is separated from the next by a single blank. The *name* is the name of the variable to be declared, for the restrictions see above. The *specification* must be one of several possible keywords listed below. It specifies from where the variable gets its value when it appears in a rule which is to be applied. The *parameterlist* consists of one or several (but not more than 20) numerical values, separated by blanks. How much values are required and what their meaning is depends on the specification. For some specifications, the *parameterlist* is empty; the syntax is then simply

\var *name specification*,

Be careful not to forget the finishing comma. — The number of constants and declared variables together must not exceed 60.

Let us now discuss the possible *specifications*.

uniform The variable is declared as a uniformly distributed random variable. Its value is taken from a pseudorandomnumber generator each time it is evaluated. (If an already chosen value has to be used once more, the memory operator _ (underline) has to be attached to the variable name — see Section 4.4, p. 71.) The parameter list must contain two values, the *lower bound* and the *upper bound*. E.g., the declaration

```
\var x1 uniform -1.5 1.5,
```

specifies $x1$ as a uniformly distributed random variable between -1.5 and 1.5.

normal Declaration as a random variable with a normal distribution. There must be two parameters in the list, specifying the *mean value* and the *variance*.

distribution A variable declared with this specification assumes only non-negative integers as values. The choice of the integer is performed randomly, the values in the parameter list (up to 20 are allowed here) giving explicitely the probabilities for the numbers 0, 1, 2, ... E.g., the declaration

```
\var n distribution 0.1 0.7 0.2,
```

specifies n as a random variable with value 0 with probability 0.1, value 1 with probability 0.7, value 2 with probability 0.2.

user_request The value of a variable of this type is asked from the user at runtime at each occasion when the variable is to be evaluated. There is no parameter list.

table The value is taken explicitely from the parameter list: The $(n + 1)$-th value from the list is taken in the n-th generation - interpretation - cycle (the cycle counting beginning with $n = 0$). If more cycles than values in the list are executed, the last value from the list is repeated. E.g., the declaration

```
\var a table 5 6.5 7,
```

leads to $a = 5$ in cycle 0 (i.e., when the start word is transformed and the resulting string interpreted), $a = 6.5$ in cycle 1 and $a = 7$ in all further cycles.

array This specification requires that an extra file is prepared which must contain all values which the declared variable can possibly assume. The main part of the filename must be the same as for the corresponding .lsy- or .ssy-file, but the suffix must be .a*dd*, where *dd* is a two-digit integer between 00 and 99, referred to as the *file number*.
The parameter list has the form

$n\ a\ b_1\ b_2\ \ldots\ b_a$

with n representing the file number, a the *arity* of the array (an integer between 0 and 5), and b_j the upper bound for the j-th *argument*. Each time the variable appears in the context of a rule, it must be followed by a non-negative arguments, which are rounded to the nearest integer and used as indices for an entry in an a-dimensional number array given by the file. This entry, then, is taken as the value of the variable.
The file must contain $b_1 \cdot b_2 \cdot \ldots \cdot b_a$ floating-point numbers as entries in lexicographic order, starting with the entry corresponding to the index combination $(0, 0, \ldots, 0)$ and ending with the entry corresponding to $(b_1 - 1, b_2 - 1, \ldots, b_a - 1)$.
E.g., `\var p array 4 2 3 2,` in a file test.lsy specifies p to be described by the file test.a04 as a two-dimensional matrix with $3 \cdot 2 = 6$ entries. The values in the file are $p(0,\ 0)$ $p(0,\ 1)$ $p(1,\ 0)$ $p(1,\ 1)$ $p(2,\ 0)$ $p(2,\ 1)$ (in this order, separated by blanks or carriage returns), and p can only assume these 6 values. If, e.g., the expression $p(1.7,\ 0)$ is found in a string to be transformed, the value found for $p(2,\ 0)$ in the file is substituted for that expression. — If one of the upper bounds is violated (e.g., if $p(0,\ 2)$ appears), there will be an error message.

generation The variable takes as its value the number of the generation - interpretation - cycle which is just performed, the counting starting with 0. There is no parameter list.

register The declared variable is associated with the content of one of the global registers $r_0, \ldots, r_9$. The parameter list contains only one value, namely, the number of the register (which is restricted to $0, \ldots, 9$ at the time). Each time the variable is to be evaluated, the value is taken from the corresponding register.

index This specification defines the variable as an index variable for the repetition operator &, assuming the values 0, 1, 2, ... and incrementing itself with every new repetition. See Section 4.5, p. 73.

length The variable gets the value of the current steplength (ℓ) of the turtle. Because the turtle is not active in the generative step, this makes only sense in interpretative rules; cf. the example base.lsy in Section 3.2. — There is no parameter list in this case.

diameter The variable gets the value of the current turtle diameter d. Analogous to *length*.

n_value The turtle parameter n is taken. Analogous to *length*.

v_value The turtle parameter v is taken. Analogous to *length*.

color The turtle parameter p is taken. Analogous to *length*.

q_value The variable gets the value of the turtle state variable q, which stands for the relative position of a unit at the axis of the mother unit (base = 1, tip = 0). The same restrictions as for *length* hold.

xcoordinate This specification is allowed only in sensitive grammars (.ssy-files). It induces GROGRA to take as value of the variable the x-coordinate of the corresponding elementary unit tip in the just created structure (see Section 4.7 for details). There is no parameter list.

ycoordinate Analogous to *xcoordinate*.

zcoordinate Analogous to *xcoordinate*.

function A variable of this type refers to a special function which must be compiled as part of the GROGRA software before. The currently available functions are listed in Table 3 below. Some of these functions are allowed only in sensitive grammars (.ssy-files). The parameter list is n a, where n is the number of the function and a its arity (number of arguments, must lie between 0 and 5). Only numerical arguments are counted here, not a possible dependence from the generated structure (sensitivity). The arity may be 0. — When a variable declared as *function* appears in an expression to be evaluated, it must be followed (like an *array* variable) by a list of a parameters.

Table 3: List of currently implemented functions

Number of function	Sensitivity required	Arity	Return value (see below)
1	x	1	$f_1(m)$
2	x	1	$f_2(m)$
3	x	0	f_3
10		1	$f_{10}(x)$
11		5	$f_{11}(n,\ k,\ b,\ c,\ d)$

The return values are:

$f_1(m)$ = maximal opening angle for a sector emerging from the tip of the current unit upwards which contains no other units longer than m. (2-dimensional version.) See also Example 6.10, p. 157.

$f_2(m)$ = minimal distance from the tip of the current unit to base and tip of other units except the mother unit of the current unit and except all units shorter than m. See also Section 4.7.

f_3 = sum of all values of attribute n of the system of units emerging from the current unit (including this unit itself).

$f_{10}(x)$ = greatest integer smaller or equal to

$$\frac{x}{2} + 2.3 - \exp(0.022\ x + 0.06).$$

$$f_{11}(n,\ k,\ b,\ c,\ d) = \begin{cases} 0 & \text{if } d = 0 \text{ or } d = 1, \\ b \cdot n/d & \text{if } n < d \cdot (n+k),\ d \neq 0,\ 1, \\ (b+cd)k/(1-d) - cn & \text{if } n \geq d \cdot (n+k),\ d \neq 0,\ 1. \end{cases}$$

Bounded variables

Bounded variables are not declared. They are bound to the rule where they appear as part of the formal parameter list of the l.h.s., and they may only be used in this same rule (in its r.h.s., in its condition or in its probability expression). E.g., the rule

```
m(a0, t) # F(a0) [ + k(a0/2) ],
```

has the two bounded variables $a0$ and t, and only $a0$ is used on the r.h.s. The value of such a variable is determined in the moment when the l.h.s. *matches* with a symbol in the string to be transformed by rule application (either in the generative, or in the interpretative application step). E.g., if the above rule is applied to m(12, 4.2), the resulting assignments will be $a0 = 12$, $t = 4.2$ (and m(12, 4.2) will be transformed into F(12) [+ k(6)]). — Up to 40 bounded variables (formal parameters) are allowed per rule.

Registers

An extra role is played by the *register variables* r_0, r_1, …, r_9. These have no fixed names, to use them as variables in expressions requires the declaration of some variables with the specification register and with the corresponding register number as parameter (see above). The register contents are initially zero. They can be initialized with other values by the statement

\set **I**n *value*,

or, equivalently,

\register n *value*,

where n is the register number. An alternative is the user request at runtime, induced by

\ask **I**n *question text*,

(cf. the \ask-statement in Section 4.1). All these statements must appear in the directive part of a grammar file and are executed before the grammar begins to transform the start string. If a later changement of register values is required, the turtle commands **I**$n = (x)$, **I**$n+ = (x)$, **I**$n* = (x)$ or J$n = (x)$, J$n+ = (x)$, J$n* = (x)$ are to be used (see Section 4.1).

4.4 Expressions and conditions

The *general form of a growth grammar rule* (cf. Section 3.2, Definition 8) is

(*condition*) *l.h.s.* # *r.h.s.* ?(*probability_expression*),

for generative rules, and the same with ## instead of # for interpretative rules (see Section 4.2). (Be careful to leave blanks between the different parts of a rule.) The *left hand side* (l.h.s.) consists always of a single symbol, possibly followed by a list of formal parameters, separated from each other by commas and altogether enclosed in parentheses, like

leaf(x1, x2, t)

No constants and declared variables are allowed to appear in this list.
The *right hand side* (r.h.s.) consists of a string of such parametrized symbols. However, in the case of the r.h.s. there can be *arithmetical expressions* instead of single parameter names as arguments of the symbols. We speak simply of *expressions*. Hence, the general form of a constituent of the r.h.s. is

symbol(*expression_1*, …, *expression_k*)

if k is the arity of *symbol*. The r.h.s. consists of several of such constituents, separated by blanks. (The r.h.s. may also be empty.)

An expression may contain constants, declared variables and the formal parameters from the l.h.s., which are combined with each other by *arithmetical operations* or with the help of function symbols (see Section 3.2, Definition 2 for a formal definition of expression syntax). The usage of arithmetical operations and functions is very close to the conventions of the programming language C (see, e.g., [15]). The following extensions to the C syntax have to be noted:

- There is an additional binary operator ˆ symbolizing exponentiation, i.e. $a^b = a\hat{}b$. The ˆ binds closer than $*$ and $/$, and these operators bind closer than $+$ and $-$.
- In the case of functions with no arguments, the parentheses are omitted. (E.g., $a + 2 * \mathsf{func}$ instead of $a + 2 * \mathsf{func}()$ in C.) (Such a function must necessarily be declared as a variable of type function, see Section 4.3, p. 68.)
- For numerical values, GROGRA has no operators like &&, ||, &, |, !, =, % which can be used in C.

More explicitely, the following operators can be used:

- the binary arithmetical operators $+$, $-$, $*$, $/$, ˆ
- the unary operators $-$, _ ($-$ as prefix, _ as suffix operator; for the meaning of _ see below)
- the unary standard functions exp (exponential function), log (logarithm to the base e), sqrt (square root), atan (arcustangens with argument in radians), atg (arcustangens with argument in degrees).
- the binary standard functions min, max
- declared variables of type function or array with the number of arguments specified in the declaration (see Section 4.3).

Expressions can be nested; arbitrarily many parenthesis levels are allowed (the usage, however, being restricted by the machine stack). The length of an expression must be less than 100 characters in the current GROGRA version. Examples for arithmetical expressions are

$$x$$
$$42$$
$$(4 + x) * 1066.5/y1$$
$$\mathsf{atg}(45 + 3 * x\hat{}2)/\mathsf{min}(\mathsf{time} + \mathsf{log}(x),\ -p * q)$$
$$a0 + 17.5 + \mathsf{f1}(\mathsf{sigma},\ 12.5 * \mathsf{exp}(v))$$

(the f1 in the last line must be a declared function or array variable of arity 2).

The **memory operator** is symbolized by an underline character (_) immediately following a variable name. It is allowed only for declared variables and cannot be applied to more complicated expressions. Its usage is restricted to cases where the same variable name did already appear in the same rule, and its effect is that it forces the variable to take on its former value again. This is especially senseful in the case of random variables (type uniform, normal or distribution; see Section 4.3). E.g., if *vt* is declared by

```
\var vt uniform 0 100,
```

then in the r.h.s.

```
a(0.7 * vt) L(vt_) D(vt_/10) b(vt) c(vt_)
```

the values of *vt_* in L(vt_) and D(vt_/10) will be the same as for the first (randomly chosen) *vt* in a(0.7 * vt), while the argument of *b* will be a new random value between 0 and 100, and the argument of *c* will be identical with that of *b*.
The memory operator is allowed only for variable identifiers without arguments. However, a construction like the following is legal:

```
\var a array 1 2 10 10,
b(i, j) # D(a(i,j)) N(2.5*a_) F,
```

Here, the reading of $a(i,\ j)$ from the file with suffix .a01 takes place only when the argument of D is evaluated, and the found value is also substituted for a_ in the argument of N. (The same procedure is possible for variables of type function.)

Expressions which cannot be evaluated (like 1/0, sqrt(-1) or log(0)) induce an error message at runtime which gives an information about the expression causing the problem, e.g.

```
illegal sqrt in evalu!
```

(evalu is the name of the internal GROGRA subroutine which evaluates arithmetical expressions.) In such cases there will probably also be a further, subsequent error message

```
Evaluation of [ string ] failed. atomeval unfinished.
```

with the demand to accept this message by pressing the return key. Here, *string* is the expression where the evaluation problem occurred. Other possible error messages in the case of syntactically incorrect expressions are

```
evalu: plevel < 0
evalu: suspicious parenthesis level
```

indicating a false arrangement of parentheses, or

```
evalu: var. type wrong
evalu: illegal function syntax
evalu: array or function type misunderstood.
```

Declarations and usage of declared variables have to be checked again if one of these messages occurs. The error message

```
evalu: readfromarr failed
```

indicates that something with the file from which a variable of specification array gets its values (see Section 4.3) is wrong. Furthermore, the messages

```
evalu: not able to evaluate at all
```

and

```
expression string not canonical
```

indicate some error in the expression which cannot be further specified. Check the variable identifiers, the correct usage of separating blanks and commas and the distinction of upper case and lower case letters in such a case.

The **probability expression** of a rule — if one appears — has the same syntax as the arithmetical expressions appearing in the r.h.s. However, in many cases it will consist of a single floating point value between 0 and 1 (and the enclosing parentheses can then be omitted). An improper syntax of the probability expression will be indicated by the error message

```
Error in anwendfi / evalu (probability)
```

or

```
Error in abarbfi / evalu (probability).
```

(The first message refers to generative rule application, the second one to interpretative rule application.)

The **condition** consists of arithmetical expressions which are combined by *comparison operators*, namely,

>	(greater than)
<	(less than)
>=	(greater or equal)
<=	(less or equal)
= or ==	(equal)
!=	(unequal).

These comparisons can in turn be combined using *logical operators*:

&&	(and),
||	(or),
!	(not).

Parentheses inside a conditional expression are allowed. The whole condition has to be enclosed in parentheses.
Thus, a possible condition would be

```
(t >= 0 && ! x = 2 || !(x_^2 < t/2 ))
```

(! binds stronger than &&, and && stronger than ||). In the case of an incorrect syntax, there will appear error messages like

```
condevalu: Bad parenthesis level
condevalu: Bad || sign
condevalu: first part of || statement bad
condevalu: parentheses do not match
condevalu: evalu of left side failed
condevalu: no possibility to evaluate the expression
Error in anwendfi / condevalu
```

or

```
Error in abarbfi / condevalu.
```

The order of evaluation is always: First the condition (if one exists), then the probability expression (if one exists), then the r.h.s. of the rule from left to right.

4.5 The repetition operator

The repetition operator is symbolized by the & character in connection with a parentheses pair < > (less- and greater-character). Its usage is restricted to the r.h.s. of rules where it serves as a means to replicate certain parts of the string. The & character bears one argument (an explicit numerical value or an expression enclosed in parentheses ( )) which specifies how often the part between the subsequent < and > is to be repeated. E.g., the rule

```
a # &4 < b >,
```

yields, when applied to the symbol a, the string b b b b. (The repetition factor must follow the & character immediately, whereas the < and > character must be surrounded by blanks.) In the current GROGRA version, the repetition factor is restricted to numbers between 0 and 200. Non-integer values are rounded to the nearest integer. For the expression behind the & character, all possibilities for arithmetical expressions stand open (see Section 4.4). E.g., the following rule is allowed

```
a(t) # b &((t/2)^2) < a(0) F > c,
```

and leads, when e.g. applied to a(3), to the string

```
b a(0) F a(0) F c
```

because $(3/2)^2$ is rounded to 2.
Between the < > pair, it is possible to use an *index variable* which serves as a counter for the repetitions. This variable must be declared in the directive part of the grammar file as a variable of specification "index" (see Section 4.3). The counting begins with 0. E.g., the following combination

```
\var j index,
a # &4 < RH(j * 360 / 4) [ b ] >,
```

leads, when applied to a, to the string

```
RH(0) [ b ] RH(90) [ b ] RH(180) [ b ] RH(270) [ b ] .
```

An important restriction in the current GROGRA version is that it is forbidden to use nested repetitions like &(m) < &(n) < b > >. This restriction will probably be removed in one of the next GROGRA versions.

4.6 Method calls

Sometimes it is useful to perform certain calculations during the interpretation of a grammar which can badly be specified by the grammar itself. For this purpose, GROGRA offers the possibility to link certain procedures, so-called *methods*, with the main program. They can then be invoked during the interpretation of a turtle string by a simple command. Methods are identified by their number. The turtle command for the execution of a method is

$$\mathsf{M}n,$$

with n being an integer, the method identifier.

The currently available methods are

method 1: a procedure for assimilate distribution in a simple model plant (see Example 6.7, p. 151), utilizing the registers 1 to 4,

method 2: a procedure which re-calculates the diameters of all elementary units of the actual attributed geometrical structure which is in construction when the method is called. The calculation of diameters is done by assigning a default diameter (depending on length) to all terminal shoots and by assuming a conservation of sectional area in all branching nodes ("DA VINCI's law", see [78]). The method works only in RAM mode (see Section 5.4).

method 3: a procedure which eliminates all elementary units with accumulated n-value equal to zero in the currently constructed structure. The accumulation of n-values (standing for needle surfaces in the applications) is done recursively by adding the n-value of each unit to all topological predecessors of this unit where the q-value is ≤ 0.5, and half of the n-value to the accumulated value of the unit itself (unit-central accumulation). The method works only in RAM mode (see Section 5.4).

4.7 Sensitivity

Sensitive growth grammars are used when the geometrical arrangement or the attributes of the last created structure have to be taken into account during the generative rule application leading to the next structure (cf. Section 2.1). GROGRA enables this by the usage of a protocol file (named protok.oll) which saves the internal addresses of the elementary units of the last created structure in the order of the associated turtle commands and thereby establishes a link between the string to which the generative rules are applied and the last structure. (Interpretative rules are not allowed to make use of sensitivity.) Because the writing and reading of this auxiliary file costs extra time, GROGRA handles sensitivity as an optional feature which must deliberately be chosen by writing the grammar into a file with suffix .ssy instead of .lsy (as for ordinary, non-sensitive growth grammars) and by selecting the submenu item

sensitive growth grammars <S>

in the "Generation of a new branching structure"-submenu. Non-sensitive grammars can also be applied under this option, but this costs unnecessary extra time.
Sensitivity is introduced in growth grammars in two possible ways (which can be combined):

- by the use of declared variables of the types xcoordinate, ycoordinate or zcoordinate, which assume the value of the respective coordinate of the elementary unit endpoint in the last structure which is associated to the symbol at work,

- by the use of sensitive functions, which must be declared as variables of type function, namely, the functions 1, 2 and 3 in the current GROGRA version (see Section 4.3, p. 68). Sensitive functions are able to refer to the associated elementary unit *and* to the whole last structure as its global context.

The elementary unit in the last created structure to which a symbol is associated is always the unit which was created by the last preceding F command. If there is no such command, it is not defined and the application of a sensitive rule will lead to a program breakdown.

If an interpretative rule was used to create the last structure, only the unit created by the last F command in the r.h.s. of the interpretative rule can be referred to by sensitive rules. E.g., in the sensitive two-phase system

```
\var z zcoordinate,
a # b c a,
(z > 50) c # [ RU60 P14 F10 ],
b ## P4 F5 P2 F10,
```

the variable z will always contain the height of the top end of the longer, colour 2 (green) subsection of the axis built by b, and the branching determined by the second, sensitive rule will depend on this height.

The variable types xcoordinate, ycoordinate and zcoordinate have to be carefully distinguished from the types length, diameter, n_value etc. (see Section 4.3). A variable of coordinate-type refers to the unit associated to the actual symbol in the *last created* structure and can be used only in generative rules (in sensitive grammars). A variable of one of the types length, diameter, n_value, v_value, color or q_value, on the other hand, refers to the *actual* turtle state and thereby to the structure *which is just to be created*, and can be used only in interpretative rules (in nonsensitive or sensitive grammars).

An example for a sensitive function is function number 2, which calculates the distance to the nearest unit which is longer than the value given as argument (cf. Section 4.3, p. 68). The following system

```
\var  f  function 2 1,
   ⋮
(f(10) > 50)  a # [ RU30 F a ] RU30 F a,
   ⋮
```

determines the shoot tip corresponding to a to branch and grow only if no shoot longer than 10 length units comes with its base or end nearer than 50 length units. (The shoot itself and its mother shoot are excluded from the distance calculation.) If the restriction to shoots longer than 10 length units is not needed, it is possible to declare

```
\var f function 2 0,
```

and then to write

```
(f > 50) a # [ RU30 F a ] RU30 F a,
```

causing f to investigate the distances of *all* shoots except the shoot corresponding to a itself and its mother shoot. See Chapter 6 for further examples of sensitive grammars.

Chapter 5

Reference guide to the GROGRA software

5.1 The internal representation of a branching structure

Each attributed geometrical structure (cf. Section 4.1) which is generated according to a growth grammar with the syntax described in the last chapter, is internally represented by GROGRA as a linked list in the main memory (RAM mode) or as a sequentially organized external data file (HD mode, see Section 5.4). The two modes differ only in organizational aspects, and we will refer to the RAM mode here, which is the default when GROGRA is started. See Section 5.4 for a list of the restrictions inherent to HD mode.

The list elements are data sets (records), each corresponding to an *elementary unit* (shoot) of the geometrical structure. The data enclosed in such a record stand in close correspondence to the turtle state variables (cf. Section 4.1) which were actual at the moment when the corresponding elementary unit was created by the turtle. We call them *elementary unit variables*. They contain all geometrical informations and attributes which are available in GROGRA about this specific elementary unit. We refer to them partially with the same names as for the turtle state variables:

$P = (p_x,\ p_y,\ p_z)$, a vector representing the position of the *basis* of the elementary unit (i.e. the midpoint of the basal circle, see Fig. 23),

$Q = (q_x,\ q_y,\ q_z)$, a vector representing the position of the *tip* of the unit (i.e. the midpoint of the circle at its top),

$H = (h_x,\ h_y,\ h_z)$, a vector of length 1 in the direction of the unit's axis,

$L = (l_x,\ l_y,\ l_z)$, a vector of length 1, orthogonal to H, resembling the L direction of the turtle in the moment of creation of the unit,

$U = (u_x,\ u_y,\ u_z)$, a vector of length 1, orthogonal to H and L, resembling the U direction of the turtle in the moment of creation of the unit,

i, an integer serving as *identifier* of the unit,

s, a reference (pointer) to the *successor unit* in the internal list,

m, a reference to the *mother unit*,

c, a reference to a unit prolonging the same axis (see Section 5.5, Transformation for HYDRA, p. 119),

ℓ, a real number representing the *length* of the elementary unit, i.e. the distance between P and Q,

d_b, a real number representing the *diameter* of the unit at its *basis*,

d_t, a real number representing the *diameter* of the unit at its *top* end (normally identical with d_b),

n, a real number representing some non-geometrical parameter of the unit, currently used for *needle surface*,

p, an integer representing the *colour* of the unit (see Appendix 1),

q, a real number between 0 and 1 representing the *relative position* of the unit's basis along the mother unit ($0 =$ unit emerges at the top of the mother unit, $1 =$ unit emerges at its bottom),

b, an integer representing the *branching order* of the unit,

g, an integer, called the *generative distance*, indicating normally the number of units between the current unit and some mother-less "root unit", but also used for other purposes (see Section 5.5),

x_b, y_b, x_t, y_t, four auxiliary variables (real numbers) used for different purposes during the graphical display and in transformation and analysis procedures (especially during the transformation for HYDRA, p. 119: x_b for the LSC value, y_b for the ratio length/LSC, x_t for the criterion distance, y_t for the relative deviation $(\ell - x_t)/x_t$; furthermore in the "lengths and angles" analysis: x_b for the cluster index, y_b for the number of daughter unit clusters.)

The elementary unit variables s, c, x_b, y_b, x_t and y_t are dedicated to internal purposes and are normally not accessible to the user. The rest of the variables can be controlled for each shoot by calling the menu item "List the actual structure" in the main menu of GROGRA. For each structure and for each unit of the structure (displayed in the order given by the linked list established by the s pointers) there will be a screen display (which can also be printed or written into a file) giving the elementary unit variables for one unit in the following form:

Shoot number i, emerging from shoot number (i of m):

length:	ℓ
diameter at bottom:	d_b
diameter at top:	d_t
leaf parameter:	n
color attribute:	p
order:	b
generation number:	g
q value:	q

position of beginning of shoot:

x coordinate:	p_x
y coordinate:	p_y
z coordinate:	p_z

position of end of shoot:

x coordinate:	q_x
y coordinate:	q_y
z coordinate:	q_z

sh:	(h_x, h_y, h_z)
sl:	(ℓ_x, ℓ_y, ℓ_z)
su:	(u_x, u_y, u_z)

The reference m to the mother unit, being essentially a pointer in the programming language C, is not displayed directly, but represented by the identifier i of the mother unit. There will be the display -2 if no mother unit exists. — All floating point values will be given with an accuracy of 6 digits behind the decimal point.

If a growth grammar was applied, there will normally be a complete *sequence of structures* present in the memory. They are numbered, beginning with 1, and structure number k is indicated in the "list" mode in a line

Structure number k:

which precedes all elementary units of this structure. By the second item of the submenu entered when "List the actual structure" was chosen, it is possible to select a specific structure number and shoot number for display.

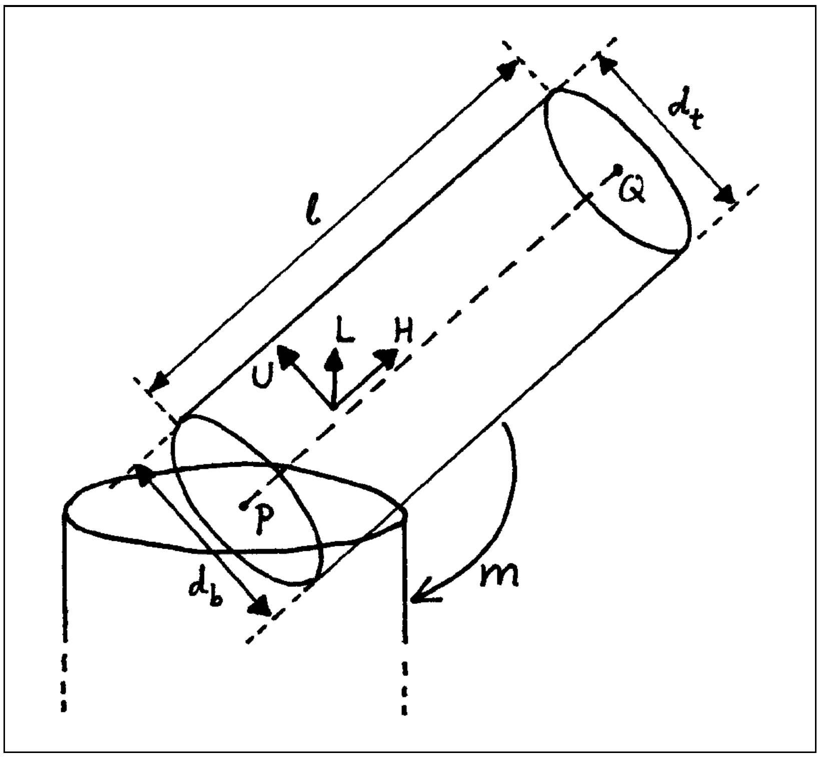


Fig. 23: An elementary unit with its geometrical variables and mother unit

5.2 Menu items

We refer to each menu item in the form

item text [submenu where the item appears].

Some menu items appear only when specific transformations have been done before. This will be noted in the comments to the items. Each menu item can be activated by the mouse or with some key on the keyboard (cf. Section 3.2).

a Generation of a new branching structure [Main menu]

This is normally the first menu item to be invoked when a new structure is to be created. If there is already a structure present in the memory which is not yet saved on disk, after activation of this item there will be a small submenu

Attention: There is already a structure.
Overwrite <U>
Back to the Main Menu <Z>

Choose "Overwrite" if really a new structure shall be made.

give all segments explicitely $< E >$ [Generation of a new branching structure]

This item is included for test purposes only, and its usage is not recommended to the normal user of GROGRA. It enables the user to specify an attributed geometrical structure "by hand", i.e. by the online input of all necessary data for each elementary unit. This specification will be a rather time-consuming labour even for small structures.
When this item is invoked, the first input to make is a coordinate triple for the point where the new geometrical structure shall emerge, i.e. the basal point P of the first shoot. This is required by the text

```
Please specify the position of the reference point.
x coordinate?
```

When all three coordinates have been specified (e.g., by typing "0"), the next output will be

```
Mother shoot number 0
        How many daughters shall this shoot have?
```

The newly created structure will have an invisible basic elementary unit with length 1, diameter 0 and order -1, extending from $(r_x,\ r_y,\ r_z - 1)$ to $(r_x,\ r_y,\ r_z)$, if $(r_x,\ r_y,\ r_z)$ is the specified reference point. This basic unit, which serves for reference purposes, is referred to as "Mother shoot number 0". With the specification of the number of its daughters, it is specified how much elementary units will emerge from the basic reference point of the structure. (Type "0" here if you want to leave this input procedure quickly.)
After an integer > 0 is typed in, GROGRA demands the input data for each daughter shoot, one after the other (i.e. the elementary unit tree is specified in breadth-first order). The positions of the new units are to be specified by angles, lengths and relative positions of branching points relatively to the mother unit, i.e. no absolute coordinates are to be typed in besides the very first, basal triple. The first such relative specification is asked for by the text

```
Which angle (in degrees) shall the shoot have with
respect to the mother shoot axis?
```

A branching angle is required here (angle α in Fig. 24). The next question is

```
Which angle (in degrees) shall the shoot have in the
plane orthogonal to the mother shoot?
```

This asks for the spatial orientation (azimuth), i.e. angle β in Fig. 24. $\beta = 0$ means a direction pointing as steeply as possible "upwards" or, in the case of a mother axis parallel to the z axis, the direction of the x axis.

Next, the relative position q (see Fig. 24), the length ℓ, the bottom and top diameter d_b and d_t, and the "leaf value" n are to be specified. (Only the top diameter d_t is taken into account in the graphical display in the current GROGRA version.) The branching order b is not specified, it is constantly 0 for all branches except the fictitious basal reference branch with order -1. The colour is always "light green".

By the specification of the number of daughter shoots, the user can control how many units he wants to specify further, and thereby how large the structure shall be.

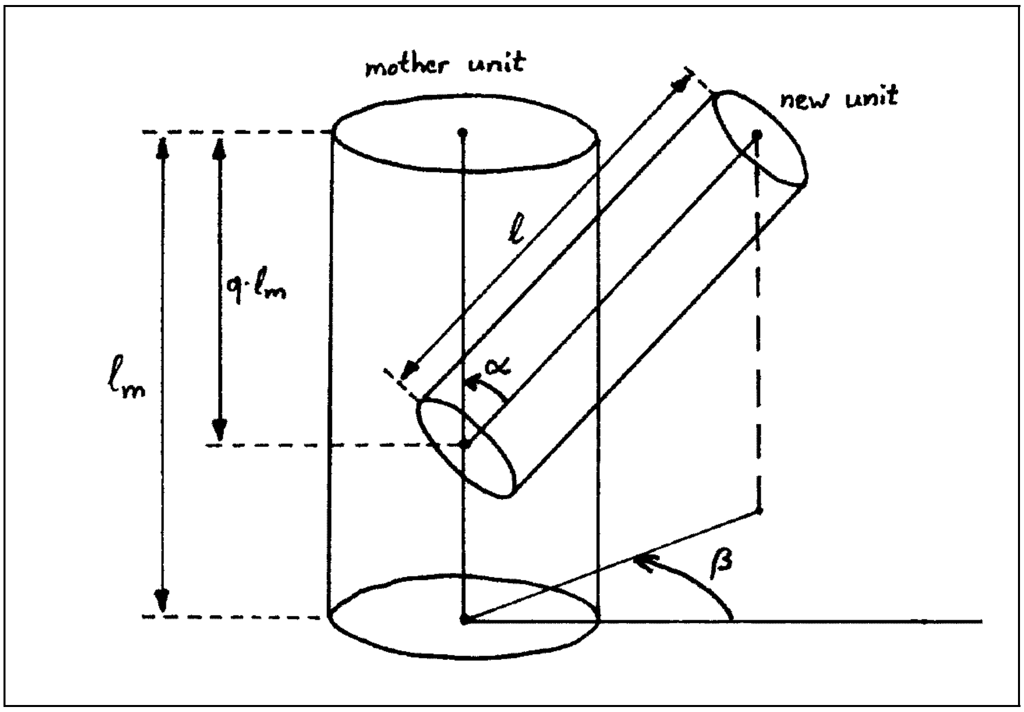


Fig. 24: Relative specification of an elementary unit (shoot)

Turtle geometry without grammar <T> [Generation of a new branching structure]

This item is also mainly intended for test purposes. It enables the user to give in a string which is then directly treated as a turtle command sequence. No interpretative or generative rules come into play here. All turtle commands are allowed, but with numerical arguments only, not with arithmetical expressions as arguments. Additional symbols are also allowed, but they make no sense here, as they are treated like blanks. The default values for the turtle state are the same as those given in Section 4.1. There is no possibility to alter them by a set- or

ask-command.
The input of the turtle command string is required by

Please give the string to be executed:

The string may extend into the second line (don't use the return key, just continue writing), but its length is restricted to 127 characters.

non-sensitive growth grammar <N> [Generation of a new branching structure]

This is the standard item for the creation of a structure from a growth grammar, when no sensitivity is required. To make use of it, it is necessary that at least one grammar file with the suffix .lsy — created with some text editor, and containing a non-sensitive growth grammar with the syntax described in the preceding chapter — exists in the subdirectory where GROGRA is executed.
When the menu item is activated, GROGRA announces

$*\,*\,*$ Application of a parametric L-system $*\,*\,*$
The following L-systems are actually available:

and gives then a list of the available .lsy-files. If there are too many files for a single screen display, the continuation of the list must be allowed by a keypress. Afterwards, GROGRA asks

Name of the L-system file (without extension):

E.g., if the execution of epi1.lsy is demanded, one should type in epi1 and then the return key. The correct reading of the grammar file (together with a first syntactical check) will be confirmed by

L-system successfully read.

In the next step, the start word must be typed in. For all examples in this manual, this is the $*$ (star character), but it is possible to give in any string here. If, however, the start string consists only of characters for which no rule of the loaded grammar is applicable, GROGRA will not send any error messages, but just produce the empty structure (seen as a black screen in the graphical display).

After reading the start word, the number of steps to be executed is asked for. This is at the same time the number of structures which can be created from the grammar and from the start word, forming a developmental sequence (see Section 2.1). If the number exceeds a certain limit (which depends on the machine and is 49 for the PC version at the time), the number is refused and the user is asked again. If the number is accepted, there will appear the question

Execution only of specified developmental steps (give <s>
and <return>) or of all steps (any other input):

The standard input would be just <return>, which induces GROGRA to execute all generation - interpretation - cycles completely. By typing "s", some interpretative parts can be deliberately omitted by the user. E.g., if the number of steps is 5 and only the steps 2 and 5 are specified, the execution scheme will be that of Fig. 25.

$$\begin{tikzcd}
\alpha \arrow[r] & \sigma_1 \arrow[r] & \sigma_2 \arrow[r] \arrow[d, Rightarrow] & \sigma_3 \arrow[r] & \sigma_4 \arrow[r] & \sigma_5 \arrow[d, Rightarrow] \\
 & & \sigma_2' \arrow[d] & & & \sigma_5' \arrow[d] \\
 & & S_2 & & & S_5
\end{tikzcd}$$

Fig. 25: Restricted interpretation of a non-sensitive growth grammar

The specification of the numbers of the structures to be created is done after s<return> is typed in. Each number is to be written in a single line, and the whole list has to be finished by an empty line.

GROGRA will then perform the prescribed generation - interpretation - cycles, indicated by the comment "First step of 5: Structure is generated." etc. Eventually, ask-statements or user-defined variables cause interruptions where some values have to be given in. The text "Structure is generated" appears only for those steps for which the interpretation is in fact done.
If the cycles are executed correctly, the message

```
Prescribed number of steps is done.
Execution of this L-system finished.
```

will appear. If a memory shortcut arises, a warning will appear before (see Section 3.2), but the structure will nevertheless be generated, eventually with missing units. (Choose the HD mode (see Section 5.4) or another machine in such a case.)

During the execution of the grammar, the auxiliary files lstri.str and lhilf.str will be created on the disk. Both contain afterwards the turtle command string corresponding to the last developmental step. If a grammar does not work how it should, it can be useful to have a look at these files.

The sequence of structures is now present in the memory, and there are the possibilities to save, show, list, analyze or transform it.

sensitive growth grammar <S> [Generation of a new branching structure]

The procedure after activation of this menu item is essentially the same as for non-sensitive growth grammars. Instead of .lsy-files, files with the extension .ssy are listed here. The option to execute only specified steps is not possible for sensitive grammars, because each generative step requires the presence of the structure created from the string to be transformed (cf. Fig. 12, Section 2.1). Additional to lstri.str and lhilf.str, a file protok.oll is created on disk, containing unit addresses (normally of no interest for the user).

b Read a structure from a file [Main menu]

Before this item is activated, there should be at least one file with suffix .dta, .dtb, .dtd, .pbg, .sbg or .map, containing encoded informations specifying attributed geometrical structures, present in the subdirectory where GROGRA is executed. The different data formats are explained in detail in Section 5.5.

Standard data (format .dta) <A> [Read a structure from a file]

The dta format is the only one which allows it to store a complete developmental sequence of structures in one and the same file. Hence, a sequence of structures is read when this menu item is invoked. (The sequence can, however, also consist of only one structure.) The dta format being the standard format used by GROGRA for saving structures in files, it contains all necessary informations to reconstruct a sequence of attributed geometrical structures (see Section 5.5, p. 110, for an explicit description of the data format). However, the reading process can take considerable time for larger structures, because the re-installation of the mother unit linkages requires extensive searching.

After activation of the menu item, a list of the available files with suffix dta (in the subdirectory where GROGRA is executed) is displayed, as in the case of lsy- or ssy-files for grammar execution. One of the filenames from the list (without the suffix) has to be typed in. During the reading process, there will be a message

```
Reading... please wait.
```

If the reading fails or the file cannot be opened, the program will be stopped with the message

```
There is some trouble with the file.
The program halted.
Press any key.
```

Otherwise, the message

```
reading process finished.
Press <return> to continue.
```

will indicate that the read structure is now present in the memory and can be worked with.
During the reading of a dta file, there is no rigid syntactical check whether the data in the file are correctly arranged. Thus, it can happen that after an apparently successful reading process the structure is either erroneous or empty. But this will normally not occur if the dta file is not manipulated by hand but written directly by GROGRA itself (see the menu item "Save in standard format (.dta)").

Autocad exchange data (format .dtb) <**B**> [Read a structure from a file]

The file specification and reading is performed in the same manner as for dta data above. However, each dtb-file contains information about only one developmental step, and moreover, the information carried in a dtb-file is far more restricted and does not allow the faithful reconstruction of the structure. Only the elementary unit variables P (basis position), Q (tip position), d_b (basis diameter), and d_t (top diameter) are read; H (direction) and ℓ (length) are re-calculated from these data. The colour is constantly set to "white" for all units, the leaf parameter n to 0. There is no calculation of branching order or generative distance, the corresponding variables being set 0, as are q, L and U. The topological information (mother unit linkages) is also lost.

data from HYDRA <**C**> [Read a structure from a file]

This menu item is devoted to the reading of structure descriptions in the format used by the discretization and water transport simulation programs DISC and HYDRA (see [41]). Contrary to the other reading items, no list of the available files is given after activation of this menu item. This is due to the fact that several distinct suffixes of filenames are allowed here, namely, .pbg, .sbg, .bag and .map (see Section 5.5, p. 119, for the meaning of pbg- and sbg-files and for detailed informations about the interface to HYDRA). The filename has therefore to be specified *including* the suffix.

As HYDRA refers to trees, the *tree species* is to be specified by a number before the reading process starts. (GROGRA displays a list of the possible tree species with their respective identification numbers.) Afterwards, like in the other reading items, there will be the message

```
Reading... please wait.
```

Additionally, the unit numbers will be shown in the form dnr = n, forming a long column of numbers on the screen. As the result of the reading, there will be two structures present in the memory, which are identical: a standard structure (more precisely, a sequence of structures consisting of one structure only, bearing the number 1) and a pbg-structure (see Section 5.5, p. 119. Note that the reading of an sbg- (or bag-) file does also yield only a pbg-structure, never an sbg-structure ("last modified structure") in the sense of GROGRA!) The input from the file is finished when the message

Reading process finished, press <return>.

appears.

The information about the elementary units available from the HYDRA formats is restricted and does not allow a faithful reconstruction of a structure. The following variables are read from the file for each unit: m (the mother unit identifier, which is reconstructed from a number found in the file), ℓ (the length), d_b (basis diameter), d_t (top diameter, always equal to d_b), n (needle surface). There is no angle information available from the file. The branching angle for all side branches (units not prolonging the axis of the mother unit, an information which can be taken from the file) is assumed to be 50°, the azimuth is taken randomly for each elementary unit. From these angles, from the length and from the already constructed mother unit the start and end position P and Q of the unit as well as the directions H, L and U will be calculated (cf. the item "give all segments explicitely", p. 83, in the submenu "Generation of a new branching structure"). The colour p is always "white", the q-value 0 (this being indeed the correct value in a structure having side branches only at the top ends of the units), the generative distance 0 and the branching order b determined recursively: The basic unit (with no mother unit) has order 0, and a daughter unit of a unit having order b has order $b+1$, if it is a side branch, and order b, if it prolongs the axis of the mother unit. The auxiliary variables x_b, …, y_t (including the LSC) are not read from the file, but have to be recalculated.

descriptive data (format .dtd) <**D**> [Read a structure from a file]

The descriptive data format dtd (see Section 5.5, p. 112) is meant for singular geometrical structures obtained from field measurements. The reading procedure is started in a similar manner as for dta- or dtb-files. After the specification of the filename, GROGRA will ask

Are buds to be included? (y/n):

If "n" (or an arbitrary other letter different from "y" or "j") is typed in here, lines in the dtd-file which are marked by the letter "K" (and are meant to specify buds) are not translated into elementary units like the other data lines. Otherwise, all lines will be taken into account.

GROGRA will also ask for a "default shoot diameter". The number which is typed in as an answer here is taken for the diameter variables d_b and d_t for all those elementary units for which no diameter is specified in the dtd-file (the diameter specification is optional in the dtd format, see Section 5.5). The value should normally have an order of magnitude of about 1.

The execution and completion of the reading process is indicated by

> Reading ... please wait.

and

> Reading process finished, press <return>.

As in the case of explicit construction of branching structures by hand, an invisible reference shoot is constructed at the basis of the structure, having order -1 and length 0 here. Positions and directions of the other shoots are calculated from lengths, relative positions and angles which are specified in the dtd-file. The branching order of a unit is calculated recursively: A daughter unit of a unit having order b has order $b+1$, if an azimuth specification was made in the dtd-file or if the branching angle relative to the mother unit is greater than 0, otherwise it is treated as an axis-prolonging unit having the same order b as the mother unit. This automatic order calculation is overridden if an explicit order specification (by an O or V statement) was made for the unit in the dtd-file. The generative distance is calculated automatically as $g+1$, if that of the mother unit is g, but is overwritten by G- or J-statements in the dtd-file (see Section 5.5). The colour is light green throughout the structure. The leaf parameter n is assumed to be 0 if no N-specification is read from the dtd-file. Positions, directions, topological linkages and q-values are calculated for each unit, and the structure obtained from a dtd-file will therefore be quite thoroughly specified and thus be comparable to artificial structures from growth grammars.
It is to be remarked that a structure created from a dtd-file will not necessarily lie in a suitable position for graphical display in "side view". The option "arbitrary direction of view" should be chosen then.

c **List the actual structure** [Main menu]

This item delivers explicit control informations about elementary unit variables. When the item is activated, there will first be a submenu entitled "Output device" listing the devices "Screen <B>", "Printer <D>" and "File <F>". If File is chosen, a file name has to be specified later. If a pbg-structure (or a pbg- and an sbg-structure) is present in the memory (see Section 5.5, p. 119), there will appear a small submenu entitled "What shall be given?" and containing the items "The unmodified structure <A>" (meaning the original sequence of structures) and "The first modified structure <B>" (referring to the pbg-structure) — and, eventually, also "The last modified structure <C>" (referring to the sbg-structure). The next submenu is entitled "Output of which structures / shoots" and has the two items

> all <A>
> beginning from a specified structure / shoot <S>.

If "all" is chosen, the order of display will be

Structure number 1
  Elementary unit number 1
  Elementary unit number 2
   ⋮
Structure number 2
  Elementary unit number 1
  Elementary unit number 2
   ⋮
⋮

In the other case, the display will begin with the structure and unit number specified by the user and then proceed in the same order. The display of elementary unit variables itself is explained in Section 5.1 (p. 79 ff.). It is continued by the return key and can be stopped by typing "q" and return.
If a structure has no elementary units, this will be indicated by

Structure number n: empty.

However, it can happen that no display at all is shown, and the main menu is immediately there again. In that case, all structures are empty.

d Store or transform the actual structure [Main menu]

Here, the complementary process to reading from file, i.e. saving, is accessible together with specific transformation procedures. Before this menu item is invoked, there should be a structure present in the memory, either created from a grammar or read from a file. A submenu entitled "Which kind of saving or transformation shall be carried out?" will appear immediately after activation of the item.

Save in standard format (.dta) <**A**> [Store or transform the actual structure]

This saving option stores the whole developmental sequence of structures which is currently present in the memory in one file, using dta-format (see Section 5.5, p. 110). The filename is to be specified (without the suffix .dta). It should respect the obligations for filenames in the DOS (resp., UNIX) operating system.
When the writing process has come to an end, there will appear the message

Saving finished, press <return> to continue.

The dta-file with all informations about the structure will now be present in the subdirectory where GROGRA is executed. The saved structure will remain to be present in the memory such that further display, analysis, transformation or saving is possible.

Save in .dta format, only one developmental step <**E**> [Store or transform the actual structure]

After activation of this item, GROGRA asks for a number between 1 and the number of structures forming the actually present developmental sequence. If n is typed in, only the n-th structure will be saved in dta-format.

Save in format for external exchange (.dtb) <**B**> [Save or transform the actual structure]

The dtb-format is mainly used for communication with the AUTOCAD system. When the menu item is activated, the number of the structure to be saved is asked for, as under the preceding item. Afterwards, a filename (without suffix) has to be specified.

Transform and save for HYDRA <**C**> [Store or transform the actual structure]

By this menu item, a more complex transformation and saving procedure is initiated which is explained in more details in Section 5.5, p. 119 ff. First, the number of the structure (as part of a developmental sequence) to be transformed has to be specified, like under the two preceding items. When this is done, GROGRA will ask

```
Screen protocol (with delays) required? (n=no):
```

If another input than "n" is given, some messages about extremal situations, transformation performance and specific parameters will be displayed on the screen during the subsequent transformation process, causing some delay of 1 or 2 seconds for each message to make it readable. As this can cost a considerable amount of time when a large structure is transformed, there is the possibility to switch off this automatic screen protocol by typing "n". However, all informations will be written into a special protocol file named standard.gpr in every case.

Subsequently, GROGRA will copy and modify the specified structure (see Section 5.5, p. 119), indicating the transformation steps more or less extensively on the screen (and also in the protocol file which can be studied later). This transformation closes with the screen display

```
STORING THE ACTUAL STRUCTURE USING DATA FORMAT C
The data are written into a file with the extension
                    .pbg.
name of the file (without the prescribed extension .pbg):
```

Here, a filename without suffix has to be specified. The file created in this writing step contains the informations describing a "primary base grid", i.e. a structure which resembles the original structure topologically but which is made up only of

elementary units which extend from one branching node to the other, or from a branching node to a branch tip. (See Section 5.5, p. 119, for more details.) This modified structure is saved as pbg-file *and* will also be present in the memory. It will be referred to in the menu items as the "first modified structure". After saving the pbg-file (together with some data display on the screen), there will appear a new submenu entitled "Further transformation of the structure" and containing 7 items. Basically, there is now the alternative to go back to the main menu (item a) or to continue the transformation process, leading to a so-called "secondary base grid" (sbg) which is topologically and geometrically changed by melting branch nodes which are "too close together" (items b to g). There is the possibility to create and save several sbg-structures, one after the other.
The submenu items b to g, showing the text

> Secondary dissection using median *(nth percentile, minimum)*
> of L/LSC: *(number)*

or

> Secondary dissection using arbitrary (L/LSC)-creation factor

offer several possible parameter combinations for the creation of the sbg-structure (see Section 5.5). If one of the items b–f is selected, the indicated percentile value (resp., minimum) will be chosen as "creation factor", whereas item g demands the explicit specification of this factor by the user with the call

> Please specify the L/LSC value to be used as creation factor:

When the desired number is given in, there will appear another question, which is also the next output after one of the items b–f was activated:

> Please specify the min. ratio for L/crit.dist. (preferable: 0.5):

When this input is done, there will appear another screen protocol (if not switched off), and eventually the display

```
STORING THE ACTUAL STRUCTURE USING DATA FORMAT C
The data are written into a file with the extension
                        .sbg.
name of the file (without the prescribed extension .sbg):
```

Now, after specification of a filename without suffix, the sbg structure will be saved and will also be present in the memory. This structure will be referred to in the menu items as the "last modified structure".

GROGRA will afterwards return to the submenu "Further transformation of the structure". If now one of the items b–g ("Secondary dissection...") is selected again, the sbg-structure will be overwritten (in the memory, but of course not in the sbg-file, if a new filename is specified). When we return to the main menu, we

have now both modified structures (the pbg- and the sbg-structure) — together with the original, unmodified sequence of structures — in the background and can list or watch them by graphical display.
Attention: If the item "Transform and save for HYDRA" is activated a second time, both structures will be overwritten.

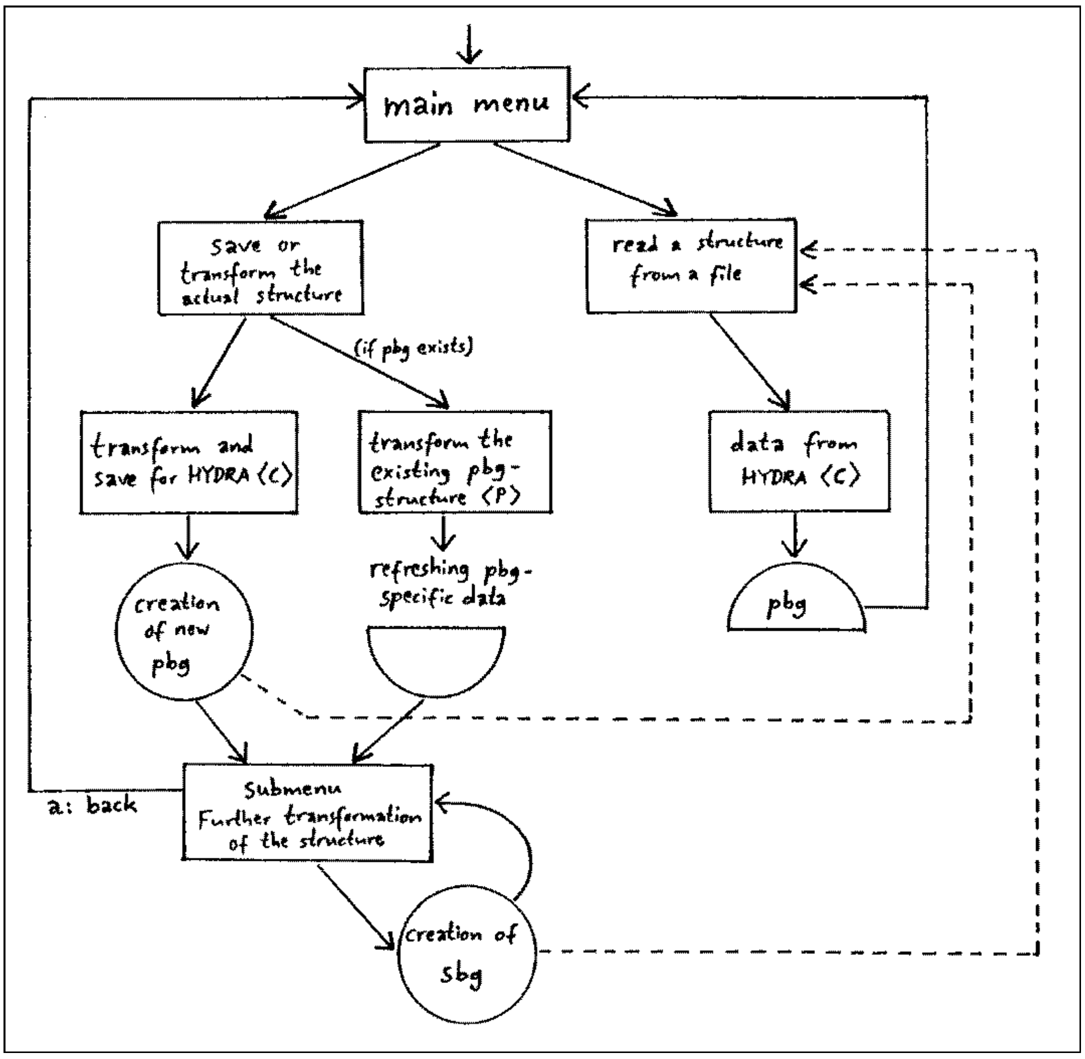

main menu
save or transform the actual structure
read a structure from a file
(if pbg exists)
transform and save for HYDRA <C>
transform the existing pbg-structure <P>
data from HYDRA <C>
creation of new pbg
refreshing pbg-specific data
pbg
a: back
Submenu Further transformation of the structure
creation of sbg

Fig. 26: Menu calls associated to the interface to HYDRA

Transform the existing pbg-structure <P> [Save or transform the actual structure]

This menu item does only appear when a pbg-structure is present in the memory, either created by activating the preceding item "Transform and save for HYDRA" (see above) or by reading a pbg- or sbg-file. It enables the creation of sbg-files from input in HYDRA style read from a file or from a just created pbg-structure, when the transformation process was deliberately interrupted by the choice "a: Back to the main menu" in the submenu "Further transformation of the structure".

The procedure after activation of this menu item is an abbreviated version of the procedure encountered when having activated "Transform and save for HYDRA" (see above). There will be a re-calculation of some data (LSC values) associated to the pbg-structure, and afterwards the submenu "Further transformation of the structure" (which was already discussed above) will appear, giving the opportunity to create sbg-structures.
Fig. 26 visualizes the possible orders of menu calls in connection with the interface to HYDRA. See also Section 5.5, p. 119.

Transform all the parameters N <**N**> [Save or transform the actual structure]

By activating this menu item, it is possible to overwrite the variable n (standing in most applications for leaf area of a shoot) of all elementary units of a structure. (This step changes the structure, so the structure should possibly saved in a dta-file before.) As the modification of the n values will be carried out for only one single structure, not for the whole developmental sequence which is possibly present in the memory, the first question will be for the number of the structure to be transformed, like for the other menu items described above. (There is no possibility foreseen to modify the n-values of a pbg- or sbg-structure.)
After the number of the developmental step is specified, there appears a submenu entitled "Transformation of parameter N (Attention! This is irreversible!)" which offers, besides the possibility to go back immediately, the following options:

Transformation according to linear law <**B**> [Transform all the parameters N]

If N is the old value of variable n and N' the new one, the new value will be calculated for each elementary unit from the equation

$$N' = interc + nfact \cdot N,$$

where $interc$ and $nfact$ are real-valued constants which have to be specified by the user when this option is selected.
Of course, this transformation makes only sense if there are nonzero N-values present in the actual structure.

Transformation according to power law <**C**> [Transform all the parameters N]

This option is similar to the preceding, it differs only in the transformation law:

$$N' = nfact \cdot N^{expon},$$

where $nfact$ and $expon$ are again user-specified constants. This as well as the preceding transformation are designed primarily for the purpose of transforming one leaf parameter (e.g., needle dry weight) into another (e.g., needle surface) with the help of some standard regression equation.

Linear dependence from length and age-dependent decline <**D**> [Transform all the parameters N]

In contrast to the preceding transformations, this one is designed for structures where no values of the variable n are yet specified for the elementary units. It calculates the new N-values in dependence of two other elementary unit variables, namely, length (ℓ) and age (a), the latter being taken from the generative distance g by $a = g_{\max} - g$, where $g_{\max}$ is the maximal value which the elementary unit variable g assumes in the structure to be transformed. The transformation equation is

$$N = agefact(a) \cdot (interc + nfact \cdot \ell).$$

The real-valued constants *interc*, $nfact$ and *agefact* (0), *agefact* (1), ... , *agefact* (7) are asked from the user. It is assumed implicitely that $agefact(a) = agefact(7)$ for all ages $a > 7$.
(For realistic conifer examples, one should choose *interc* to be roughly zero, $nfact$ roughly 80, $agefact(0) = 1$ and $agefact(a)$ as tending against 0 for high values of a.)

In all cases, the transformation is carried out without a confirming message; one turns immediately back to the main menu.

e Show the actual structure graphically [Main menu]

Under this item, not only the graphical display on the screen is accessible, but also the options to write files in HPGL or Postscript format for a plotter or printer.
The graphical display of an attributed geometrical structure is done by first projecting it along a user-selected direction on a plane orthogonal to that direction (parallel projection) and then determining a rectangular part of this plane for showing it on the screen and / or plotter / printer. The first submenu entered when having activated the menu item is therefore entitled "Direction of view" and offers three possibilities:

side view <A>
view from above <B>
arbitrary direction of view <C>

The first option defines the projection to be carried out parallel to the y axis, the second one parallel to the z axis. The third option entails two requests:

GRAPHICAL OUTPUT
Please specify the direction of view:
    Angle in the xy plane (y-direction = 0) in degrees?
    Slope of view relative to the horizontal (in degrees)?

Here, two angles, an azimuth β and a slope γ, have to be specified (Fig. 27).

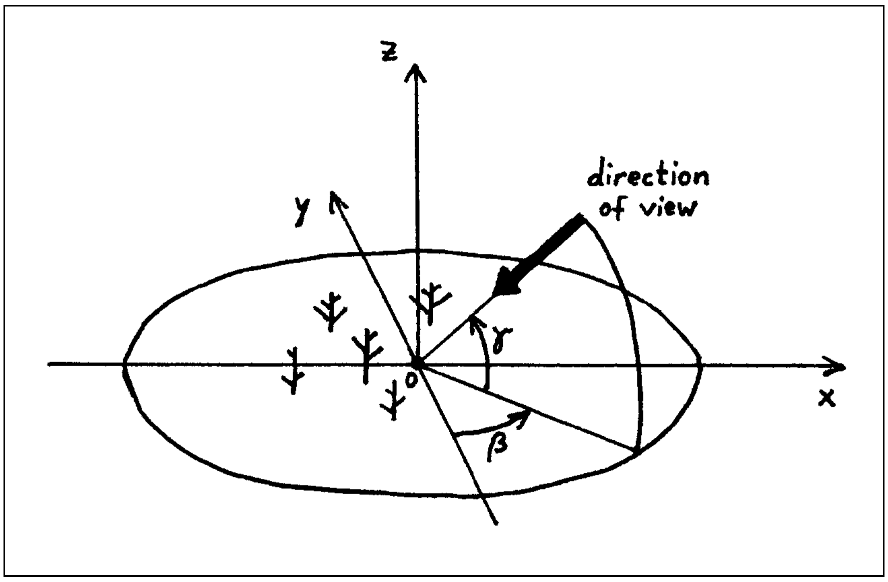


Fig. 27: Specification of view

If a pbg-structure, or a pbg- and an sbg-structure, are present in the memory (see Section 5.5, p. 119 ff.), there will appear a submenu

> What shall be given?
> The unmodified structure <A>
> The first modified structure <B>
> The last modified structure <C>

(the last item shown only when an sbg-structure is present). The first option refers to the standard developmental sequence, the second one to the pbg-structure, and the third one to the last sbg-structure which was created (and which is then automatically the only one still present in the memory). The display of a pbg- or sbg-structure includes always only one developmental step.

In the UNIX version of GROGRA, there will be another submenu entitled "Display" with the options

> Standard format <S>
> Full screen <F>

The option "Standard format" defines for the graphical display a window on the screen which has the same size than the window used for the menus. This can have advantages because a part of the workstation screen remains free for other applications to be run parallel. In the DOS version, the "full screen" option is assumed automatically.

After these specifications, there will possibly be a certain waittime because the coordinates in the projection have to be calculated and their maximal and minimal values to be determined. Then a text display entitled "Now there will be the graphical output" appears which contains the coordinate limitations of the area from the projection plane to be shown, and informations about the commands which are possible in the graphics mode of GROGRA. These commands will be explained in Section 5.3, p. 107. By a keypress (<return> in the UNIX version) one will enter the graphical display, showing the actually present developmental sequence of structures, one structure after the other (the next one being displayed by pressing the <space> key), and with the number of the structure displayed in the upper left corner. All structures of a developmental sequence are shown in the same scaling. Structures which are empty (or which are not generated according to a specification to leave them out) produce a black screen showing only the number. The graphics mode is left when all structures of a sequence are shown, or when the <return> key is pressed. The main menu will be entered afterwards. (See Section 5.3 for further possibilities in the graphics mode.)

f Analyze the actual structure [Main menu]

Several analysis options can be applied to the structure which is present in the memory. Most of them write their results directly in files, formatted for further treatment by statistical analysis programs like SAS or by other modelling or display software. Only the first two options indicated in the submenu, named "elementary analysis" and "pathlength analysis", offer also the possibility to write their results directly to the screen.
Among the analysis options, there are also the data interfaces to the software systems GROBOL (stem analysis) and 3dCLIP (grid with cubic cells), which will be discussed in further detail in Section 5.5.

elementary analysis <A> [Analyze the actual structure]

This analysis option provides some fundamental data about all the structures of the developmental sequence, e.g. numbers of elementary units (shoots), length and volume sums, average branching angles. After activation of the menu item, GROGRA will ask for the output device (Screen <B>, printer <P>, or file <F>). If the file option is selected, a filename has to be specified later. The filename will be automatically completed by the suffix .txt.
If a pbg-structure, or a pbg- and an sbg-structure, are present in the memory (see Section 5.5, p. 119), after the output device - menu there will appear another submenu entitled "What shall be analyzed?" and giving the choice between "The unmodified structure <A>", "The first modified structure <B>" and, if an sbg-structure is present, "The last modified structure <C>" (cf. the menu item "Show the actual structure graphically" above). The first option refers to the standard

developmental sequence, the second one to the pbg-structure, and the third one to the last sbg-structure created.
For larger structures, the elementary analysis can take considerable time. Eventually, there will appear the heading "Basic data", and for each structure the lines

Number of shoots
No. of terminal shoots
aver. no. daughter sh. of nt-sh. *(= average number of daughter shoots of non-terminal shoots)*
global sum of all shootlengths
sum of shoot volumes
sum of values of parameter N
maximal z coordinate (height)
max. extension in x direction
max. extension in y direction
max. radius projection xy-plane *(= maximal radius of the projection of the structure along the z-axis on the xy-plane)*
average branching angle
average contraction factor

Most of these parameters are self-explanatory. The "average contraction factor" is the mean value of the ratios ℓ/ℓ_m, where ℓ is the length of an elementary unit not prolonging the axis of its mother unit, and ℓ_m the length of the mother unit. (Prolongation of an axis is assumed if both units have the same branching order.) The "average branching angle" does also refer only to non-prolonging, i.e. properly branching units.
The propagation to the next developmental step is done by <return>. For the developmental steps 2, 3, etc., there will in addition be given the increment relative to the preceding developmental step for each of the above parameters.

pathlength analysis <**B**> [Analyze the actual structure]

This option works only for the standard developmental sequence of structures, not for pbg- or sbg-structures. The specification of the output device is done like in the "elementary analysis" above, the automatically attached filename suffix being ".pfa" here. If the output device is "file", only the last developmental step will be analyzed. (If the analysis of another step is desired, one can save the structure to be analyzed as a single-step dta-file and then load it from the file.)
For each structure, a table with 5 columns will be given. Each line in the table corresponds to an elementary unit.

Column 1: Number of the elementary unit.
Column 2: Diameter d_b of the unit.
Column 3: Average length of all paths emerging from that unit and

going to a terminal unit (= unit without daughters).
Column 4: Length of the longest path of that kind.
Column 5: Number of the terminal unit where the longest path ends.

It is possible to use the pfa-files as input for SAS when further statistical analysis of the pathlength - diameter - relation is of interest (cf. [84]).

Because the screen display is implemented only for test purposes in this case, a table output with more lines than fitting on the screen will not be interrupted automatically. To avoid a loss of information, use the printer or file output instead.

grid with cubic cells <**C**> [Analyze the actual structure]

This item as well as the next one, "grid with cubic cells, simplified", represent the data interface to the 3dCLIP radiation regime simulation software.
The whole three-dimensional space where the structures "grow" is divided into cubic boxes, whose orientation and sidelength can be specified by the user. For each box (cell, voxel) of this discretizised space, several parameters concerning the part of the structure lying in this box are calculated and written into a file. The details are explained in Section 5.5, p. 127, here only the steps to be performed will be commented.
The cubic grid analysis is only applicable to one single structure out of the standard developmental sequence (not to pbg- or sbg-structures). Thus, the number of the developmental step is to be specified first. After the line

```
Doing analysis related to cubic grid. Please wait.
```

has appeared, several further specifications have to be made. First of all, an orthonormal basis has to be given, describing the directions of the cell's edges. This is done by the input of two vectors (coordinate by coordinate); the first of them is taken as the first grid direction, whereas the second one (which must not be a multiple of the first) is projected on the plane orthogonal to the first vector. The resulting vector is the second grid direction, and the third direction is determined by orthogonality to the first two directions.

The next specification fixes the side length of the cells. Because the cells are assumed to be cubes, only one length is to be given in here. (Note that the smaller the value, the more cells will be generated and the larger the resulting data set of the analysis, leading possibly to very large time-consuming calculations.)
Afterwards, a display

```
Saving grid-related data
```

will appear, and a filename will be required. This name gets automatically the extension .kub.
During the writing process, which can take considerable time, there will be comments like

```
... beginning to initialize boxfile
      preface of file finished.
Initialization complete.
```

For informations about the contents of the kub-files, see Section 5.5, p. 127.

grid with cubic cells, simplified <**D**> [Analyze the actual structure]

The steps to be carried out under this menu item are the same as under the preceding item, "grid with cubic cells". Only the data set written to the kub-file is reduced, see Section 5.5, p. 127.

stem analysis <**E**> [Analyze the actual structure]

This item generates a file for the further display and analysis with the software GROBOL (growth of boles engine). All data collected here concern only the "stem", i.e. the set of elementary units of a structure which have the branching order 0. Only the standard developmental sequence is analyzed, no pbg- or sbg-structure. The data for all structures of the sequence are written into one and the same file.
Immediately after the activation of the menu item, the filename will be asked for. It will be automatically complemented by the suffix .bol. After its specification, there will be a control line

```
Nullniveau: <number>
```

and then the message

```
Saving of stem data finished, press <return> to continue.
```

See Section 5.5, p. 126, for a description of what the bol-file contains.

numbers of daughter shoots <**F**> [Analyze the actual structure]

Under this menu item, a file with special data meant for further analysis with SAS can be produced. Only one single structure out of the standard developmental sequence (no pbg- or sbg-structure) can be analyzed. The number of the developmental step has to be specified first. Then the line

```
ANALYSIS OF THE ACTUAL STRUCTURE: NUMBER OF DAUGHTER
SHOOTS
```

must appear, and a filename will be requested. It gets automatically the suffix .dat, thus indicating its purpose as data input for SAS. The end of the writing process is indicated by the message

```
Analysis is finished, press <return>.
```

The produced dat-file will contain a table with 9 columns and as much lines as elementary units with non-negative branching order are present in the analyzed structure. Each line of the table corresponds to an elementary unit (shoot). The informations in the columns are the following:

> *Column 1:* The number i of the elementary unit, encoded in the form s$<$ *number* $>$, where $<$ *number* $>$ has exactly 6 digits,
> *column 2:* The length ℓ of the elementary unit,
> *column 3:* The diameter d_b of the unit,
> *column 4:* The branching order b of the unit,
> *column 5:* The age a of the unit, calculated as $a = g_{\max} - g$, where g is the generative distance of the unit and $g_{\max}$ the maximal generative distance appearing in the structure which is analyzed,
> *column 6:* The number of daughter units of the unit (including axis-prolonging daughters),
> *column 7:* The number of *subapical* daughter units of the unit, i.e. daughter shoots with q-value less than 0.2 and branching order $b + 1$ (non-prolonging),
> *column 8:* The number of *medial* daughter units of the unit, i.e. daughter shoots with $0.2 \leq q < 0.8$,
> *column 9:* The number of *basal* daughter units of the unit, i.e. daughter shoots with $q \geq 0.8$.

Note that the values of the last three columns must not necessarily sum up to the value of column 6, because there can be a daughter unit prolonging the axis of the current unit (i.e. having the same branching order b). Elementary units with negative branching order (e.g., the fictitious basal shoots of dtd-read or hand-specified structures) are not analyzed and do not appear as lines in the table.

lengths and angles <**G**> [Analyze the actual structure]

The purposes and the steps to perform under this menu item are similar to those of the preceding item, "numbers of daughter shoots" (see above). The filename suffix is also .dat here.
After the specification of the filename, GROGRA will ask for a "Cluster distance" d_c. This distance refers to an iterative algorithm which will be applied to each elementary unit of the analyzed structure, and which has the purpose to collect all daughter units of that unit lying sufficiently close together into a "cluster", identifiable by a cluster index. The cluster index 0 is reserved for axis-prolonging daughters, i.e. units having the same branching order as the unit considered. The clusters of side branches are numbered by 1, 2, ... according to the position of the cluster at the elementary unit (the more basal the position, the higher the number).
The clustering algorithm is performed for each elementary unit in the following manner:

Clustering algorithm

> First, the absolute positions of all daughter unit basal points at the unit at work are collected in an ordered list.
> Then, the two entries in the list having the smallest distance from each other are replaced by their mean value, if their distance is smaller than d_c.
> This step is repeated until all mutual distances of entries are $\geq d_c$.
> At last, all daughter units with identical entries are conceived as belonging to the same *cluster* and get the same cluster index.

(See Section 5.5, Transformation for HYDRA (p. 119) for another application of this clustering algorithm.)
The cluster index of a daughter unit is written into its x_b variable. It represents a discrete positional information, in contrast to the real-valued positional variable q.
This information is used during the determination of two elementary units which are searched for each unit during the analysis, called the "big brother" unit and the "cousin" unit.

The *big brother* of an elementary unit is the axis-prolonging unit of its mother unit, i.e. the daughter (or "son") of its mother which has the same branching order than the mother (Fig. 28). It must not necessarily exist. If the unit is itself axis-prolonging, it is its own big brother.

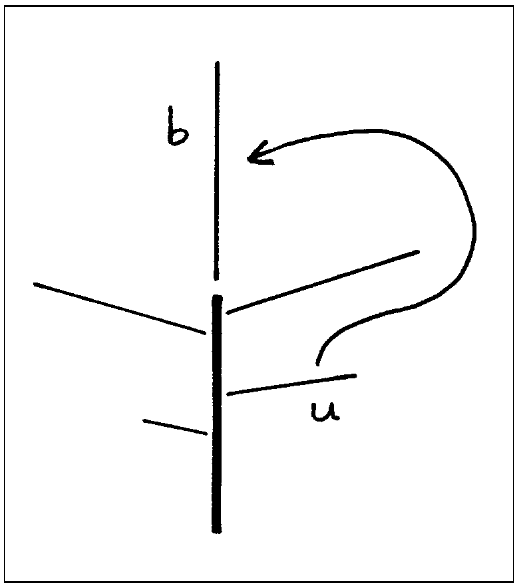


Fig. 28: Big brother (b) of unit u

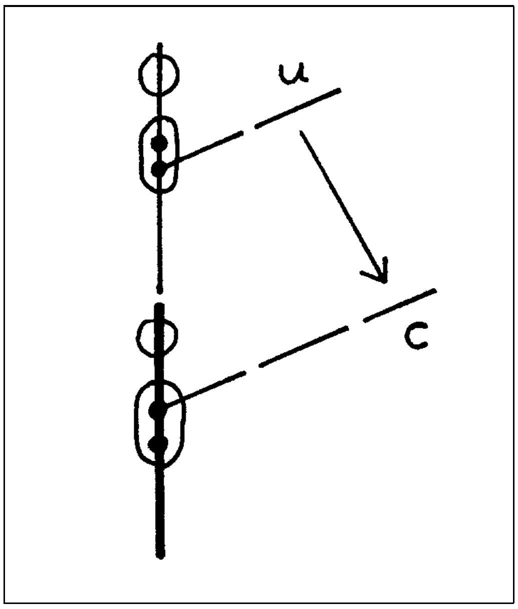


Fig. 29: Cousin (c) of unit u

The *cousin* of an elementary unit u is a unit with the same generative distance g than u. When we call "main ancestor" of a unit that unit which is the first one having a smaller branching order when iteratively mother units are considered, the main ancestor of unit u must be the axis-prolonging daughter unit of the main ancestor of the cousin c. Furthermore, the cluster where the branch bearing u emerges must have the same cluster number than the cluster from which the branch bearing c emerges (Fig. 29).
The cousin must not exist, and it is not necessarily uniquely determined. GROGRA chooses everytimes the first cousin it finds.

We can now explain the content of the table forming the dat-file which is produced during this analysis. The table has 13 columns, and each line corresponds to an elementary unit, like in the preceding analysis option.

Column 1: The number i of the elementary unit, encoded in the form s$<$ *number* $>$, where $<$ *number* $>$ has exactly 6 digits,
column 2: The number i_m of the mother unit, encoded in the form m$<$ *number* $>$, where $<$ *number* $>$ has exactly 6 digits,
column 3: The branching order b of the unit,
column 4: The age a of the unit (cf. the preceding option, "numbers of daughter shoots"),
column 5: The cluster index x_b, determined by the clustering algorithm,

column 6: The relative position q of the unit,
column 7: The absolute position of the unit at the mother unit, i.e. $q \cdot \ell_m$ (ℓ_m = length of mother unit),
column 8: The branching angle, i.e. the angle between the unit and its mother unit,
column 9: The length ℓ of the unit,
column 10: The number of daughter clusters of the unit (except the eventually existing axis-prolonging daughter),
column 11: The length of the big brother of the unit, if it exists; otherwise: -1,
column 12: The length of a cousin of the unit, if one exists; otherwise: -1,
column 13: The length ℓ_m of the mother unit (if it exists; otherwise: -1).

During the analysis, there will be the comments

```
... labelling daughter shoots ...
... analyzing ...
shoot s000001
shoot s000002       etc.
```

The end of the writing process is indicated by

```
Analysis is finished, press <return>.
```

w Service functions and explanations [Main menu]

Here, a submenu with 7 items, entitled "Explanations and Service", is evoked.

Explanation of the turtle commands <**T**> [Service functions and explanations]

A short overview of the turtle command language is given on three screen pages. A detailed explanation of the same material is given in Section 4.1 of this manual (page 50 ff.).

Explanation of the expression syntax <**E**> [Service functions and explanations]

Some remarks about the construction of arithmetical expressions are given on one screen page. A detailed explanation can be found in this manual in Section 4.4 (page 69 ff.).

Explanation of the L-system syntax <**L**> [Service functions and explanations]

Some material about rules, conditions, variable declarations and the repetition operator, including some examples, is given on 7 screen pages. More elaborated explanations on these subjects can be found in the Sections 4.2 – 4.5 of this manual (page 61 ff.). One further screen page shows the actually implemented restrictions on parameter name lengths, variable numbers, arities etc. It is important to check these values when a new GROGRA version is to be applied.

Change the colors <**F**> [Service functions and explanations]

When this item is activated, the user can choose between two possibilities: Coloured (the default setting) and monochrome. However, the colour adjustment does only have effect on structures which are generated afterwards, not on the display of an already existing structure. Moreover, only the default value (p_g) of the unit colour will change to white, i.e. an explicit colour specification by a P command in the grammar file will override the "monochrome" adjustment.
The colour change menu item is essentially a remnant of a former GROGRA version where the explicit colour control with P commands was not yet implemented.

Change the language <**A**> [Service functions and explanations]

The activation of this menu item works as a switch between English and German text display. The default language for GROGRA is English. When the German modus is switched on, all menu displays and other text output (except a few error messages which do seldom occur) is given in German. To undo this change, the same item is to be activated again.

Change the internal mode of memory <**S**> [Service functions and explanations]

This is a switch which toggles between "RAM mode" (the default setting) and "HD mode", referring to the storage medium where GROGRA holds internally the actually present developmental sequence of geometrical structures. (There is no influence on the handling of the strings generated by the grammars; these are saved in auxiliary files on the hard disk in any case.) HD mode offers more place for large structures, but RAM mode is generally quicker and allows all possible analysis and transformation options, what is not the case for HD mode. (See Section 5.4, page 108, for the restrictions inherent to HD mode.)
When the mode of memory is changed, the present sequence of structures will be deleted. However, it is possible to write a sequence of structures to a dta-file in one mode and to read it again in the other mode (if there is enough space). When on a PC the memory is not large enough for a structure, it can eventually be generated in HD mode, saved in a dta-file, and possibly later analyzed or transformed on a workstation with more available RAM in RAM mode.

q Quit the program. [Main menu]

If there is a structure still present (independently of RAM- or HD-mode), GROGRA will, before finishing with deleting that structure, display a small submenu

> Warning: The structure is not saved.
> Delete structure <L>
> Back to Main Menu <Z>.

(This submenu will not appear if saving to a dta-, dtb- or pbg-file was done before.) To leave GROGRA definitively, the first item, "Delete structure", is to be selected. If "Back to the Main Menu" is chosen, nothing will happen to the structure, and the main menu is displayed again.

5.3 The graphical display

The graphical display of GROGRA is activated via the menu item "Show the actual structure graphically" (see Section 5.2 above, page 96). In RAM mode, the auxiliary unit variables x_b, y_b, x_t, y_t will for each unit be overwritten with the coordinates of the unit's bottom and tip in the chosen projection plane. The area to be seen on the screen is determined from the maximal and minimal values of these coordinates. The diameters of the elementary units are shown in roughly the same scale as the lengths.

The way a unit is shown on the screen is rather simple: According to its top diameter d_t, it is drawn as a line or as a filled rectangle, the colour always being specified by the unit variable p. (If p is 0 or a multiple of 16, nothing is shown.) Only if the unit is seen from a direction directly from above in the projection, it is drawn as a filled circle. (For choosing the direction of view, see Section 5.2, p. 97.)

During the graphical display, several single-key commands are available to evoke special options, among them plotting, printing and zooming. *All these commands work only when graphical display is already entered.*

<space> causes GROGRA to continue with showing the next developmental step of the current structure sequence. If already the last structure is shown, one will return to the main menu. — All steps of a developmental sequence have the same scaling.

<return> stops the graphical display and causes GROGRA to return to the main menu.

z (= zoom): After *z* is typed in, a rectangular part of the screen content can be specified with the mouse to be displayed in maximal magnification. This is done by pressing the left mouse button first at the position of the lower left corner, then at the position of the upper right corner of the rectangle which is to be magnified. (If the selection is done in the wrong order, there will be an error message at the screen bottom.) The zooming option can be used iteratively by pressing again the *z* key and then choosing a rectangle once more. — If a small part of a large structure is selected, it can cost some time until the display has its final form after zooming, because the parts lying "beyond the borders" of the screen take also time while drawing the structure.
The scaling factor achieved by zooming is automatically maintained during the next developmental steps when the <space> key (or the "*c*" key) is pressed.

e (= default expansion): This key serves as "undo" for the zooming, i.e. the original scaling is re-established.

h (= HPGL-plot): For the currently shown developmental step, a file is written in HPG language, containing the plotter commands necessary to draw the structure with a HPGL-compatible plotter. Before writing the file, a submenu will be shown where the paper format is to be specified (DIN-A-4 or DIN-A-3). Furthermore, a filename (which gets automatically the suffix .hpg) and a pencil index for monochrome plotting are to be specified. There is also the possibility to plot in a polychrome manner by choosing the pencil index 0. The actual pencil for each unit will then be determined by a reduction modulo 8 of the colour index *p* of the unit.

p (= Postscript output): Analogously to the *h* command, a file is written containing the graphical informations for the currently shown structure, but now in Postscript language for the display on a Postscript-interpreting printer. A paper format (DIN-A-4 or DIN-A-3) is to be selected and a filename has to be specified, which gets automatically the suffix .ps. In the current GROGRA version, the printing is monochrome. After writing the informations to the file, the same structure will appear on the screen again, and other graphical display commands can be used.

d (= print): This key does only work when a NEC-P5 or compatible printer is directly installed at the parallel lpt1 - printer port (PC version only) and is ready to print. It will produce an immediate printer hardcopy of the current screen content. This option is meant for rough test pictures; for higher-quality copies, the Postscript output (command *p*) is recommended.

c (= cycle): When this key was pressed, GROGRA will automatically show each developmental step for 1 second, then turn to the next step, and after showing the last step begin with the first one again. This "movie" modus is useful for demonstrations and for situations when the user is not willing to press repeatedly the space key. The automatic propagation through the sequence can be stopped at every moment by simply pressing <space>. (Every other key will also do.)

5.4 RAM- and HD-mode

The internal mode of representing attributed geometrical structures can be changed from the (default) RAM-mode to HD (hard disk) mode. This is done by activation of the corresponding item in the submenu "Service functions and explanations" (see Section 5.2, p. 106). In HD-mode, the capacity for developmental sequences of structures is only limited by the available space on the hard disk. However, in HD-mode there are certain restrictions to the capabilities of GROGRA.

When HD-mode is active, GROGRA writes new structures (whether constructed from grammars or read from files) into an auxiliary file named vlstru.tem, which will be located in the same subdirectory where GROGRA is working. When the initialization of this file fails, GROGRA will stop with the message

```
Bad mistake! File vlstru.tem could not
be opened. Program halted. Press <return>.
```

In the file, the elementary units will be represented sequentially, each one by the variables

i	(identifier)
i_m	(mother unit identifier)
ℓ	(length)
d_b	(basis diameter)
d_t	(top diameter)
n	(leaf parameter)
p	(colour index)
b	(branching order)
g	(generative distance)
q	(relative position)
P	(basis position vector)
Q	(top position vector)
H	(head direction vector)
L	(left direction vector)
U	(upward direction vector)

(in this order). The representation is not in ASCII code, but in a machine-dependent, more compact encoding. The single units are separated by a marker consisting of one character, namely, "a" in the interior of a structure description, "n" when a new structure begins, and "e" at the end of the file.
It is to be emphasized that the variables c, x_b, y_b, x_t and y_t are not existing in HD mode.

The following GROGRA functions *do not work in HD mode*:

- The generation of structures from *sensitive* grammars,
- the input of data in HYDRA format (pbg, sbg, bag or map),
- the transformation and output of data for HYDRA,
- the transformation of the unit variables n,
- the analysis options except the stem analysis. That is, the interface to 3dCLIP (cubic grid analysis) is also not operating in HD mode,
- the J specification in the dtd file format (see page 115),

- the method calls $M2$ and $M3$ (see Section 4.6, page 74).

Generally, working with structures in HD mode will take more time than in RAM mode because of the necessary file access operations. This is also true for the graphical display.

5.5 Interfaces and data formats

All external data which GROGRA reads or writes are encoded with the standard ASCII character set.
The different data formats which GROGRA handles can be distinguished by the extensions of the filenames which are used. We exclude the *Postscript* and *HPGL* formats (filename suffixes .ps and .hpg) from this discussion; consult a Postscript or HPGL manual for informations concerning these graphics languages. Also, the tables created by certain analysis options in .dat- or .pfa-files will be excluded from this Section because they were already explained in Section 5.2, menu item "Analyze the actual structure" (p. 99; 101 ff.). The input format for the growth grammars (.lsy- and .ssy-files) is explained in Chapter 4.

Standard format (filename suffix .dta)

This is the format normally used by GROGRA to save a complete developmental sequence of attributed geometrical structures in a file. It can also be used to save single structures (see Section 5.2, p. 92).
The structures of a developmental sequence are written sequentially into the file; each new structure is announced by an extra line with the single letter "n" in its leftmost position. The end of the file is marked by the letter "e". For each structure, the elementary units are again listed sequentially in the order how they are linked by their s references (or, in HD mode, how they are arranged in the file vlstru.tem). The variable set for each unit takes three lines in the file and is announced by an upper case "S" at the beginning of the first line:

$$\begin{array}{llllllllll} \mathsf{S} & i & i_m & \ell & d_b & d_t & n & p & b & g \; q \\ & p_x & p_y & p_z & q_x & q_y & q_z \\ & h_x & h_y & h_z & l_x & l_y & l_z & u_x & u_y & u_z \end{array}$$

The numerical values are separated by blanks. i_m denotes the identifier index of the mother unit. It is artificially set to -5 in the case when no mother unit exists. See Section 5.1, p. 79, for the explanation of the other variables.
Note that the elementary unit variables c, x_b, y_b, x_t and y_t are not saved in the dta-format.

Exchange format for AUTOCAD (filename suffix .dtb)

The data written in dtb-format can serve as input for the AUTOLISP interpreter of the commercial CAD software AUTOCAD (see [3]) to produce simple wireframe-models of single structures (see Fig. 30).

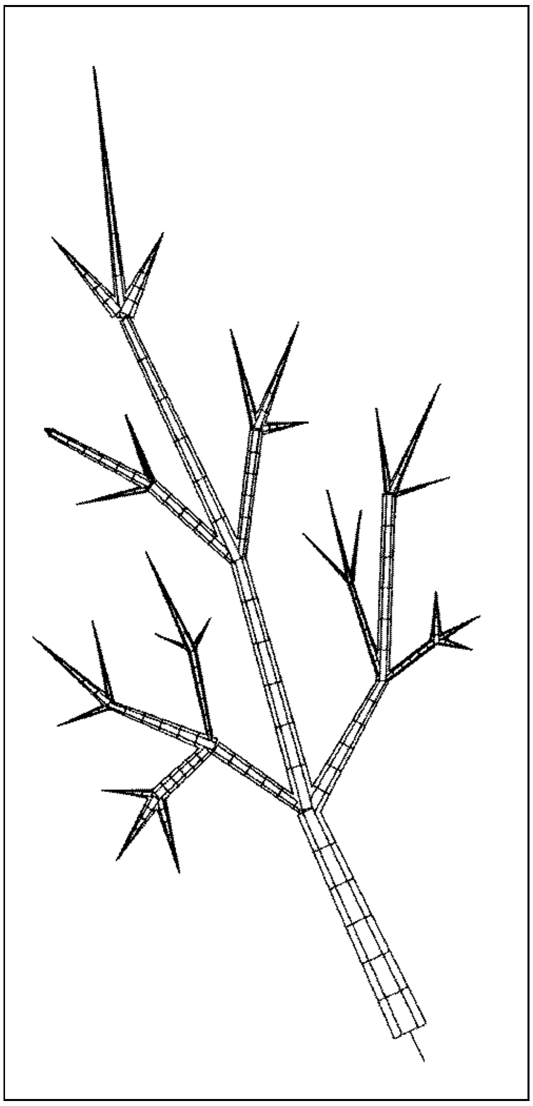


Fig. 30: A wireframe model of a structure
made with the help of the interface to AUTOCAD

Only a restricted data set for each elementary unit is saved in this format. Furthermore, before saving takes place, a rewriting is applied to all prolonging units (i.e. daughter units with the same branching order as their mother unit): The bottom diameters d_b are adapted to the values of the top diameters d_t of the mother units. This transformation ensures a smoother connection of the units in the AUTOCAD display (Fig. 31).

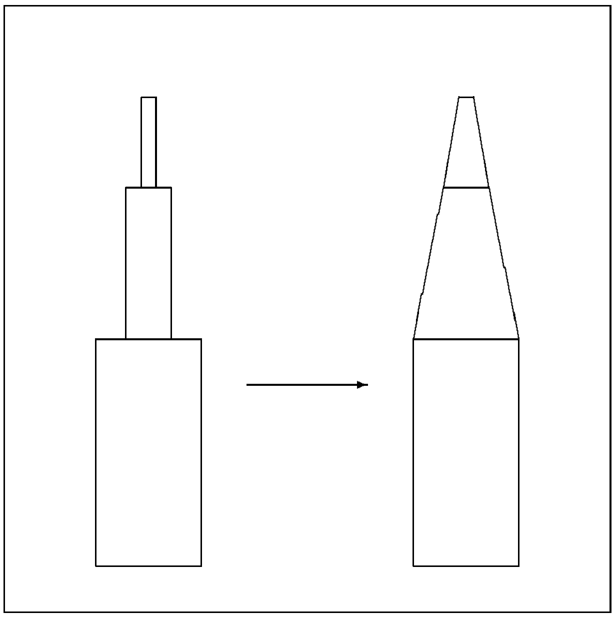


Fig. 31: Adaptation of diameters d_b preceding the saving in dtb format

The data format itself is rather simple. Only one developmental step is saved in a dtb-file. Each elementary unit corresponds to one line in the form

$$(((p_x \;\; p_y \;\; p_z) \;\; d_b) \;\; ((q_x \;\; q_y \;\; q_z) \;\; d_t))$$

where $P = (p_x,\ p_y,\ p_z)$ is the bottom and $Q = (q_x,\ q_y,\ q_z)$ the top position of the unit (cf. Section 5.1, p. 79). Files in dtb-format can also be read by GROGRA (see Section 5.2, p. 88).

Descriptive format (filename suffix .dtd)

This data format is used for reading only. It is meant for morphological descriptions coming from measurements at living branches or branch parts of trees (especially conifers). A dtd-file specifies one structure, no development is included. A labelling of the growth units (shoots) — not necessarily with numbers — is presupposed, such that the topological structure of a branching system can be reconstructed. Shootlengths are also obligatory, whereas the specification of diameters is optional (GROGRA asks for a default diameter for those units whose diameter was left unspecified before reading a dtd-file). Each shoot corresponds to an elementary unit and is described by a single line in the dtd-file. The order how the lines are arranged is only restricted by the requirement that the description of a mother shoot must come before those of its daughter shoots. If the option "buds are to be included" is chosen (see Section 5.2, p. 89), GROGRA constructs extra elementary units for lines containing a "K" in the dtd-file, describing buds (even in the case when the specified length is 0). If this option is neglected, lines containing a "K" are ignored.

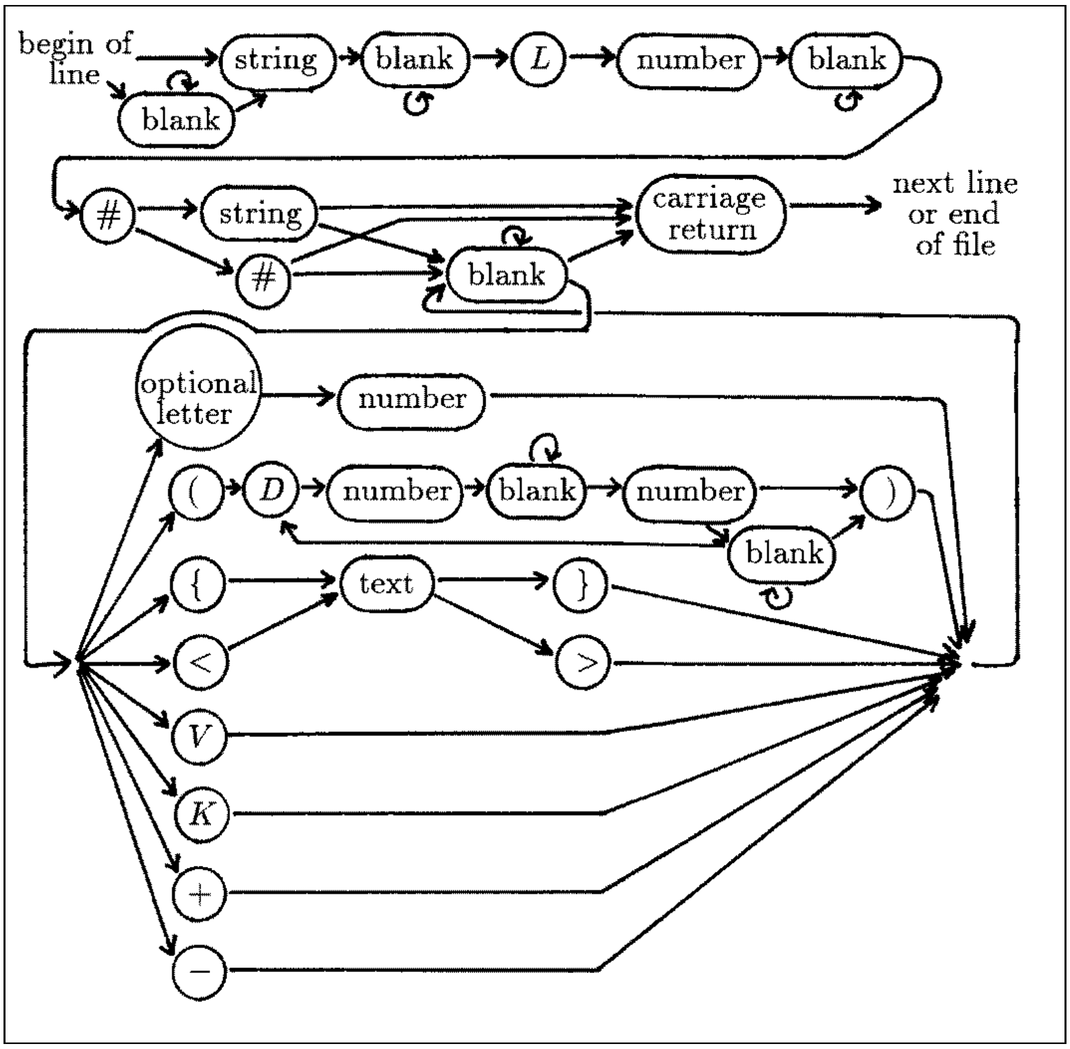


Fig. 32: Syntax of a shoot description line in a dtd file

The syntax of a single line of a dtd-file is given by the syntax diagram in Fig. 32. "Optional letter" is here one of the letters *A*, *G*, *J*, *M*, *N*, *O*, *P*, *R*, *S*, *W*, *X*. Instead of upper case letters, lower case letters are permitted as well (also for *L* and *D*), but have a slightly different meaning (see below). "string" means a string of arbitrary characters not containing # or blanks, "text" may contain blanks, but no } or > character, and "number" refers to decimal numbers in the usual floating point representations. (After G, J, O and R, only integers are allowed.)

The meanings of the different elements constituting a line — and thus a shoot description — are explained in the following table.

Leading string (without preceding extra letter): The unique identifier of the shoot. It can be a number, a word or a combination of letters, digits and extra symbols. GROGRA translates the identifiers into numbers according to an internal list which is created and actualized during reading a `dtd`-file.

L The number immediately following the letter L is the length of the shoot, usually measured in mm.

The string immediately following this character is the identifier of the mother shoot. It *must* be identical to an identifier appearing in some line before as the leading string. The only exception: When the # character is followed directly by another #, the shoot is handled as having no mother shoot. (The position of its origin is then assumed to be (0, 0, 0).)

A The number immediately following the letter A is interpreted as the distance between the basis of the mother shoot and the basis of the shoot itself, i.e. its absolute position at the mother shoot axis, measured from the basis (in the same length unit as L). Usually, this will be a number between 0 and the length of the mother shoot.
If the specification by A is missing, the position will automatically be assumed to be the length of the mother shoot, i.e. the shoot will emerge at the extreme top of its mother shoot.
The q-value of the created elementary unit is also determined from this specification.

W The number immediately following the letter W is interpreted as the branching angle in degrees between the shoot and its mother shoot. When the W specification is omitted, this value is set 0, i.e. the shoot has the same direction as its mother shoot in that case.

S The number immediately following the letter S is interpreted as an azimuth in degrees, i.e. as the angle between the projection of the shoot on the plane orthogonal to the mother shoot axis and the U-vector of the mother shoot. An S specification has no effect when the branching angle is 0. If no S, R, $+$ or $-$ appears in the line, the azimuth is assumed to be 90°.

R The integer immediately following the letter R is interpreted as a short form of azimuth specification, replacing the S option. The R specification is translated as: $1 = 0°$, $2 = 45°$, $3 = 90°$, $4 = 135°$, $5 = 180°$, $6 = 225°$, $7 = 270°$, $8 = 315°$ (i.e. $S = (R-1) \cdot 45°$). $R3$ corresponds normally to a direction to the right of the mother shoot, $R7$ to the left, and $R1$ upwards

(see Fig. 33 for an illustration of the R-directions. The mother shoot has to be thought to go through the centre of the picture.)

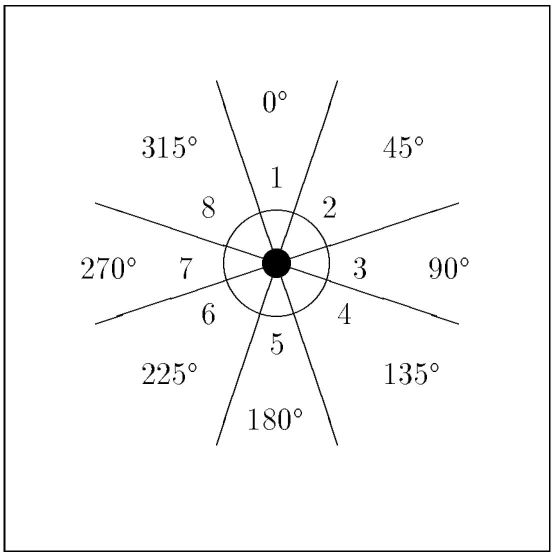


Fig. 33: Azimuth specification by R (inner numbers) and S (outer numbers)

$+$ The single symbol $+$ (enclosed by blanks) is equivalent to $S90$ or $R3$. It specifies normally a direction to the right of the mother branch.

$-$ This is equivalent to $S270$ or $R7$ and specifies normally a direction to the left of the mother branch.

O The integer following the letter O is interpreted as the branching order of the shoot. If there is no O or V in the line, the order is determined automatically: If the branching angle is greater than 0, or if an azimuth was specified, the order is the order of the mother shoot increased by 1, otherwise it is the order of the mother shoot.

V The single letter V (enclosed by blanks) enforces the branching order of the shoot to assume the value of the branching order of the mother shoot. (Prolonging shoot)

G The integer following the letter G is interpreted as the generative distance of the shoot (cf. Section 5.1, p. 79). If no G or J appears in the line, the generative distance is determined from that of the mother shoot by increasing it by 1.

J The integer following J is interpreted as the "age" of the shoot. The generative distance g is calculated from this age by the formula $g = j_{\max} - j + 1$, where $j_{\max}$ is the maximal age appearing in the `dtd`-file. (This way of specifying g does not work in HD-mode.)

N The number immediately following the letter N is directly taken as the parameter n of the corresponding elementary unit. It can stand for the needle surface of the shoot, or for needle dry weight. If no N appears, the default is 0.

K The single letter K (enclosed by blanks) specifies the shoot to be a "bud" and eventually to be omitted if the user wants so (see above).

(*Dnumber number* ... *Dnumber number*) The numbers immediately following the letters D are diameter values, the numbers following them are the positions at the shoot where they are measured, relative to the basis of the shoot. (Hence these numbers should lie between 0 and the length of the shoot.) The positions are assumed to appear in ascending order. In the current GROGRA version, only the first and the last D-specification in the list are really interpreted, namely, as d_b and d_t. (If only one number pair, e.g. (D2.5 0), is given, GROGRA identifies d_b and d_t both with the same value, here 2.5.) If no D-list appears in the line, GROGRA will assume for d_b and d_t the user-specified default value which is asked for before a dtd-file is read.

$\{text\}$ or $< text >$: comment, which is not interpreted.

M, P, X: These letters, followed by numbers, are reserved for specifications which are not interpreted by the current GROGRA version. (M stands for total dry weight of the shoot, P for pure shoot dry weight (excluding the needles), X for the exposition (point of the compass) relative to the stem.)

If lower case letters instead of upper case letters are used in a dtd-file, the corresponding number is meant to be an estimation rather than a measured value. However, GROGRA makes in its current version no difference between upper and lower case letters in dtd-files.
The colour of an elementary unit read from a dtd-file is light green. Its L direction is determined differently in two cases:

Case 1. If the order of the shoot is 0 or 1, L is orthogonal to the vertical direction (and, naturally, to H). This implies that U points "as steeply as possible" upwards, like in the case of explicit construction by hand (cf. Section 5.2, p. 83). If the H direction is already the vertical, the above description does not specify L uniquely. In that case, L will be $(0,\ -1,\ 0)$.

Case 2. If the order is greater than 1, L lies in the plane which is spanned up by H and the head direction H_m of the mother shoot. Furthermore, the signs of L and U are chosen in a way that U differs by no more than 90° from U_m. (If $H = H_m$, it will be assumed $L = L_m$ and $U = U_m$.)

(This convention will possibly be changed in a later GROGRA version because it can lead to some inconsistencies.)

The structure generated from a dtd-file will have at its basis a fictive elementary unit of colour 0 (= invisible), order -1, length 0 and directions $H = (0, 0, 1)$, $L = (0, -1, 0)$, $U = (1, 0, 0)$. This unit yields the reference direction as mother shoot for the shoots specified in the dtd-file by "##" as having "no mother shoot".

Table 3 shows an example of a dtd-file content, and Fig. 34 the corresponding structure.

Table 3: Example of a dtd file content

65-1	L108	##	O1	R1	W90	(D4.0 0 D3.3 20 D2.8 63 D2.6 93)
65-2	L81	#65-1	V			(D2.7 0 D2.3 40 D2.0 68)
65-3	L100	#65-2	V	+	W25	(D1.8 0 D1.6 86)
65-4	L0	#65-1	A68	–	K	
65-5	L22	#65-1	A86	–	W70	
65-6	L48	#65-1	A98	–	W45	(D1.6 0)
65-7	L41	#65-6	V			
65-8	L18	#65-2	A45	–	W70	{ befallen }
65-9	L0	#65-2	A62	–	K	
65-10	L60	#65-2	A73	–	W55	(D1.4 0)
65-11	L0	#65-3	A91	–	K	
65-12	L40	#65-2	A75	+	W50	{ verkrümmt }
65-13	L0	#65-2	A60	+	K	
65-14	L67	#65-1	A101	+	W50	(D1.7 0)
65-15	L54	#65-14	V			
65-16	L0	#65-15	A49	–	K	
65-17	L32	#65-14	A62	+	W55	
65-18	L27	#65-1	A81	+	W65	

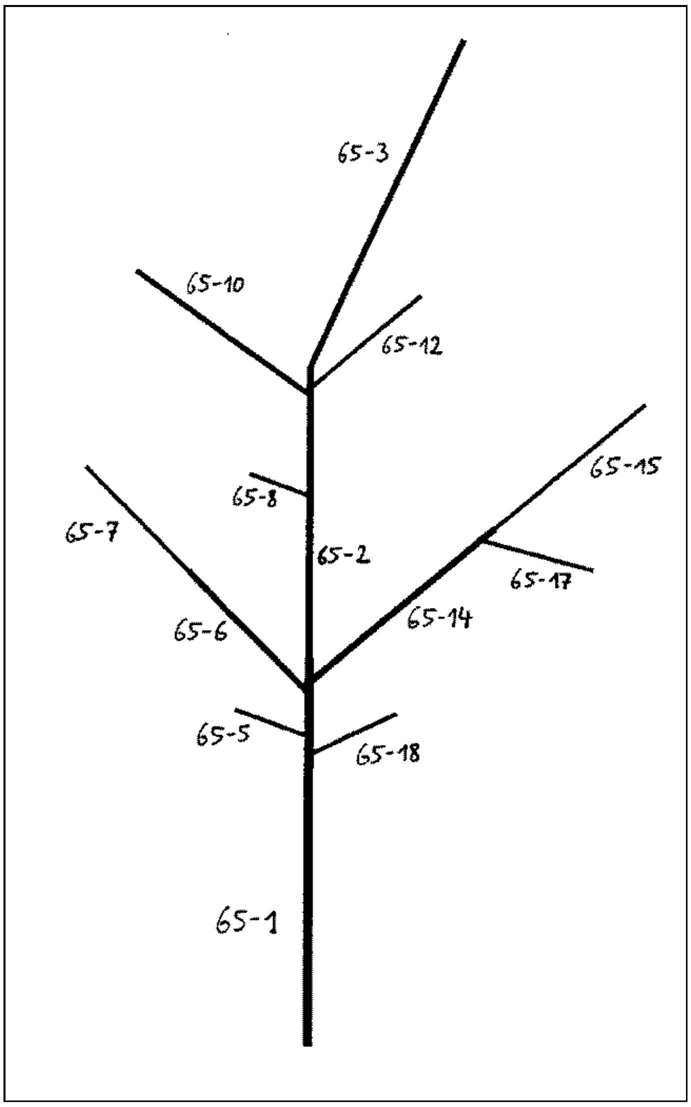


Fig. 34: Graphical visualization of the structure defined by the dtd specification of Table 3 (with shoot labels)

The transformation for HYDRA

This transformation process, ending with files with suffixes .pbg or .sbg (see p. 125 ff. below), is designed for making GROGRA structures usable by the software systems DISC and HYDRA, the latter simulating the tree-internal water transport.

Detailed informations on DISC and HYDRA, including the theoretical background, will be found in [41]. Notions like "leaf specific conductivity", "soil index" or "criterion distance", which will be occasionally used in the subsequent remarks, will also be defined there.
The transformation is started by selecting the menu item "Transform and save for HYDRA" (see Section 5.2, p. 92, for the steps to perform). Only one structure out of a developmental sequence can be transformed. Before the transformation is started, it should be ensured that the n-variables of this structure have generally positive values, at least at the terminal (i.e. daughter-less) units. Because the transformation makes use of leaf-specific conductivities and thereby of accumulated needle surfaces, it makes no sense to apply it to a leafless tree, and this would cause error messages.

The leaf-specific conductivity (LSC) of an elementary unit is calculated according to the formula

$$\mathrm{LSC} = c \cdot d_b^\gamma / n_a,$$

where n_a is the accumulated needle surface area (the sum of all needle surfaces of the units contained in the branching system emerging from the current unit, plus half the needle surface of that unit itself), and c and γ are constants which depend on the tree species. If a pbg-structure is read from a file (see Section 5.2, p. 88), the tree species is asked from the user. When a new pbg- or sbg-structure is constructed, GROGRA in its current version assumes *Picea abies* to be the tree species.

The transformation of a structure into a pbg-structure consists of the following steps (cf. Fig. 35):

1. Creation of forward references in the old structure

The elementary unit variable c is used to link each unit with its axis-prolonging daughter unit (daughter unit with the same branching order b), such that afterwards each axis can be followed upwards by using the pointers c.
Note that the correct execution of this step requires that the branching orders in the structure are correct. Especially, the axes must be distinguishable by equal branching orders of their constituents, otherwise they cannot be identified by GROGRA. Possible sources of errors concerning the branching orders are the bracketing in the grammars or the O- and V-specifications in a dtd-file.

2. Creation of a copy with back references

A copy of the original structure is made. The c variables of the units of this copy point each to the corresponding original unit. (Hence, in the copied structure there are no forward references.)

3. Reorganization of the copy, forming the axis structure

Each axis in the copied structure (an axis consisting of successive units with the same branching order b) is melted into *one elementary unit* representing the whole *axis*. The new length is the sum of the lengths of the constituting units, the direction is that of the basal unit of the axis. Daughter axes will be translated such that their origin maintains contact with the mother axis, and q-values will be transformed consequently. The resulting structure will be hold in the memory during the following transformation steps. We speak of the **axis structure**. The c pointer of each unit in the axis structure refers to the *basal unit* of its corresponding axis in the original structure. By using the forward references in the original structure, it is possible to find for each position on an axis of the axis structure the corresponding unit in the original structure. This possibility will be used in the following when local diameters or local leaf area densities are needed in the axis structure.
The number of units of the axis structure (number of axes) is displayed on the screen and written to the protocol file standard.gpr. It is usually much smaller than the number of original shoots (units).

4. Creation of a copy of the axis structure (the later pbg-structure)

The c references of the new copy are also copied from the axis structure, i.e. they point to the original structure. The c references of the old axis structure, however, are now replaced by *forward references* to the corresponding copied axes.

5. Dissection of each axis of the copied axis structure (creation of the primary base grid pbg)

Each axis is now splitted at each branching node. (However, the length of a unit is bounded to be not smaller than a minimal distance which is 1 length unit in the current implementation.) The diameter- and n-values for the resulting axis parts are determined by referring to the original structure via the c references. (The middle position of the axis part is used here as the position which is searched in the original structure to get the local values of d and n/ℓ.)
The resulting structure will be referred to as the *primary base grid structure* (pbg-structure). It is distinguished by the property that all q-values (relative branching positions) in it are 0, and that each unit ends either in a branching node or in a branch tip. Mathematically, the units of the pbg-structure form the arcs of a (directed, rooted) *tree* in the sense of graph theory, with the further restriction that no vertices (nodes) with degree 2 exist in that tree.

6. Relabelling of the pbg units

The index variables i of the units of the pbg-structure are now overwritten; the new numbering being arranged in *reverse depth-first order*. The basal unit will have the highest index; to the periphery, the indices are monotonically decreasing. Among all daughter units of a unit, the axis-prolonging unit (if one exists) gets the lowest index.

7. Calculation of LSC values and of statistical data in the pbg-structure

The LSC values are written in the x_b-variables of the units of the pbg-structure, the ℓ/LSC-ratios (length expressed in meters here) in the y_b-variables. For the ℓ/LSC-ratios, a ranking is carried out and certain percentiles are determined, which are later on displayed and written into the head of the pbg-file. (It should be remarked that the g-variables of the pbg-units are overwritten during this ranking, such that the generative distance of a unit cannot be read from a pbg-structure. However, this notion has no sense in a pbg-structure anyhow, because the dissection into elementary units is no longer botanically motivated.)

8. Writing of the pbg-file

For the file format, see p. 125 below. Interwoven with the writing process, some further statistical analysis will be carried out, concerning sizes of proper whorls. (This analysis overwrites again the g-variables in the pbg-structure.) A proper whorl is one consisting of at least one daughter unit. The mean proper whorl size is the average number of daughter units of all nonterminal units (cf. "elementary analysis", Section 5.2, p. 98).
The number of segments (units) of the pbg-structure will be protocolled on the screen and in the protocol file.

9. Creation of forward references in the pbg-structure

Like in step 1 (see above), the c references will now be used to connect the segments forming axes in the pbg-structure. In the following phases, the pbg-structure will completely replace the original structure in its function as a basic reference for local diameters and leaf area densities.

After this step has been executed, the control will return to the user; the menu "Further transformation of the structure" will appear (see Section 5.2, p. 93). When the user decides to go back to the main menu, the axis structure and the pbg-structure remain in the memory as well as the original structure, and the pbg-structure can be watched in the graphical display. When afterwards the menu item "Transform existing pbg structure" is activated (p. 94), only a part of step 7 (calculation of LSC values) will be repeated. However, when a pbg-structure is read from a file, there is no axis structure present, and steps 1–3 and 7 have to be done before the menu "Further transformation..." is entered.

When the user decides in this menu to create an sbg-structure and has specified a *creation factor* f_c and a *tolerance factor* f_t (minimal ratio length / criterion distance), the following steps will be executed:

10. Creation of a new copy of the axis structure (the later sbg-structure)

In this step, a possibly present sbg-structure from a former transformation will be overwritten in the memory.

11. Dissection of each axis of the copied axis structure, taking account of the local minimal distance (creation of the secondary base grid sbg)

This step is analogous to step 5 above, but before splitting an axis, a list of the daughter axis origins at this axis will be created, and the *clustering algorithm* (which was formerly mentioned in the context of analyzing a structure, p. 103) will be applied to that list. The minimal cluster distance is determined locally as

$$f_t \cdot f_c \cdot \mathrm{LSC}_\ell,$$

where LSC_ℓ is the LSC value of the locally corresponding segment of the pbg-structure, which is obtainable via cross-referencing to the pbg-structure and forward-referencing in the pbg-structure, all by the c pointers.
The daughter axes collected in one cluster will be translated such that they get a common origin (which was determined as the cluster position by the clustering algorithm). Hence, no segment between two branch nodes (or between a branch node and the end of an axis, which is considered equivalently as a branch node by the clustering algorithm in this case) will be shorter than the above-expressed local minimal distance. These segments become the units of the sbg-structure after splitting.

If a complete axis A is shorter than the local minimal distance $d_{\min}$ at its origin, three cases have to be distinguished (let ℓ be the length of A):

(i) A has no daughters and $\ell < d_{\min}/2$.
The axis A will be completely eliminated in the sbg-structure.

(ii) A has no daughters and $d_{\min}/2 \le \ell < d_{\min}$.
The axis A will be inflated to the length $d_{\min}$. Other variables except the length and the end position will not be changed.

(iii) A has itself daughter axes.
In this case, A (as a "structural axis") will remain unchanged, despite of its too short length.

All three cases are protocolled and counted.

The structure resulting from this step will be referred to as the *secondary base grid structure* (sbg-structure), because it is derived from the pbg-structure. In general, it will be topologically different from the pbg-structure: Branching nodes have been melted; possibly even some axes are lost (case (i) above).
The following steps resemble the steps 6–8:

12. Relabelling of the sbg units

(Again, the indices i will be arranged in reverse depth-first order, following the same restriction for the prolonging segments as described in step 6 above.)

13. Calculation of LSC values and of statistical data concerning the pbg- and sbg-structure

14. Writing of the sbg-file.

Afterwards, another sbg-structure with other f_c and f_t values can be created, starting with step 10 again and using the same pbg-structure. The whole transformation process with all 14 steps is visualized in Fig. 35.

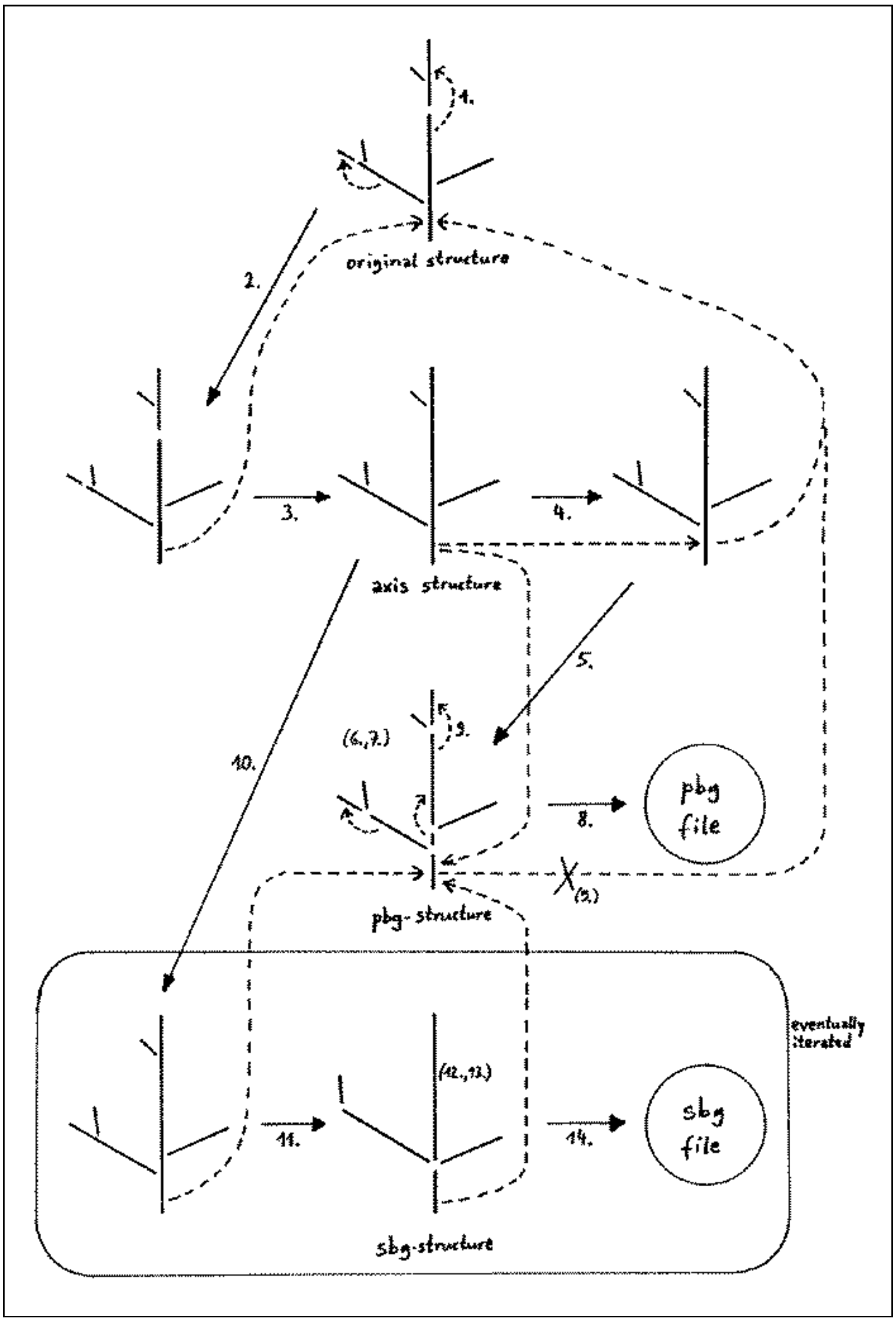


Fig. 35: The transformation steps of the interface to HYDRA

pbg format (filename suffix .pbg)

A pbg-file consists of two parts, a *head* and a *main part*. The head contains general informations about the structure, whereas the main part consists of lines each of which describes a segment (elementary unit) of the pbg-structure.

The head has the following format:

```
(1 empty line)
filename, generated by GROGRA, date and time
(4 empty lines)
————————————————————————————————————————————————
Areas obtained independently from length
Length/LSC           Mean:...   Mean abs. dev.:...   Std. dev.: ...
Proper whorl sizes    Mean:...   Mean abs. dev.:...   Std. dev.: ...
————————————————————————————————————————————————
Ratio smaller than 50 percent of all ratios: ...
Ratio smaller than 80 percent of all ratios: ...
Ratio smaller than 90 percent of all ratios: ...
Ratio smaller than 95 percent of all ratios: ...
Smallest ratio:                              ...
————————————————————————————————————————————————
(4 empty lines)
KARTE: filename                  SI: ...
————————————————————————————————————————————————
d
```

The term "ratio" refers to length / LSC, "Mean abs. dev." means "mean absolute deviation", and "Std. dev." means "standard deviation". "SI" stands for "soil index" and denotes the number of segments (units) of the structure, including 3 artificial segments added at the basis, which are also included in the main part of the file.

A line of the main part has the format

$$i \;\; / \;\; i_{d_1} \;\; i_{d_2} \;\; \cdots \;\; i_{d_k} \;\; 0 \;\; / \;\; \pm\, i_m \quad \ell \quad d_b \quad n$$

i is the unit identifier (i.e. an integer between 1 and the soil index), $i_{d_1} \cdots i_{d_k}$ the list of daughter unit identifiers in ascending order, i_m the mother unit identifier. i_m gets a negative sign if the unit is not axis-prolonging. ($i_m = 0$ for the basis unit.) ℓ is the length (in m), d_b the diameter (in cm) and n the needle surface (in m^2) of the unit. The entries are separated by blanks.

sbg format (filename suffix .sbg)

This format is similar to the pbg-format. Some informations refer to the "source", i.e. to the pbg-structure from which the sbg-structure was obtained. The head is formatted as follows:

(1 empty line)
filename, related source *pbg-filename, date and time*
(4 empty lines)

```
Ratio used as creation factor:  f_c
Tolerance factor:               f_t
Areas obtained independently from length
Total length                    PBG: ... m   SBG: ... m
Length change (PBG-SBG)/PBG:     ...
Manipulations:                 Translat. ..., inflat. ..., elim. ...
Proper whorl sizes      Mean:...  Mean abs. dev.:...  Std. dev.: ...
--------------------------------------------------------------------
SBG |Length-CD|/CD Mean:...   Mean abs. dev.:...   Std. dev.: ...
--------------------------------------------------------------------
```

(4 empty lines)
KARTE: *filename* SI: ...

---

d

"Total length" refers to the sum of all segment lengths, "CD" stands for "criterion distance". "Translations", "inflations" and "eliminations" mean the different axis manipulations done during sbg creation which are counted. A translation occurs for each daughter axis which is translated into a cluster center, whereas "inflations" and "eliminations" refer to the cases (ii) and (i) of short axis handling (p. 122). The other abbreviations are the same as in the pbg format.

A line of the main part has the same format as in a pbg-file (see above) with the exception that in all lines (except the last 3, artificially added lines) the criterion distance (in m) is appended as an additional entry. The "criterion distance" is obtained as LSC $\cdot$ f_c for all units of the sbg-structure.

Stem analysis format (filename suffix .bol) and interface to GROBOL

The stem analysis is started via the corresponding item in the "Analyze the actual structure"-submenu (see p. 101). It takes all developmental steps into account, but gathers only informations about stem units, i.e. units with branching order 0. It is assumed that these units form a single, connected axis.
The results are written to a bol-file, consisting of several lines, each of which having the format

$$t \;\; h \;\; r_h$$

t is an integer, the number of the developmental step. h and r_h are real numbers given in exponential notation (and measured in the same length unit, usually in mm). h is the height where the stem radius r_h is determined. For each fixed t, h will ascend from 0 to the height of the highest stem unit tip, thus giving several

lines in the file. A possible non-zero z-coordinate of the bottom position P of the basal unit ("null niveau") will be subtracted automatically from all h values.
r_h is determined at the stem basis ($h = 0$), at the stem tip ($r_h = 0$) and at the middle position of each elementary unit.

Files in bol-format can be read by the program GROBOL written by D. Lanwert. This software offers several possibilities to show and animate the interpolated stem form from different points of view and with magnified radial extension. Moreover, the stem shape can be compared with empirically obtained stem data. GROBOL is also able to write lsy-files — e.g. from measured tree boles — which can again be interpreted by GROGRA in the usual manner.
GROBOL runs on the Silicon Graphics workstation only.

Cubic grid analysis format (filename suffix .kub) and interface to 3dCLIP

The data transfer to the 3D-climate and physiology-model (which is still in its developmental phase) is done by kub-files and refers to a grid consisting of cubic cells which is laid over the whole structure. The side length of the cells (grid resolution) and the orientation of the main directions of the grid are to be specified by the user after the menu item "grid with cubic cells" or "grid with cubic cells, simplified" is activated (see Section 5.2, p. 100). The orientation of the orthonormal system determining the grid's main axes can be chosen arbitrarily (see Fig. 36 for an illustration in two dimensions).

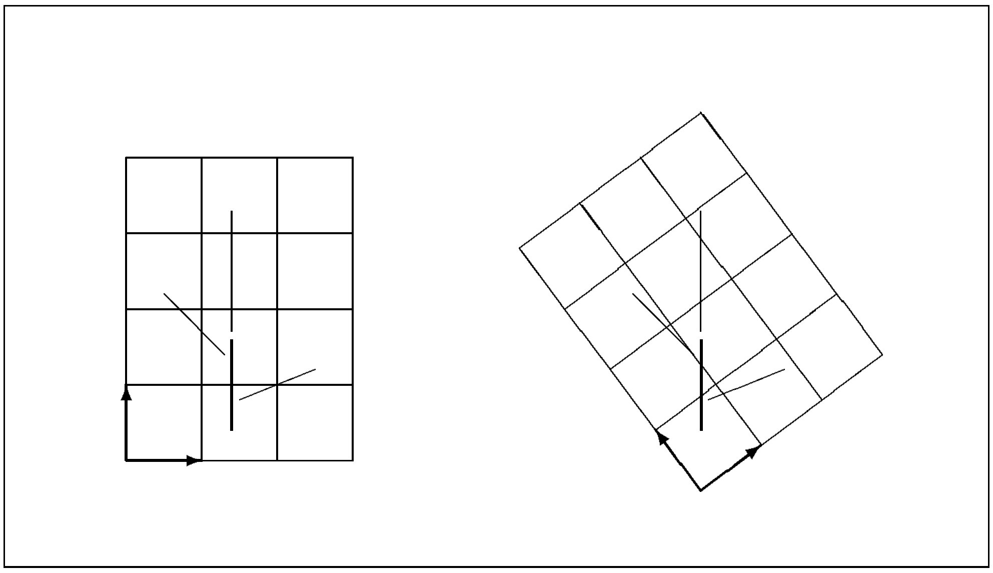


Fig. 36: Two possible orientations of the imposed grid

The cubic grid analysis sums up for each cell the lengths, volumes, needle surface areas (in the standard version splitted up into 8 age classes), and directions of all elementary units whose middle axis overlaps with that cell (Fig. 37).

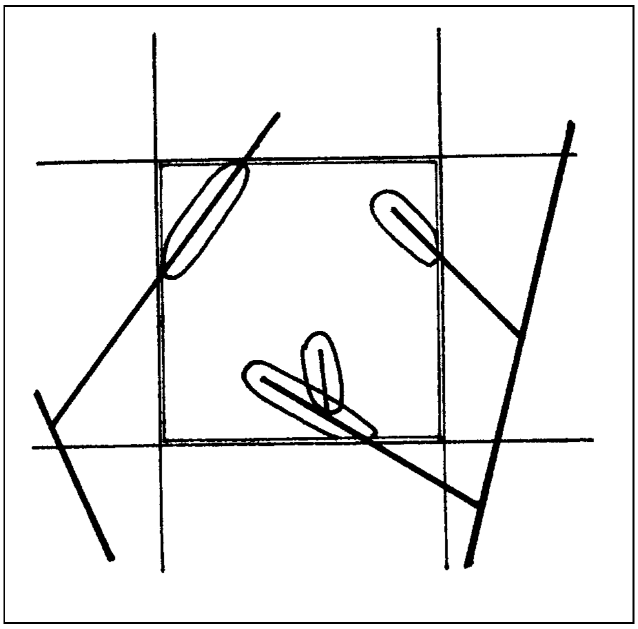


Fig. 37: Intersections of structural units with a grid cell

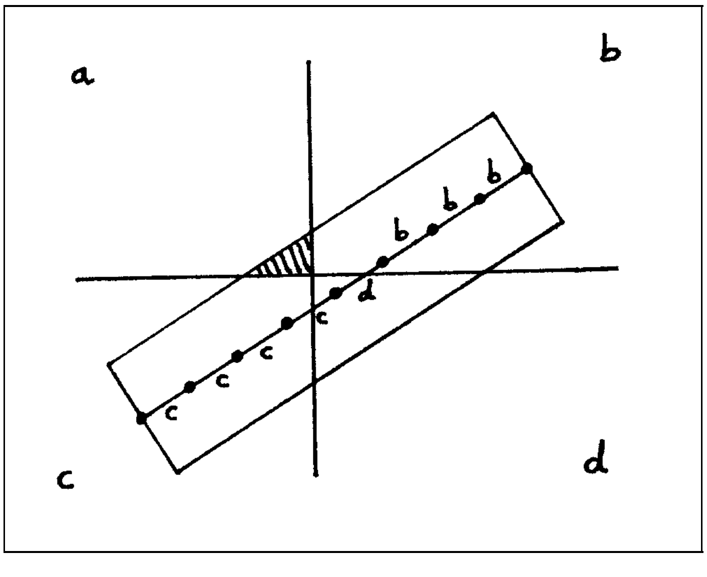


Fig. 38: Four cells, denoted *a*, *b*, *c*, *d*, and an elementary unit divided into parts making contributions to different cells

However, the overlapping parts of the elementary unit axes are not determined by analytical calculation of intersection points, but are approximated with the help of an equidistant discretization of each elementary unit. An elementary unit middle axis is divided into parts with equal length $\Delta\ell$ which is of the magnitude $s/10$, s being the sidelength of the cells. Each part with length $\Delta\ell$ contributes its volume, needle area etc. to that cell to which its left endpoint belongs (Fig. 38).

Note that the diameter of the elementary unit is not taken into account by this algorithm. Hence, cell a will get no contribution from the unit shown in Fig. 38, despite of its non-empty intersection with that unit.

A file of the format kub consists of a head and a main part, the head containing general informations about the grid and the main part consisting of lines each of which describes a cell of the grid. The order how the cell descriptions are arranged is given by the reciprocal lexicographic order of their midpoint coordinate triplets in the orthonormal coordinate system defining the grid (i.e., first the x-coordinate in the grid is increased step by step, then the y-coordinate, then the z-coordinate; see Fig. 39). All cells in a rectangular part of space are listed, including empty ones.

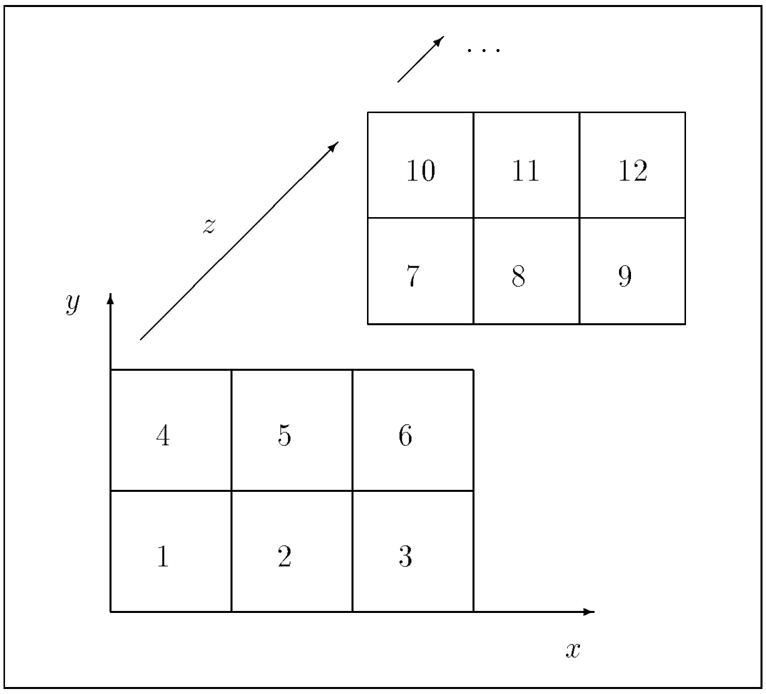


Fig. 39: Enumeration of cubic cells

The head of a kub-file consists of two lines containing the following numbers, separated from each other by one or more blanks:

k_t k_x k_y k_z s L_x L_y L_z o_x o_y o_z
a_x a_y a_z b_x b_y b_z c_x c_y c_z m_x m_y

Our abbreviations have the following meanings:

k_t total number of all cells ($k_t = k_x \cdot k_y \cdot k_z$)

k_x number of cells along the x-direction of the grid (analogously k_y, k_z)

s side length of a cell (the same side length in each of the three directions, because the cell is indeed a cube)

$L_x = s \cdot k_x$, total extension of the grid in x-direction (analogously L_y, L_z)

o_x x-coordinate of the origin of the grid in the old coordinate system used for describing the structure ("world coordinates"). All cells lie in the first (i.e. positive) octant of the new (grid) coordinate system. (Analogously o_y, o_z.)

a_x x-coordinate of the first basis vector of the grid (x-direction of the grid orthonormal system). Analogously: a_y, a_z and the coordinate triplets of the second and third basis vectors b, c.

m_x, m_y coordinates (in the world coordinate system) of a "central axis" of the analyzed structure, automatically assumed to be parallel to the z-axis. The axis is determined by the bottom position of the basal stem unit. Exposition angles will refer to this axis.

The entries in a line of the main part of a kub-file (standard version) are the following (again separated by blanks):

$$\mu_x \; \mu_y \; \mu_z \; \mathsf{V} \; v \; n_0 \; n_1 \; n_2 \; n_3 \; n_4 \; n_5 \; n_6 \; n_7 \; d_x \; d_y \; d_z \; \ell \; \gamma$$

Here, (μ_x, μ_y, μ_z) is the coordinate triplet (in world coordinates) of the midpoint of the cell which is described in that line. V means just the upper case letter V (a marker), whereas v is the summed volume of all unit parts in the cell. n_a is the sum of all areas of needle surfaces (i.e., n-values) of age a, the age of a unit being determined as $a = g_{\max} - g$, where g is the generative distance of a unit. (n_7 collects all areas of needles of age ≥ 7.) Age 0 refers to the youngest needles, etc.
(d_x, d_y, d_z) is a vector which is calculated as the sum of all directions of elementary unit parts falling in the described cell. It is something like a "weighted mean direction" of all shoots in that cell. In the case of an empty cell, all three values will be 0.
ℓ is the length sum of all unit parts ascribed to the cell, and γ is the exposition angle of the cell midpoint with respect to the above-mentioned "central axis", i.e. an angle in cylinder coordinates, given in degrees. More exactly, it is the angle between the vectors ($\mu_x - m_x$, $\mu_y - m_y$) and (1, 0) in the xy-plane (world coordinates).

In the simplified version (menu item "grid with cubic cells, simplified"), the head of the kub-file will be the same, whereas the line format of the main part is now

$$\mu_x \quad \mu_y \quad \mu_z \quad \mathsf{V} \quad v \quad n \quad \ell \quad \gamma$$

where n collects the needle surface areas of all age classes.

5.6 An overview of the modular program structure

The source code of GROGRA is currently distributed on 14 program modules, specialized on different tasks.

spezial.c	mutual emulation of PC-/SGI-specific functions
mouse.c	mouse control
drucker.c	printer control
mmen.c	menu display and file selection
vekt3.c	linear algebra in $\mathbb{R}^3$ and some functions concerning cubic cell grids
zufall.c	random variable generation and distributions
lverzw.c	initialization, writing, and I/O of elementary units and branching structures
lgraph.c	graphical display of structures
lmayer.c	structure generation from grammar files
lmethod.c	functions and methods called in grammars
ltrans.c	transformation for HYDRA
lanaly.c	analysis of structures, including stem analysis (interface to GROBOL) and cubic grid analysis (interface to 3dCLIP)
linter.c	reading and saving of structures (including dtd parsing)
lgrogra.c	menu structure of the program, explanation part, main initialization and control of grammar interpretation

Furthermore, the file lexpla.msg, containing text to be displayed in the explanation part, is part of GROGRA.

The microcomputer-version was compiled in the memory model HUGE (cf. [15]).

Chapter 6

Examples

The following examples shall indicate some possibilities of GROGRA, but do by far not exhaust the range of potential applications. More elaborated examples, designed for special purposes in forestry or ecosystem research, will be subject of further publications and have not to be expected here in the context of a quite general software documentation.
All example pictures were generated with the microcomputer version of GROGRA 2.4.

In all grammar listings, the lines are numbered. The numbering is only for reference purposes and does not appear in the files. — In all cases, the start word for the grammar application is $*$.

6.1 koch.lsy

The following non-sensitive, non-stochastic growth grammar generates the fractal VON KOCH-curve.

```
1   \angle 60,
2   * # RU90 a F,
3   a # a L*0.333,
4   F # F - F + + F - F
```

The first, declarative line specifies the angle for the $+$ and $-$ symbols. That is, $+$ stands for RU60 and $-$ for RU-60 in this L-system. The replacement process begins with the rule in line 2, which is applied to the start symbol $*$. Note that the initial direction of the turtle is "straight upward" (in z-direction), such that the RU90-command is necessary to enforce a horizontal movement in the first step. The structure generated from RU90 a F consists of just one horizontal line (not shown here). To the string RU90 a F, the rules 3 and 4 are applied in the next step. Fig. 40 shows the 6 next developmental steps in the graphical display obtainable simply under the option "side view".

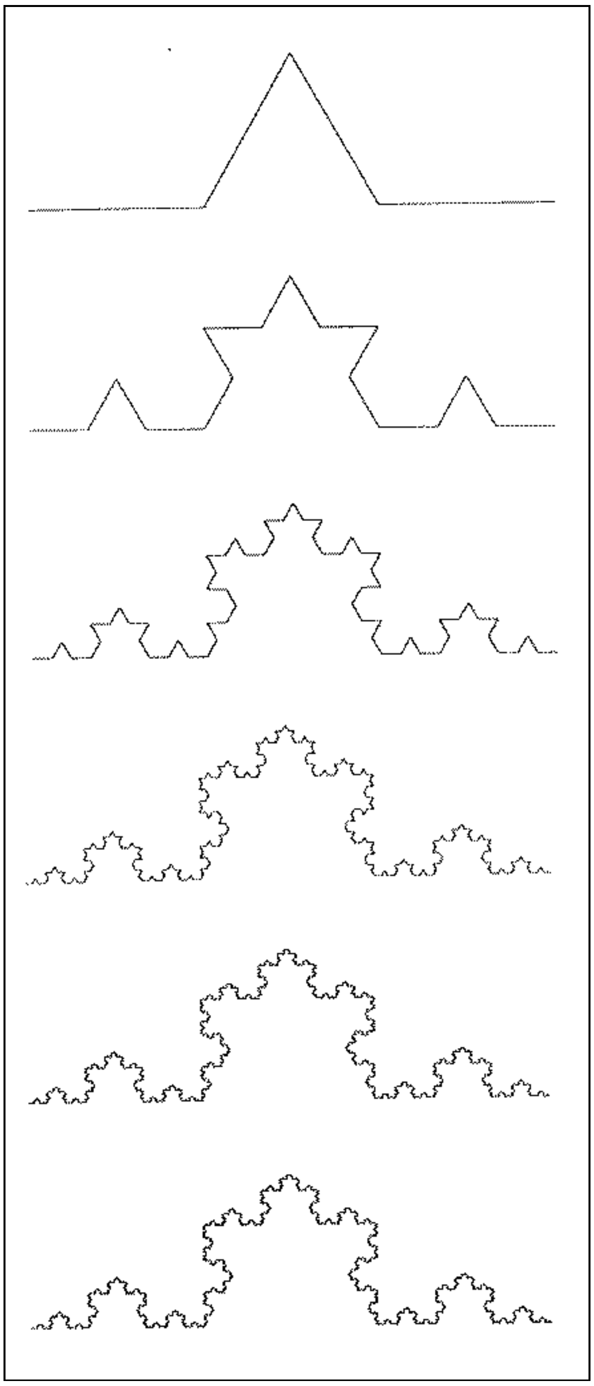


Fig. 40: koch.lsy, steps 2–7

6.2 mchange.lsy

This example demonstrates stochastic variation in its simplest form. The grammar generates six "plants", arranged in a line, and each following the same stochastic developmental rule.

```
1    *  # P12 [ a ] &5 < RU90 f*3 RU-90  [ a ] >,
2    a  # F [ RU60 b ] [ RU-60 b ] a ?0.9,
3    a  # P1 L*0.5 a' ?0.1,
4   a'  # F RH180 [ RU75 b' ] a',
5    b  # P2 F,
6   b'  # P14 F
```

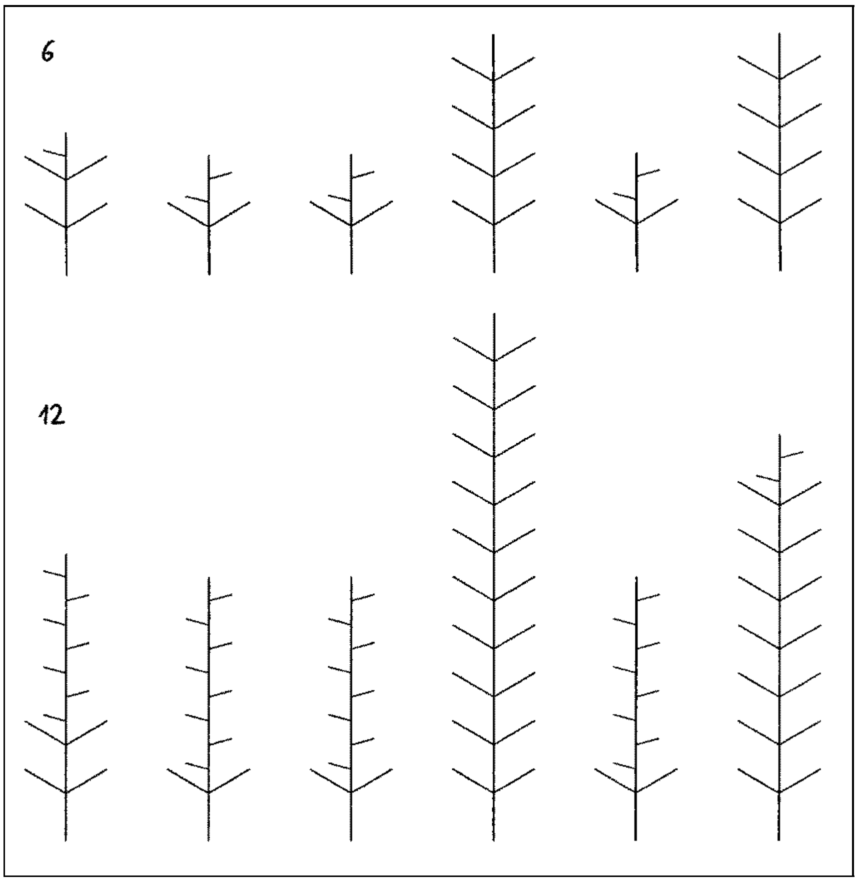


Fig. 41: mchange.lsy, steps 6 and 12

Rule 1 sets the "seeds" for the six plants in a spacing corresponding to 3 times the length of a growth unit ($f*3$). The repetition operator &5 iterates this spacing, along with the necessary re-orientations of the turtle, 5 times.

The plant, consisting of just one axis and short lateral shoots, has two "growth modes", the first one symbolized by a and b, and the second one by a' and b'. Once the first (default) growth mode is left by an application of rule 3, transforming a into a', the plant will in the next step continue to grow in the second growth mode and will never return to the first, i.e. rules 4 and 6 are then applied, replacing rules 2 and 5. The two growth modes are distinguished by different lengths, branching angles, colours and phyllotactic arrangements (two opposite side shoots in rule 2, alternating single side shoots in rule 4 — note the effect of the $RH180$ command in alternating the orientation of the subsequent b'-part). As the transition from the first to the second growth mode occurs in a non-deterministic manner (rule 3 having probability 1/10), the shape of the plant cannot be predicted. Fig. 41 shows the steps 6 and 12 of a sample development.

6.3 examp.lsy

This example shows a rather simple 3-dimensional model plant with above-ground and below-ground parts, demonstrating several types of differentiation and also de-differentiation of meristems, here in the form of total reiterations from roots.

```
 1  \var x0 uniform 0 360,
 2  \var x1 normal 0 15,
 3  \var x2 uniform -10 0,
 4  \var x3 uniform -25 25,
 5  * # [ P14 t0 ] P4 &6 < [ RH(x0) P5 bl ] > L*0.6 r0,
 6  t0 # dt F D RH137.5 [ RL80 L*0.5 P2 k(1) s1 ]
       [ RH180 RL80 L*0.5 P2 k(1) s1 ] t0,
 7  s1 # ds F D [ RH25 RU60 $ L*0.7 s2 ]
       [ RH-25 RU-60 $ L*0.7 s2 ] s1,
 8  s2 # ds F D,
 9  dt # dt D+3,
10  ds # ds D+2,
11  (t < 6) k(t) # k(t+1),
12  (t = 6) k(t) # %,
13  r0 # RG RH(x0) RU(x1) dt F D
       [ L*1.1 k(1) P15 r1 ] [ L*1.1 k(1) P15 r1' ] r0,
14  bl # rl,
15  rl # RG RL90 RL(x2) RU(x3) ds F D a1 rl,
r1 # r2,
r1' # r2',
r2 # RG RL70 RL(x2) RU(x3) ds F D r2,
r2' # RG RH180 RL70 RL(x2) RU(x3) ds F D r2,
a1 # a2,
a2 # a3,
a3 # [ RG RU180 * ] ?0.2,
a3 # z ?0.8
```

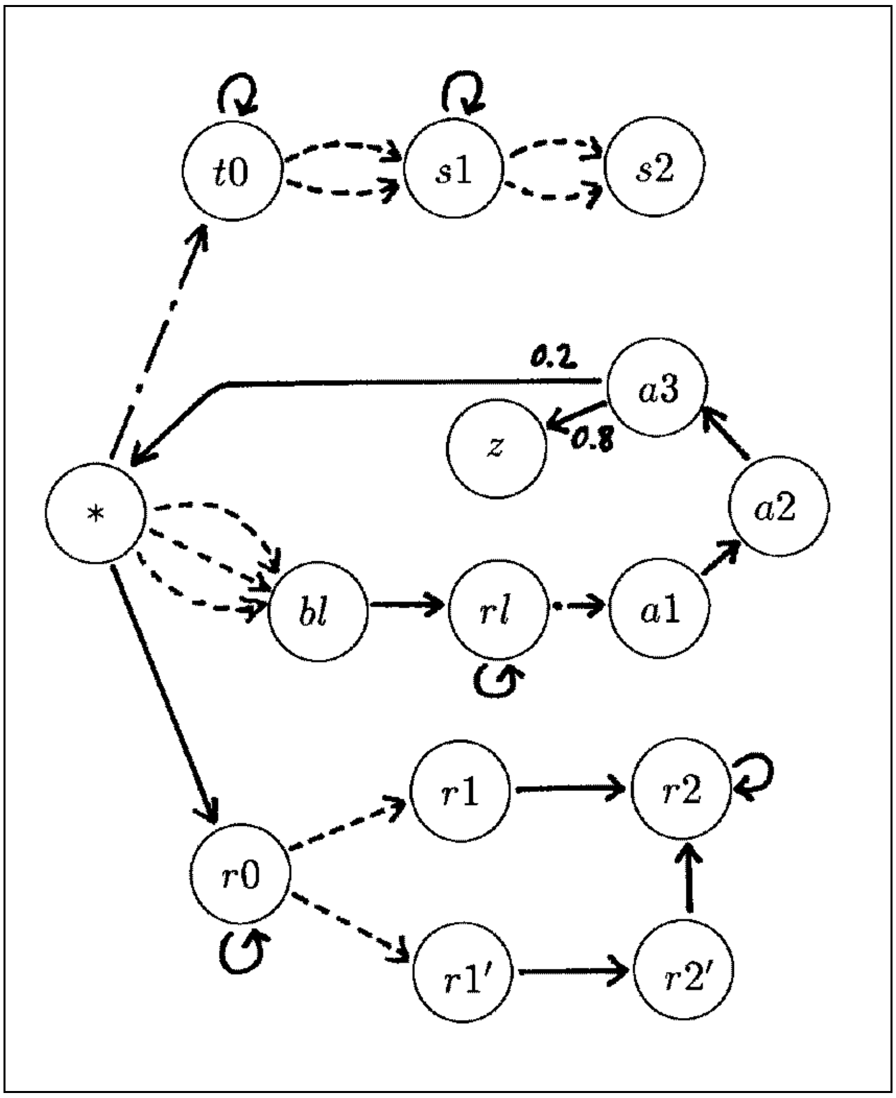


Fig. 42: Finite automaton, representing
meristem state transitions

Here, $t0$ stands for the terminal bud of the stem, $s1$ for a first-order-, $s2$ for a second-order side axis bud, $r0$ for the terminal meristem of the central root, $r1$ and $r1'$ (resp. $r2$ and $r2'$) for the two (symmetrically initiated) first-order side roots emerging from the central root after a 2-step delay. Furthermore, we have (near-to-ground) lateral root meristems rl (their initial buds denoted by bl) and adventitious buds $a1$ (later phases: $a2$, $a3$), initiated in the lateral roots and sprouting with probability 0.2 (rule 22). z stands for a "dead meristem". Fig. 42 illustrates the differentiation process which these meristems can undergo.

$t0$, $s1$ and $s2$ produce the above-ground part of the plant, bl and rl the long lateral roots, and $r0$, $r1$, $r1'$, $r2$, $r2'$ the root system emerging from the central root. Note that in rule 22, the start symbol $*$ appears again, giving rise to a whole new tree from the adventitious bud $a3$ (total reiteration).

Rules 11 and 12 trigger the shedding of first-order branches (after 6 steps). The stem and branch thickening is done by rules 9 and 10. (Note that the construction *dt F D* or *ds F D*, found in rules 6, 7, 8, 13, 15, 18 and 19, offers an alternative way to restrict the effect of the *D* command to a local setting, when combined with rules 9 / 10. A *Dl* command with appropriately incremented parameter would have had the same effect.)

The above-ground side branches stand in spiral phyllotaxy (*RH*137.5 in rule 6), whereas the side roots of the central root are arranged in two lines (rule 13). Some irregularity is introduced into the system by the stochastic variables $x0$ – $x3$, influencing rotation angles. Fig. 43 shows the steps 3, 6, 9 and 11 of a sample development from this grammar. The view direction is 20 degrees from above here. Total reiteration is starting in step 9.

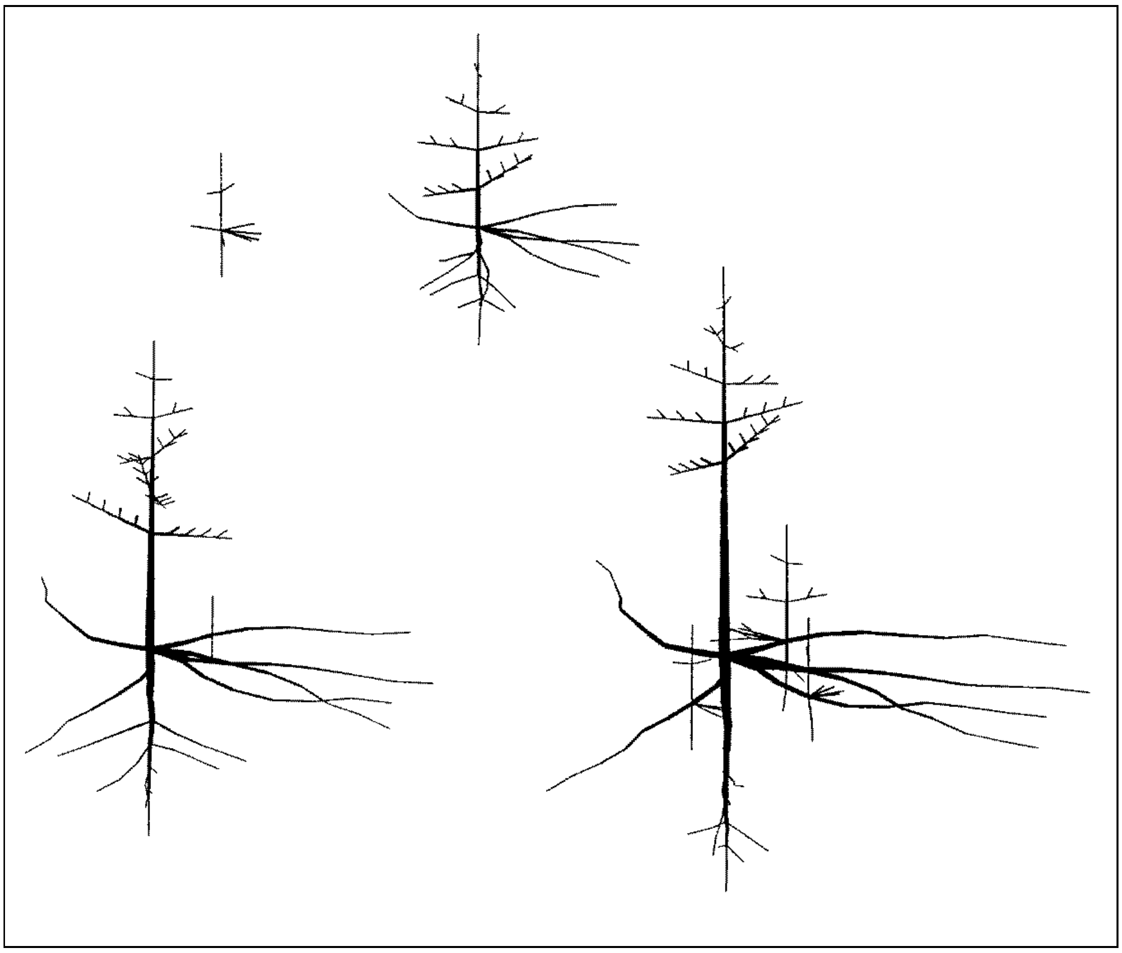


Fig. 43: examp.lsy, steps 3, 6, 9 and 11

6.4 ficht5.lsy

This growth grammar is a first version of spruce growth modelling (*Picea abies* (L.) Karst.), suited for the above-ground part of young spruce trees (3–8 years) and giving a rather simplified architectural model. (See Example 6.12 below for a more refined spruce model.) It contains 18 rules and 12 variables. Each developmental step corresponds to one year. The fundamentals on morphology underlying this grammar were taken from [47]. However, sylleptic shoots were not included here.

```
 1  \angle 45,
 2  \var x0 uniform 0 360,
 3  \var x1 uniform 0.2 0.85,
 4  \var x2 uniform 0.85 1.,
 5  \var x3 table 0 50,
 6  \var x4 normal 0 30,
 7  \var x5 normal 0 10,
 8  \var x6 normal 0 5,
 9  \var x7 normal 0 5,
10  \var a array 5 1 30,
11  \var k index,
12  \var n distribution 0 0 0 0 0 0.1 0.4 0.3 0.2 0,
13  * # P10 D1 T,
14  T # i(2.2,0) RH(x0) &3 < [ @(x1) RH(k*120+x4) RL(x3+x6) A(1) L*0.4 m1 ] >
      RH(x0) &n < [ @(x2) RH(k*360/n_ + x5) RL(x3+x6) A(1) L*0.65 s1 ] > T,
15  s1 # i(1.3,0) [ @(x2) RH15 + RU(x7) $ L*0.7 s2 ]
      [ @(x2) RH-15 - RU(x7) $ L*0.7 s2 ] G(1) s1,
16  m1 # i(0.8,0) [ @(x2) RH15 + RU(x7) $ L*0.7 m2 ]
      [ @(x2) RH-15 - RU(x7) $ L*0.7 m2 ] H1 m1,
17  s2 # i(1.3,0) [ @(x2) RH10 + $ L*0.7 s3 ] [ @(x2) RH-10 - $ L*0.7 s3 ] s2,
18  m2 # i(0.8,0) [ @(x2) RH10 + $ L*0.7 m3 ] [ @(x2) RH-10 - $ L*0.7 m3 ] m2,
19  s3 # i(1.3,0) L*0.7 s4,
20  m3 # i(0.8,0) L*0.7 m4,
21  i(s,t) # i(s,t+1),
22  A(t) # A(t+1),
23  G(t) # G(t+1),
24  H1 # H2,
25  H2 # RL2 H3,
26  H3 # RL2 H4,
27  H4 # RL4,
28  i(s,t) ## Dl+(s*t) F,
29  A(t) ## RL(a(t)),
30  (t >= 3) G(t) ## RL4
```

T stands for the apical meristem of the trunk, $s1$ for that of first-order subapical side shoots, $m1$ for that of first-order medial side shoots, and so on. Only branching orders 0–3 are taken into account. Variable $x1$, ranging from 0.2 to 0.85, indicates the "medial part" of a shoot (i.e. the region where the medial side branches emerge), $x2$ (from 0.85 to 1) the "subapical part" (the shoot tip having relative position 1 for the @-command). The initial branching angle at the stem is normally distributed around 50° (with variance 5°, line 8), but it is incremented by rules in lines 22 and 29 according to the supplementary file ficht5.a05 to which line 10 refers. The angle increments listed in this one-dimensional array are

0 15 25 32 37 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
40 40 40 40 40 40 40 40 40 40,

giving in the final state a branching angle of about 90 $(= 50 + 40)$ degrees. In the side branch system, however, the branching angle is 45° (line 1). Additionally, the second- and third-order side branches have a slight tendency upwards ($RH15$ / $RH{-}15$, resp. $RH10$ in lines 15–18). When getting elongated, the side branches perform some bending, controlled by the rules in lines 23–27 and 30.

Lines 11, 12 and 14 demonstrate the use of the repetition operator & in connection with a stochastically distributed repetition number n, specifying the number of subapical main branches in each pseudo-whorl at the stem. (Note the necessary use of the memory operator _ for n in the rotation angle specification $RH(k * 360/n\_ + x5)$, which enables the centrally symmetric arrangement of a variable number of lateral axes.) The number of medial side branches at the stem is kept fixed (to 3) for the sake of simplicity, as are the higher-order branching factors.

It is important to note that the F command appears only in the second-phase rule in line 28. The parameters s and t contain all information how the symbol $i(s,\ t)$ — i standing for “internode”, but not in a botanical sense — is to be interpreted. t serves as a clock (rule 21), and s stands for the annual branch thickening (here, 2.2 mm for the trunk (line 14), 1.3 mm for first-order subapical branches etc., see lines 14–20). In contrast to Example 6.3 above, a local D-command is used here. The length information comes from length contraction factors in lines 14–18 ($L * 0.4$ etc.). The initial length is 100 length units (millimeters), the default value of GROGRA for that variable.
Fig. 44 shows the developmental steps 3 to 7 generated from this system in side view, and in Fig. 45 the model tree of step 7 is additionally shown from above.

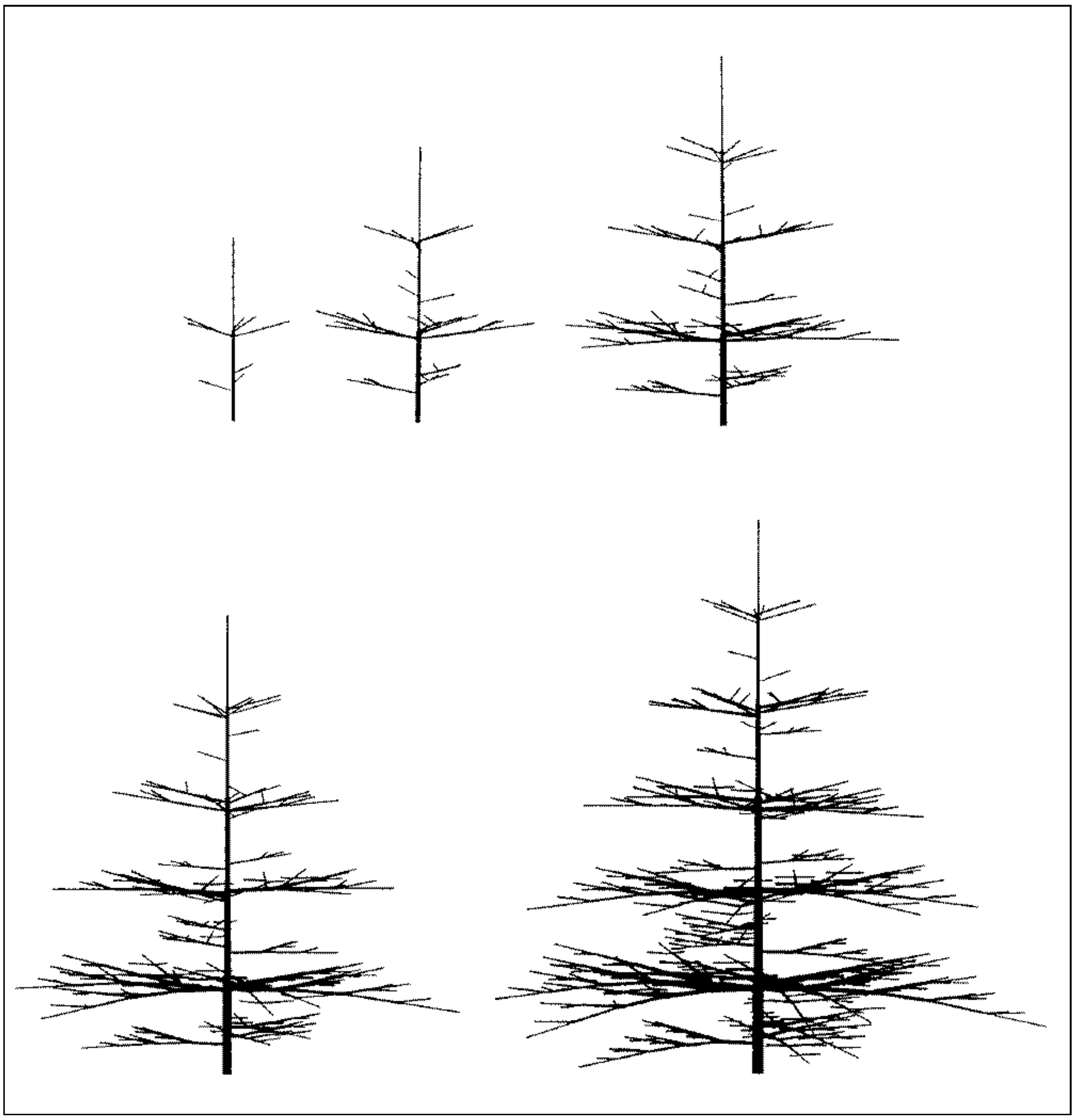


Fig. 44: ficht5.lsy, steps 3 – 7

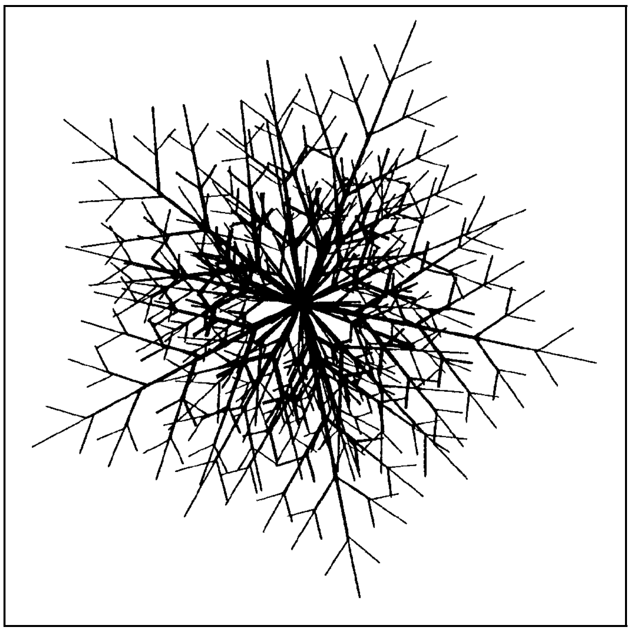


Fig. 45: ficht5.lsy, step 7 (view from above)

6.5 shoot.lsy

This stochastic, two-phase growth grammar, which is not commented in full detail here, is based on empirical data of the morphology of a 12-years-old subapical first-order side branch of a spruce tree (*Picea abies* (L.) Karst.), consisting of about 3000 shoots, which was encoded in dtd-format and analyzed by GROGRA. [*] [0] The grammar generates a branch, not a whole tree. Unlike in the previous example, the branching pattern is not fixed here. Number and strength of side branches depend on a "vitality parameter" which is directly correlated to shoot length. For branching order ("o" in the grammar), only the cases o = 1 and o > 1 are distinguished, the behaviour of a shoot being primarily controlled by its vitality ("v"). The start vitality, which is at the same time the length of the first shoot which is generated in step 2, is requested as input from the user (line 17). — The grammar has the additional peculiarity that only each second developmental step generates a non-empty structure. (This is due to the clustering of lateral buds controlled by the rules in line 22 and 25–27.) Hence, step 2 corresponds to a 1-year-old shoot, step 4 to a 2-years-old shoot, etc.

[0] * Thanks are due to Dipl.-Forstw. M. Wedler for his essential part in the laborious work of discretization of the morphometric data from this spruce branch.

```
\const m0 13.84,
\const m1 1.14,
\const m2 9.51,
\const m3 1.37,
\const slg 0.021,
\const wki 54,
\const wkf 7.7,
\const ts1 14,
\const tm1 8,
\const t0 7,
\var epsw normal 0 10,
\var epsl normal 0 5,
\var epsf normal 2 0.5,
\var start register 0,
\var sw distribution 0.5 0 0.5 0,
\var i index,
\ask l0 start vitality ? ,
\register 1 5,
* # P2 k(start, 1, 1),
(o > 1) k(v, o, p) # n(0, v) ds(v, o, p) L(v+epsl) Pl14 s(0, 90, o, p)
   ?(1.06-0.0105*v),
(o == 1) k(v, o, p) # n(0, v) ds(v, o, p) L(v+epsl) Pl14 s(0, 90, o, p)
   ?(0.3*(1.06-0.0105*v)),
k(v, o, p) # n(0, v) ds(v, o, p) L(v+epsl) Pl14 s(0, 90, o, p)
   &((-2.44)+0.052*v) < [ @(1/(1.08425+0.2469*i))
   c(i, v, o, 1/(1.08425+0.2469*i)) ] > cs(v, o, p),
Pl14 # ,
cs(v, o, p) # k(0.4*exp(1.15*log(v)), o, p),
(i=0) c(i, v, o, p) # [ f(epsf*(sw-1)) RU(wki+epsw) k(0.674*v, o+1, p) ]
   [ f(2*(1-sw_)) RU(-wki+epsw) k(0.674*v, o+1, p) ],
(i>0) c(i, v, o, p) # [ f(epsf*(sw-1)) RU(wki+wkf*log(i+1)+epsw)
   k((0.172+0.546*p)*v, o+1, p) ] ?0.7,
(i>0) c(i, v, o, p) # [ f(epsf*(sw-1)) RU(wki+wkf*log(i+1)+epsw)
   k((0.172+0.546*p)*v, o+1, p) ]
   [ f(epsf*(1-sw_)) RU(-wki-wkf*log(i+1)+epsw)
   k((0.172+0.546*p)*v, o+1, p) ] ?0.3,
(t <= m) s(t, m, o, p) # sz(t, o, p),
(t > m) s(t, m, o, p) # %,
(o == 1 && p > 0.85) sz(t, o, p) # s(t+1, ts1, o, p),
(o == 1 && p <= 0.85) sz(t, o, p) # s(t+1, tm1, o, p),
(o > 1) sz(t, o, p) # s(t+1, t0, o, p),
33  n(t, v)  # nz(t, v),
34  nz(t, v) # n(t+1, v),
35  (t <= 2)      n(t, v) ## N(m0*v^m1 + m2*v^m3),
36  (t > 2 && t <= 7) n(t, v) ## N((m0*v^m1 + m2*v^m3)*(7-t)/5),
37  (t > 7)       n(t, v) ## N0,
38  (o == 1 && p > 0.85)  ds(v, o, p) ## D(0.025 * v),
39  (o == 1 && p <= 0.85) ds(v, o, p) ## D(0.025 * v),
40  (o > 1)     ds(v, o, p) ## D(slg * v),
41  (o == 1 && p > 0.85  && t <= 3) s(t, m, o, p) ## Dl+(2*t) F,
42  (o == 1 && p > 0.85  && t > 3)    s(t, m, o, p) ## Dl+(6) F,
43  (o == 1 && p <= 0.85 && t <= 3) s(t, m, o, p) ## Dl+(0.75*t) F,
44  (o == 1 && p <= 0.85 && t > 3)    s(t, m, o, p) ## Dl+(2.25) F,
45  (o > 1 && t <= 3)     s(t, m, o, p) ## Dl+(0.28*t) F,
46  (o > 1 && t > 3)        s(t, m, o, p) ## Dl+(0.84) F,
```

We comment just a few features of this grammar file.

Line 1–4: $m0, \ldots, m3$ are parameters of a length - needle surface regression (see line 35), deduced from data of [74] and [92].
Line 5: The factor slg relates vitality (length) to initial shoot diameter (line 40).
Line 6–7: wki and wkf are constants in a regression connecting branching angle and (discrete) shoot position (i.e., cluster index i), see line 26 / 27.
Line 8–10: $ts1$, $tm1$ and $t0$ give maximal ages (lines 30–32), the parameter m in line 28 / 29 controlling branch shedding. ($ts1$ is valid for subapical first-order branches, $tm1$ for medial first-order branches, $t0$ for all others.)
Line 13+15: The construction "$epsf\ *\ (sw - 1)$" in lines 25–27 toggles stochastically between values about -2 and $+2$, thus controlling a small, but distinct distance between the (maximally 2) side branches in one and the same cluster.
Lines 14, 17, 19: Register 0 contains nothing but the start vitality which is asked from the user.
Line 19: Initialization of the primary bud. k stands always for a bud, first parameter = vitality, second parameter = order, third parameter = relative position at the mother shoot.
Line 20–21: Control of meristem death (non-deterministic).
Line 22: The central rule controlling growth from a bud and positioning of lateral bud clusters along the new shoot. s stands for "shoot" (parameters: age, maximal age, order, position), c for "cluster of lateral buds" (parameters: discrete position index, vitality, order, relative position of the cluster at the mother shoot), cs for "(trivial) cluster of (one) apical bud".
Line 24: The regression on the r.h.s. controls the shortening of successive growth units of one axis (axis trend).
Line 25–27: A cluster is expanded to one or two lateral buds (in the subapical case — line 25 — always 2). The regressions appearing in the first argument of k control the length contraction from the mother shoot to lateral shoots, depending

on the position at the mother shoot (acrotony).
Line 28–32: Shoot aging and shedding.
Line 33–37: Calculation of needle surface depending on shoot length and age (for needle loss by age see [49]).
Line 38–46: Determination of initial diameter (lines 38–40) and diameter increment per year (lines 41–46).

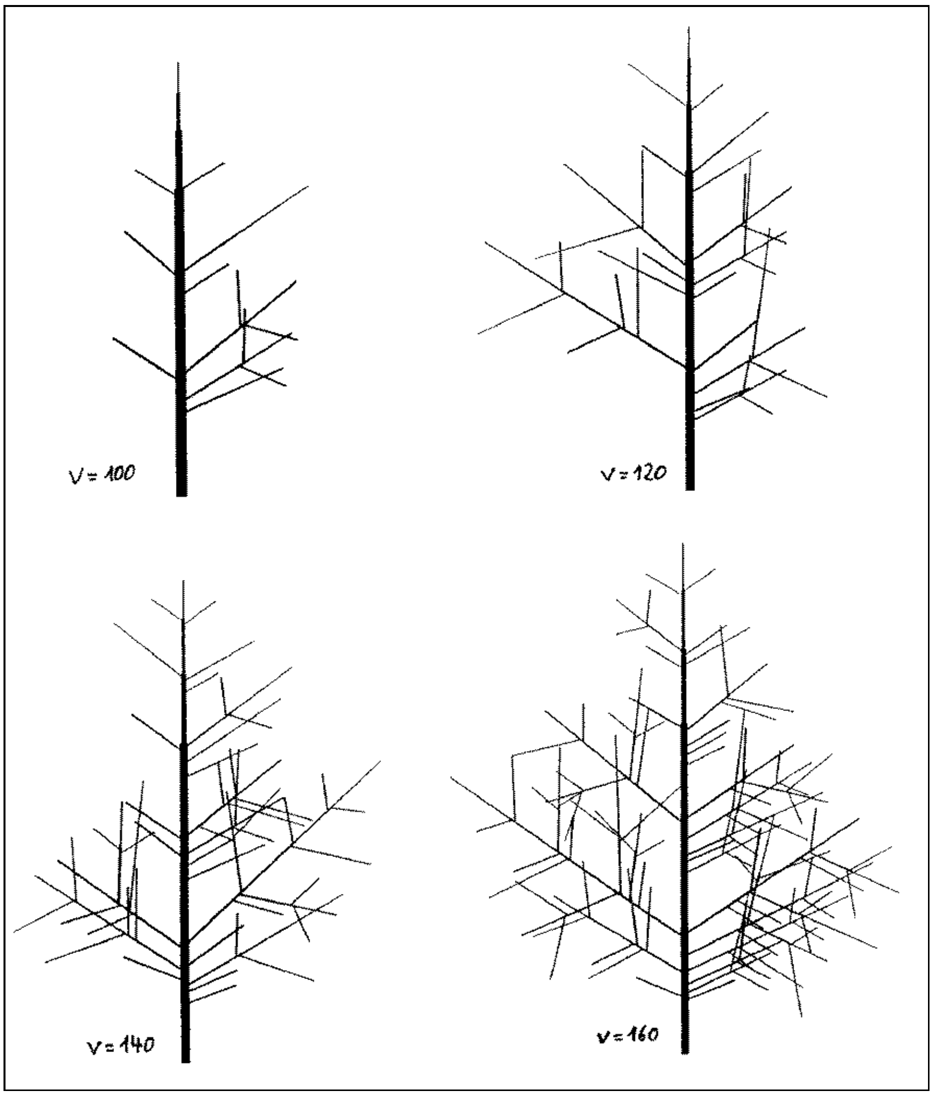


Fig. 46: shoot.lsy, 12 steps, v = 100, 120, 140, 160

Fig. 46 shows 4 different outputs of this grammar, each having the same age of 12 steps (corresponding to 6 years of shoot growth from the initial bud on), but with different start values for the vitality parameter v asked from the user: v = 100, 120, 140 and 160. (The scaling is not the same in the four cases.)

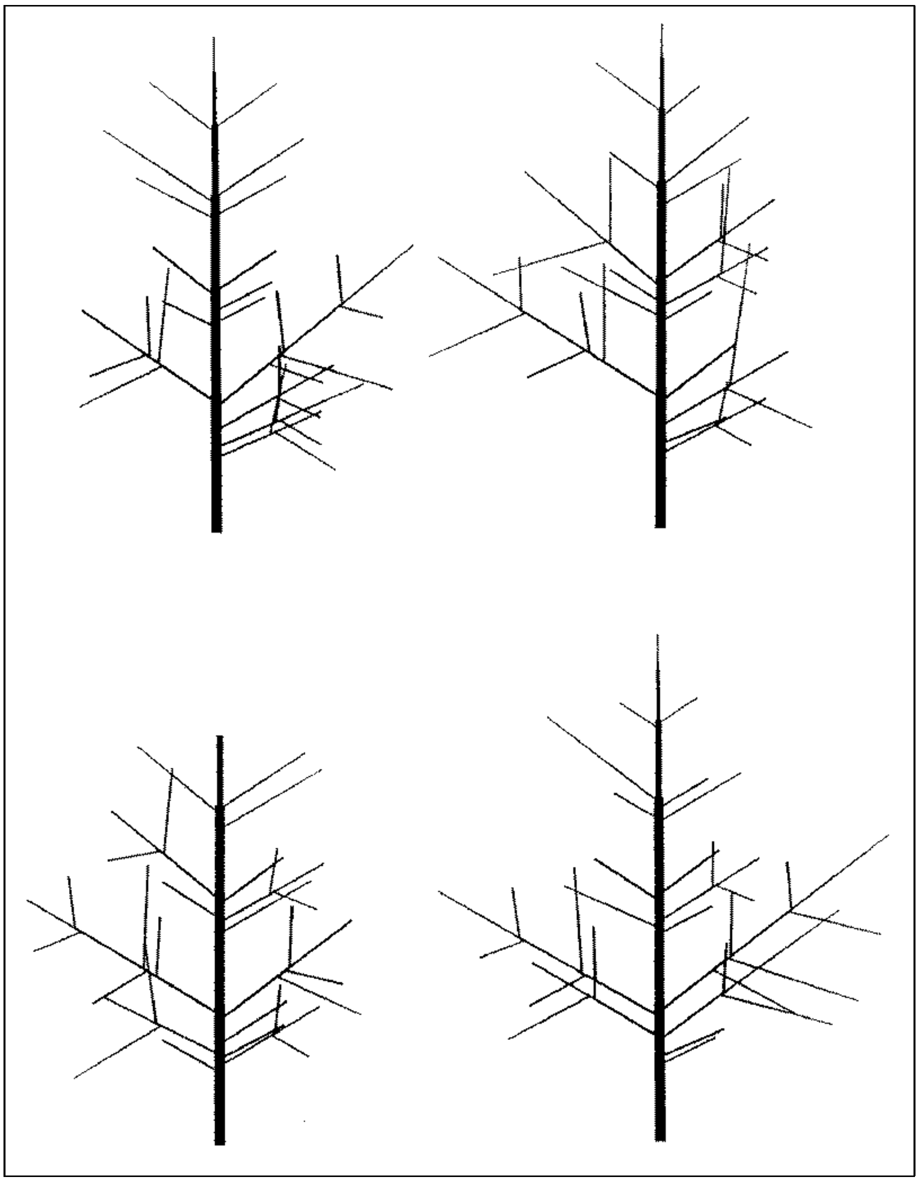


Fig. 47: shoot.lsy, 12 steps, $v = 120$, four different runs

The four results in Fig. 47 are obtained with the same number of steps (again 12) and the same start vitality ($v = 120$ in all four cases) and demonstrate the stochastic variation between shoots of the same type.

6.6 root.lsy

This is another example for a complex differentiation - dedifferentiation - scheme, not designed to fit a real existing species. Several types of root meristems and their mutual transformations are modelled.

```
\var x0 uniform 0 360,
\var x1 uniform 30 70,
\var x2 uniform 40 80,
\var x3 table 1 1 0.9 0.6 1 1 1 0.5 0.4 1,
\var x4 uniform 40 80,
\var x5 uniform -20 20,
\var x6 normal 0 14,
\var x7 uniform -13 0,
\var x8 table 1 1 1 0.8 0.8 0.6 0.6 0.5 0.4,
\var x9 uniform -15 5,
\set D 1,
* # P4 RG [ L*1.3 b2' ] [ RH72 L*1.3 b2' ] [ RH144 L*1.3 b2' ]
    [ RH216 L*1.3 b2' ] [ RH288 L*1.3 b2' ] dc F D c(1),
(t <= 9)  c(t) # P7 RG L*(x3) &5 < [ b1 ] > RH(x0) RU(x6) dc F D c'(t+1),
(t <= 9) c'(t) # P4 RG L*(x3) &5 < [ b1 ] > RH(x0) RU(x6) dc F D c(t+1),
(t = 10) c'(t) # P4 RG L*(x3) &5 < [ b1 ] > RH(x0) RU(x6) dh F D c(11),
(t = 11)  c(t) # P7 RG L*(x3) &5 < [ b1 ] > RH(x0) RU(x6) dd F D c'(12),
(t = 12) c'(t) # P4 RG L*(x3) &5 < [ b1 ] > RH(x0) RU(x6) F D c(13),
(t >= 13) c(t) # z,
dc # D+2.4 dc,
dh # D+1.5 dh,
dd # D+1 dd,
dz # D+2 dz,
b1 # b2 ?0.3,
b1 # z ?0.3,
b2 # b3,
b2' # b3',
b3 # L*1.5 RH(x0) RL90 RL(x7) hw ?0.8,
b3 # L*2 RH(x0) RL(x4) dg ?0.2,
b3' # L*1.5 RH(x5) RL90 RL(x7) hw ?0.95,
b3' # L*1.5 RH(x0) RL(x4) dg ?0.05,
hw # P4 dh F D L*0.9 [ RU50 e1 ] [ RU-50 e1 ] [ j1 ]
    RG RL92 RL(x7) RU(x5) hw ?0.83,
e1 # e2 ?0.3,
e1 # z ?0.3,
e2 # e3,
35  e3 # e4,
36  e4 # hw,
37  dg # P7 dd F D L*0.85 [ j1 ] $ RU(x5) RL(x9) dg ?0.7,
38  hw # RG [ b1 ] RL45 L*0.2 dz F D RL−45 L*4.5 L*(x8) c'(4) ?0.07,
39  j1 # j2 ?0.3,
40  j1 # z ?0.6,
41  j2 # L45 P14 RG tv,
42  tv # RH(x0) RU(x6) F L*0.9 [ a1 ] tv ?0.4,
43  tv # RH(x0) RU(x6) F L*0.8 [ a1 ] [ a1 ] tv ?0.05,
44  tv # RH(x0) RU(x6) F L*0.7 [ a1 ] [ a1 ] [ a1 ] tv ?0.2,
45  a1 # L*0.98 a2,
46  a2 # L*0.98 a3,
47  a3 # L*0.98 RH(x0) RL(x2) td,
48  tv # RG F L*0.9 [ a1 ] RH(x0) RL(x1) td ?0.1,
49  tv # RG F L*0.9 [ a1 ] [ a1 ] RH(x0) RL(x1) td ?0.05,
50  tv # RG F L*0.7 [ a1 ] [ a1 ] [ a1 ] RH(x0) RL(x1) td ?0.05,
51  tv # z ?0.15,
52  td # F L*0.98 td ?0.4,
53  td # F RG L*0.98 [ a1 ] tv ?0.4,
54  td # F RG L*0.98 [ a1 ] [ a1 ] tv ?0.1,
55  td # F RG L*0.98 [ a1 ] [ a1 ] [ a1 ] tv ?0.1,
```

Lines 12–18 characterize the development of the *central tap root*, the symbols c and c' standing for its terminal meristem, which is gradually weakening and which dies after step 12 (the z in line 18 being a "dead-end symbol" with no further rule applicable on it). In the online graphical display, the central root shows an alternating colour pattern ($P7$ / $P4$) which was introduced to get a better impression of the lengths of successive tap-root segments. The decrease in length is controlled by the table variable in line 4.
Lines 19–22 control the diameter increments of different root segment types.
Lines 23–36 describe the development of *horizontal lateral roots* and their branching. b stands for the buds of such lateral roots (the buds in the uppermost whorl at the central axis have a slightly modified development and are therefore symbolized by b'). hw is the meristem of a horizontal root. Line 31 describes the rather regular lateral branching of such roots (with a branching angle of 50°), the lateral meristems (e) developing into new horizontal roots after 4 steps of delay (if they have not died, line 33). With a relatively low probability, the b-buds can also transform to diagonal root meristems (dg, lines 28 / 30), whose development is governed by *line 37*.
Line 38 describes a *reiteration* of the tap-root, emerging from a horizontal root according to the branching pattern of a "bending priority sign", i.e. there is a smaller, delayed root meristem ($b1$) prolonging the original axis [29]. The strength of the reiterated tap-root is assumed to be age-dependent (table parameter $x8$ in line 9).

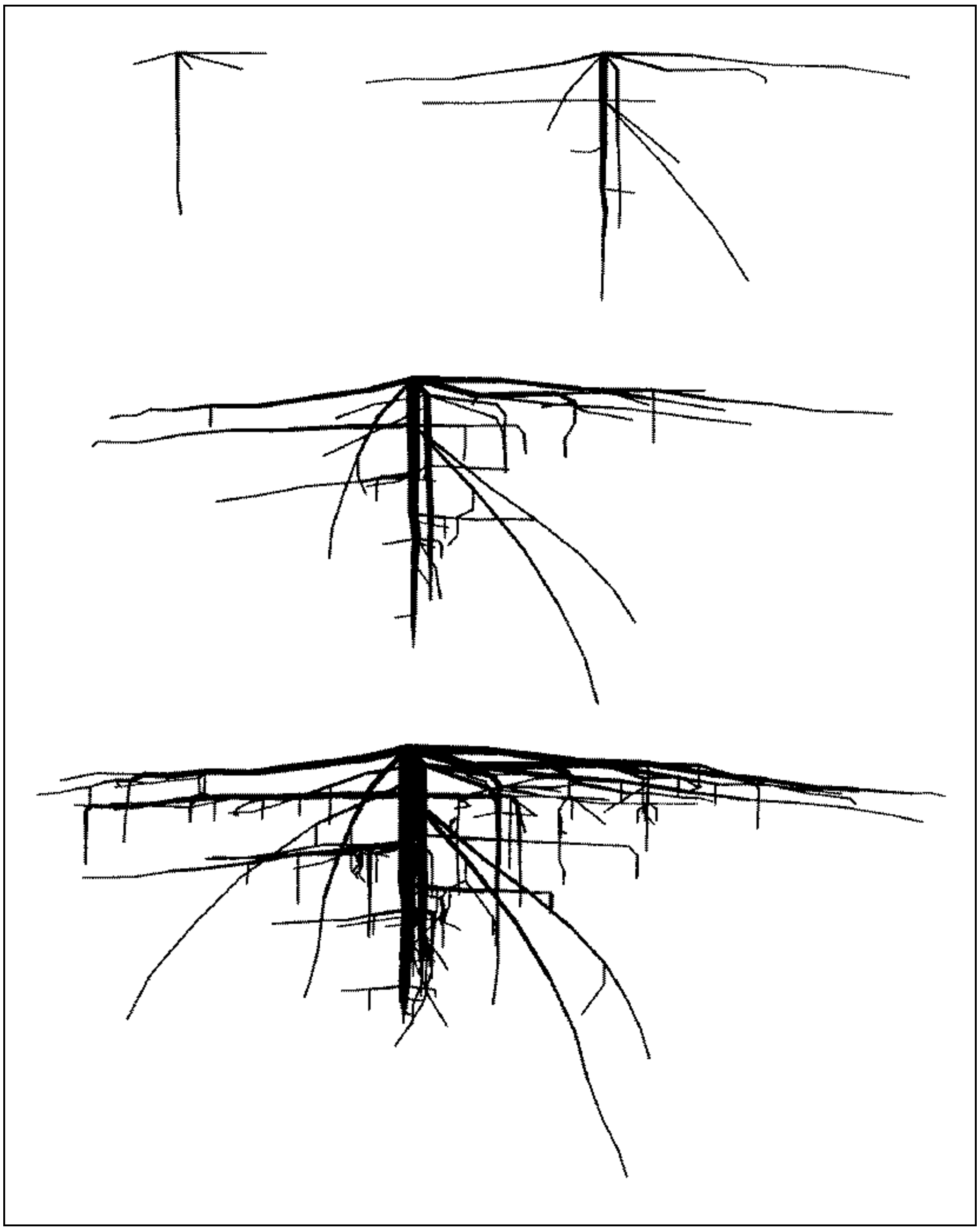


Fig. 48: root.lsy, steps 4, 8, 12, 16 (side view)

Lines 39–55 are devoted to *sinkers* which can emerge from meristems symbolized by j appearing in different positions in the lateral branching system. These sinkers have a smaller length increment and differentiate into vertically (tv) and diagonally (*td*) growing roots, the symbol *a* standing for their lateral meristems. Fig. 48 shows step 4, 8, 12 and 16 of an example developmental sequence generated from root.lsy in side view, and Fig. 49 shows the structure of step 16 from above.

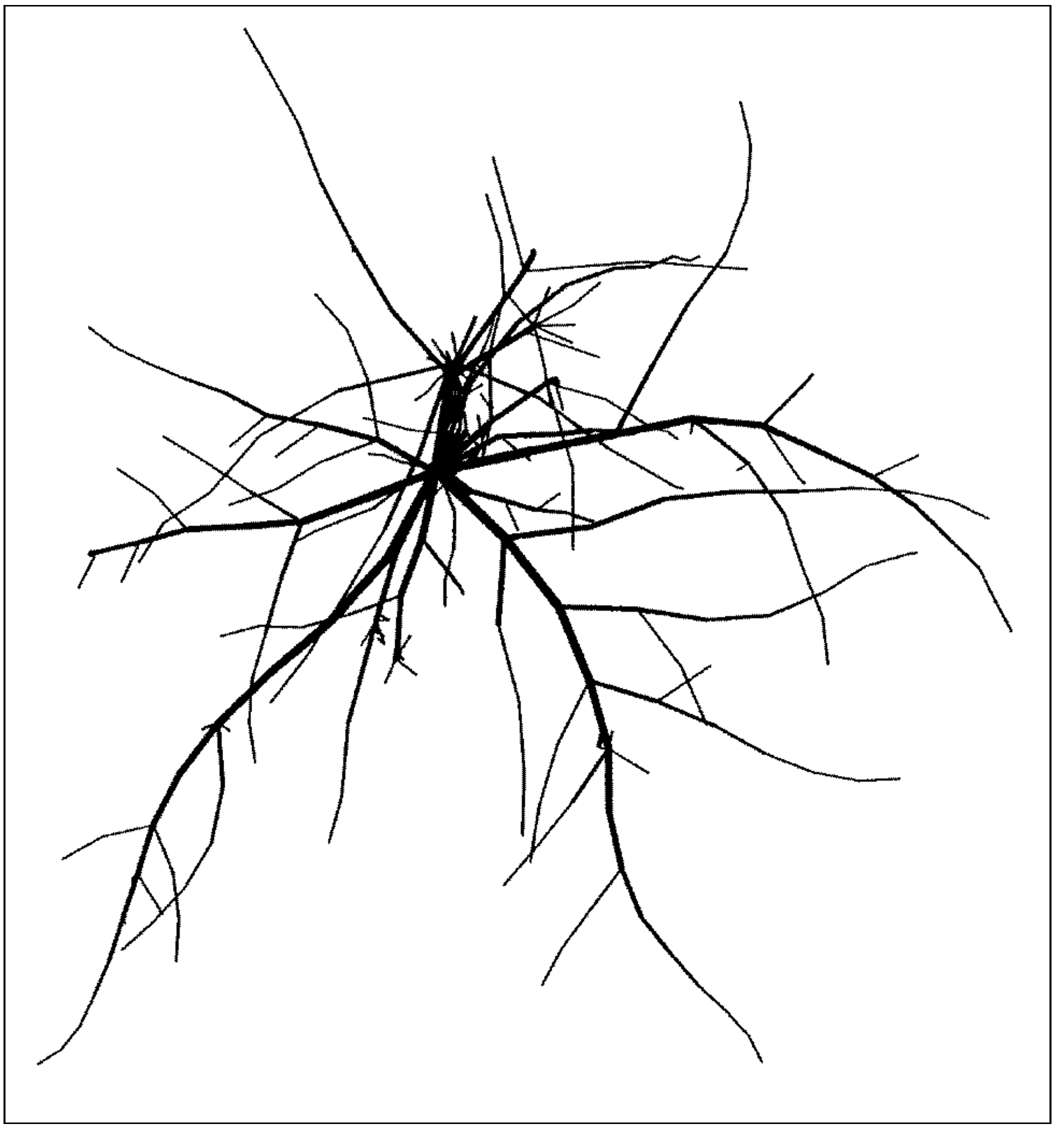


Fig. 49: root.lsy, step 16 (view from above)

6.7 as.lsy

This grammar does not serve as an example for morphological complexity, but of how method calls and register variables can be used to control growth. It generates a simplified plant, consisting of one central stem, lateral leaf-bearing branches of first order, and a single non-branched root. The plant is assumed to depend on some specific matter (we can imagine carbohydrate) for growth. The creation of each new element (stem segment, branch, root segment; in the graphical display: one line for each element) costs an amount of matter proportional to its length. Respiration is not taken into account here. New matter is produced by the branches (shown as diagonal lines) which are assumed to bear assimilating leaves and produce an amount of matter proportional to their length. In the beginning, the plant lives on an initial reserve of matter, which is realized as the content of register 3 in the grammar and asked from the user when he starts the grammar application:

```
 1  \var reg register 4,
 2  \register 1 0,
 3  \register 2 0,
 4  \ask l3 Start quantity ?,
 5  \register 4 100,
 6  * # P14 m M1 [ RG P4 u ] t,
 7  (reg >= 0) t # L(reg) Dl6 F l1+=(reg) RH180 [ RU65 P2 s ] t,
 8  (reg >= 0) s # L(reg) F F l1+=(reg) l2+=(reg),
 9  (reg >= 0) u # L(reg) F l1+=(reg) u,
10  (reg < 0) m # RU80 P15,
11  (reg < 0) P4 # P15,
12  (reg < 0) P2 # P15
```

The method number 1, evoked by the method call command "$M1$" in line 6 of this grammar, is also an integral part of the growth specification; it can be considered as the "*procedural part*" in contrast to the *rule part* provided by the grammar file. It is given in the original C language specification as a part of the source code of the program module "lmethod.c" of GROGRA:

```
void method1(void)
{
float maxinvest = 300;
float maxlength = 100;
float factor1 = 1;        /* "Demand per length" */
float factor2 = 5;        /* "Photosynthetic efficiency" */
float assimres;

if (r[4] >= 0)
    {            /* control listing of register contents */
    printf("\n r1: %f", r[1]);
    printf("\n r2: %f", r[2]);
    printf("\n r3: %f", r[3]);
```

```
        printf("\n r4: %f", r[4]);

        /* Calculation of the actual available reserve: */
        assimres = r[3] − factor1 * r[1] + factor2 * r[2];
        if (assimres <= 0)
            r[4] = −1;      /* flag for plant death */
        else
            {
            if (assimres > maxinvest)  /* optimal growth conditions */
                {
                r[3] = assimres − maxinvest;      /* new reserve */
                r[4] = maxlength;               /* maximum growth */
                }
            else                  /* restricted growth */
                {
                r[3] = 0;
                r[4] = maxlength * (assimres / maxinvest);
                }
            }
        }     /* endif */
    r[1] = r[2] = 0;  /* new calculation by l-commands in the grammar */
    }
```

The array $r[]$ is the (globally defined) array of register variables of GROGRA. $r[1]$ stands here for the total length of all plant segments created in one step and is actualized by the command `l1+ =(reg)` in lines 7–9 of the grammar. (The branches, however, are displayed with double length, as can be seen from the commands "`F F`" in line 8.) $r[2]$ is the length of the leaf-bearing branches, $r[3]$ the matter reserve of the plant (which is not assigned to some specific segment of the plant), and $r[4]$ the length to be used by the grammar for the growth of new segments (lines 7–9: "`L(reg)`"). Note that $r[4]$ (made accessible by line 1) is the only register variable which is used directly in the grammar.

In the rules, t stands for the terminal meristem of the stem axis, s for the side branch meristems, u for the root meristem. The symbol m causes plant breakdown (in a visible sense) when the matter reserves are exhausted (along with a colour change from yellow, green and red to white; lines 10–12).

Fig. 50 (a) shows the development of a plant with an initial matter reserve of 1160 units (steps 2–7) where a recovery is seen after a shortening of growth in steps 5 and 6, whereas Fig. 50 (b) shows steps 2–5 of a plant which had an initial reserve of only 1100 units and which dies in step 5 because of a lack of assimilate.

Of course, this growth grammar does not give a realistic model of the behaviour of growing seedlings, but more complex variants (e.g. including respiration, different allocation strategies or transformations between different kinds of stored carbon) could easily be realized.

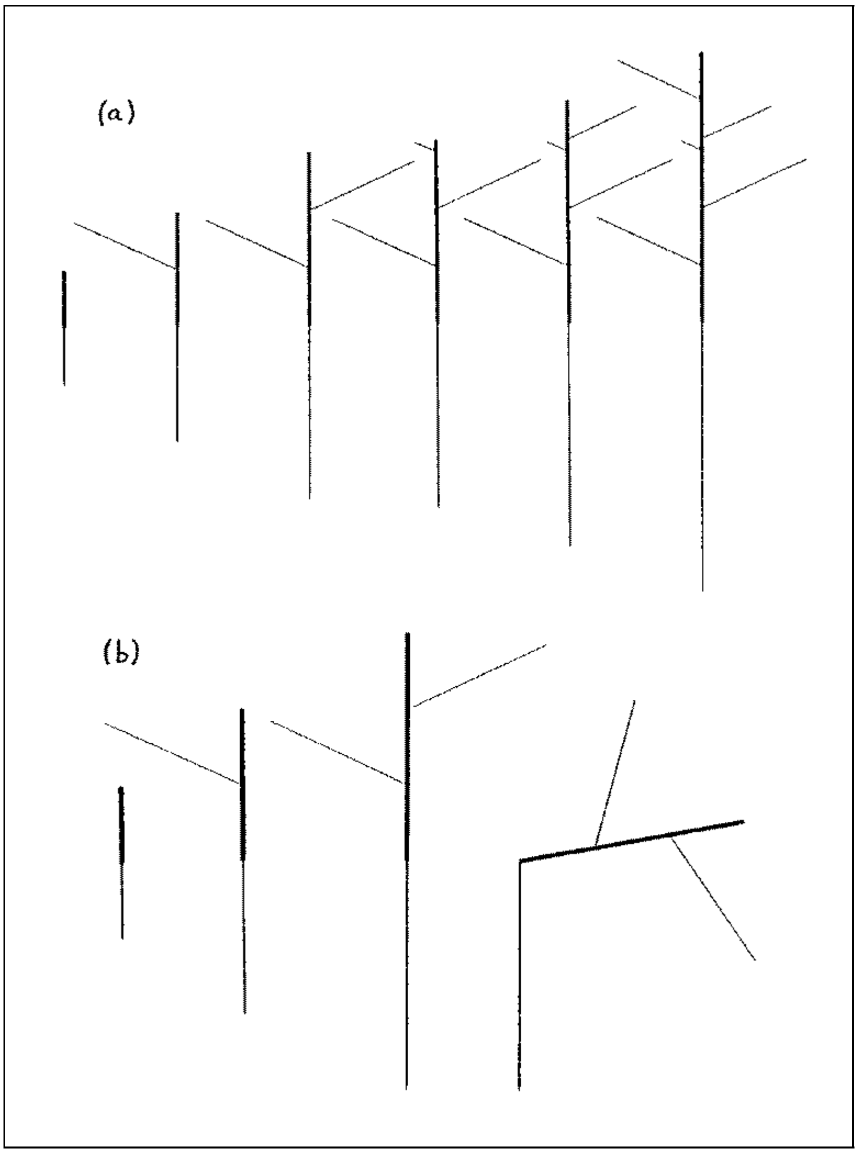


Fig. 50: as.lsy. (a) Initial reserve 1160, steps 2–7, (b) initial reserve 1100, steps 2–5.

6.8 dicho_n.lsy

A very simple example of dichotomous, exponential growth, which is included only for comparison with the sensitive grammars 6.9, 6.10 and 6.11 below.

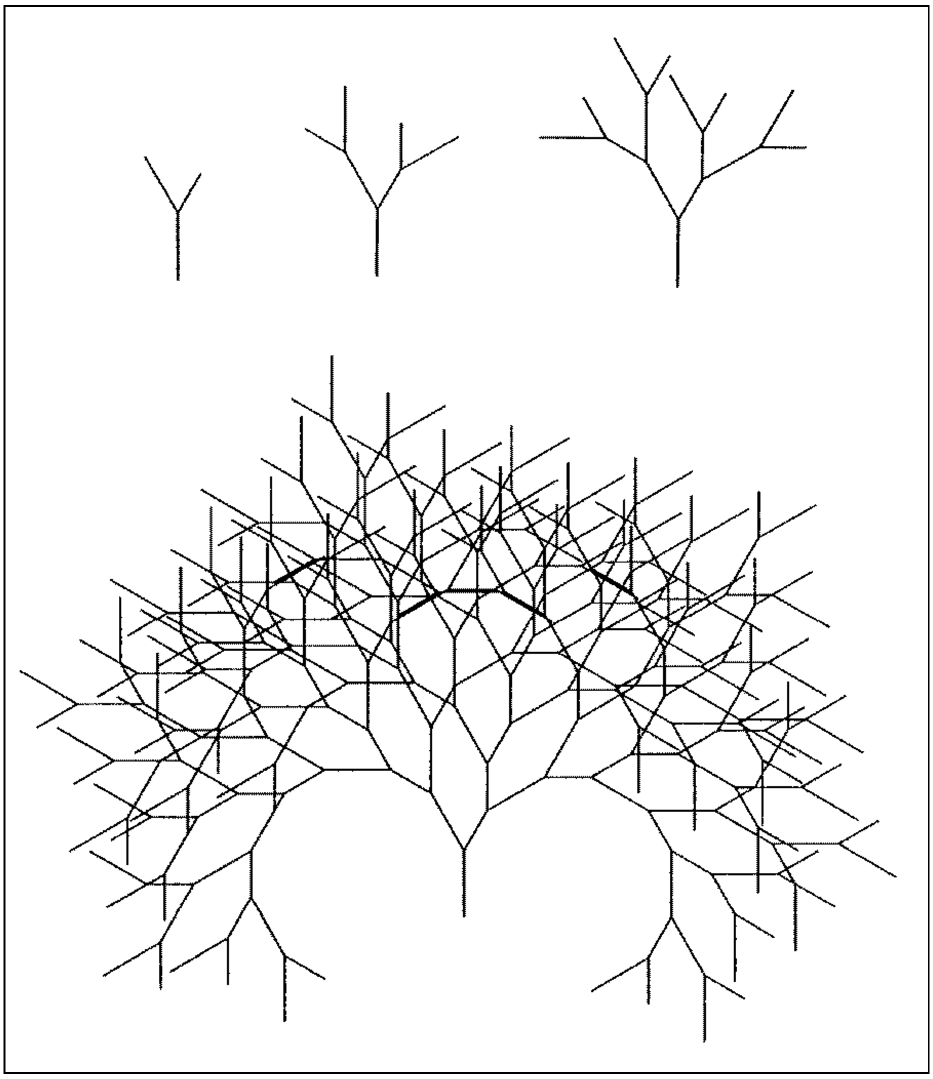


Fig. 51: dicho_n.lsy, steps 2, 3, 4 and 9

```
1    * # RH180 F100 [ RU−30 b ] RU30 a,
2    a # RH180 F100 [ RU−30 b ] RU30 a,
3    b # RH180 F70  [ RU−30 b ] RU30 a
```

Fig. 51 shows the steps 2, 3, 4 and 9 of the development which is deterministically determined by this simple grammar. (Note that rule 1 is superfluous when *a* is taken as start symbol instead of $*$.) *a* stands for the meristem generating the longer shoot type, *b* for the meristem generating the shorter shoot type. The $RH180$-command changes the orientation of the two shoots in each new generation.

6.9 dicho_d.ssy

```
1   \var f2 function 2 0,
2   * # RH180 F100 [ RU−30 b ] RU30 a,
3   (f2 > 60)  a # RH180 F100 [ RU−30 b ] RU30 a,
4   (f2 > 60)  b # RH180 F70  [ RU−30 b ] RU30 a
```

This grammar — the first example of a sensitive L-system — was derived from dicho_n.lsy by introducing the sensitive function number 2 (here used without argument) and making the application of the growth rules 3 and 4 dependent from the result of that function. $f2$ looks in the previously generated geometrical structure for the elementary unit which is associated to the symbol a (resp., b) to be transformed, and calculates the distance to the nearest other unit (excluding the mother unit — cf. Section 4.3, p. 68, and Section 4.7). The unit associated to a (resp., b) is that which was generated by the preceding $F100$- or $F70$-command. We can imagine that a and b represent sensible meristems at the tip of each non-branched shoot, ready to grow to new shoots when there is enough space around them. As the result of $f2$ must be greater than 60 length units, a circle with radius 60 must be free from other shoots. Otherwise, the "meristems" a and b remain "resting", and because no branch shedding is programmed in this grammar, they will remain inactive forever.

The resulting structure has a quite "space-filling" tendency — but remains, of course, restricted to two dimensions like dicho_n.lsy, as all branching occurs in one plane. Fig. 52 shows the structure after step 9.

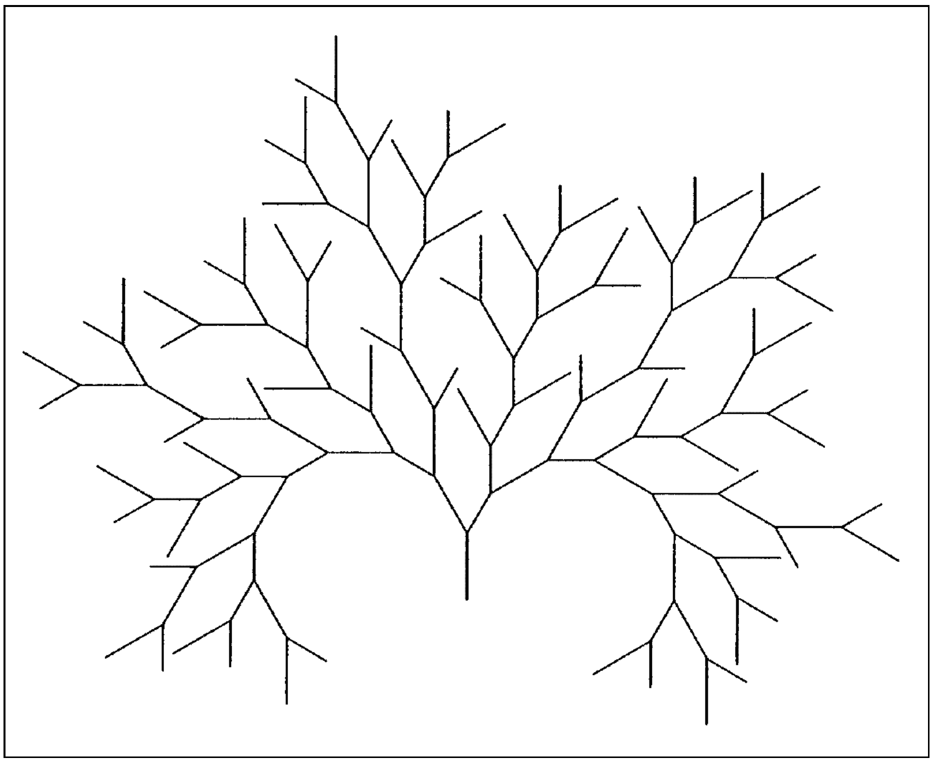


Fig. 52: dicho_d.ssy, step 9

6.10 dicho_s.ssy

Instead of the density-dependent function number 2, function number 1 is used here, which simulates a dependence of growth from obstacles in a "light cone" emerging from the tip of the considered elementary unit u. Because the structure is again two-dimensional, this cone reduces to an angular sector in the plane, opened towards the vertical (z-) direction (Fig. 53).

Function 1 returns the minimal angle which is enclosed by the line to some elementary unit end and the central axis of the sector. (The result is 0 if some unit crosses the central axis, i.e. lies directly above u.) In the conditions of the rules in line 5 and 6, this result ρ is compared with the user-defined angle r_1. If $\rho < r_1$, there will be no growth, like in the case of the "overcrowded meristems" of the previous example.

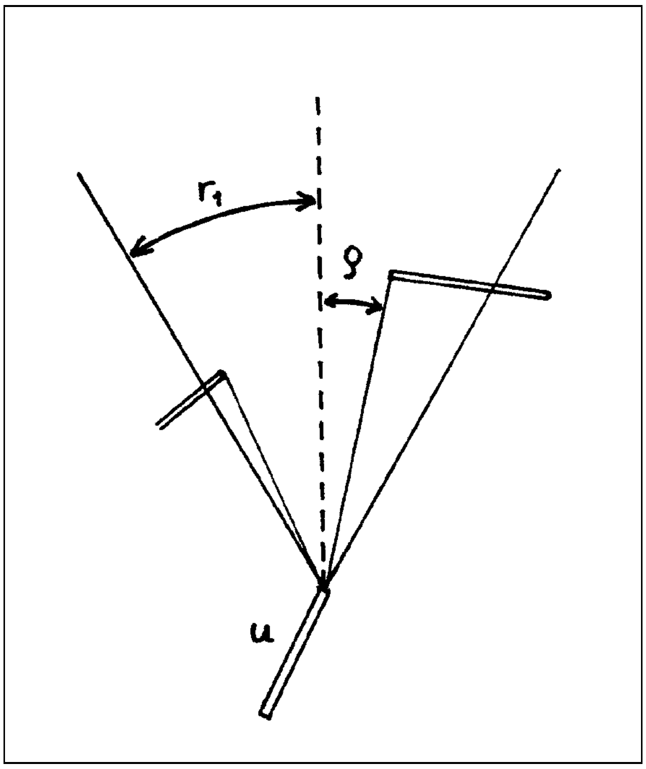


Fig. 53: Result ρ of function 1, applied to elementary unit u

```
\var f1 function 1 0,
\var r1 register 1,
\ask l1 Opening angle ?,
* # RH180 F100 [ RU−30 b ] RU30 a,
(f1 > r1)  a # RH180 F100 [ RU−30 b ] RU30 a,
(f1 > r1)  b # RH180 F70  [ RU−30 b ] RU30 a
```

Fig. 54 shows the resulting structure after 9 steps, when an opening angle r_1 of 30 degrees was chosen.

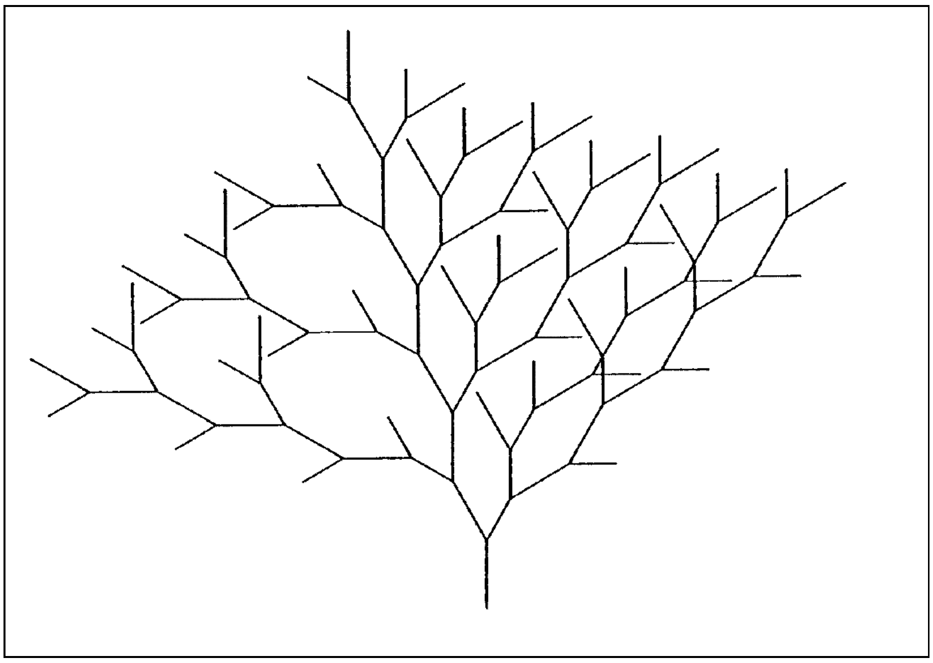


Fig. 54: dicho_s.ssy, step 9

6.11 dichomur.ssy

The sensitivity of functions can also be used to model an influence of other objects than plant shoots on the growth of a plant. The following grammar has exactly the same rules than the preceding one — except the initial rule, which creates a "wall", unchanged in the subsequent steps, besides the plant. This wall is registrated by function 1 as an "overshadowing" obstacle in the same manner as other shoots are. (The opening angle r_1 of the "light cone" is kept fixed to 30° here.)

```
\var f1 function 1 0,
* # P7 F550 RU90 F80 RU90 F550 RU-90 f*3
        RU-90 P14 RH180 F100 [ RU-30 b ] RU30 a,
(f1 > 30)  a # RH180 F100 [ RU-30 b ] RU30 a,
(f1 > 30)  b # RH180 F70  [ RU-30 b ] RU30 a
```

Fig. 55 shows steps 2, 4, 6, 8, 10 and 12 of the resulting development.

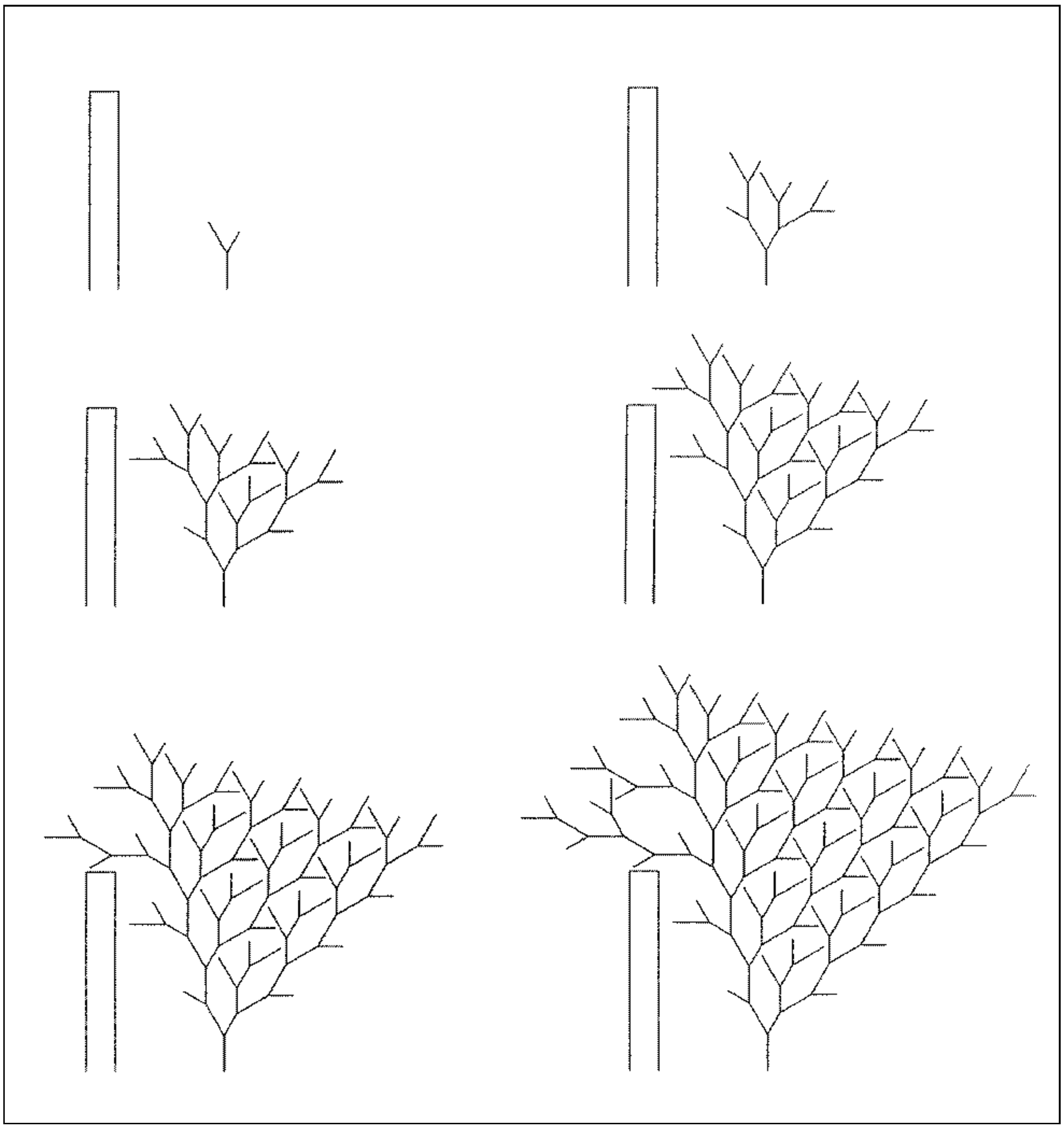


Fig. 55: dichomur.ssy, steps 2, 4, 6, 8, 10, 12

6.12 pic0.lsy

We leave now the sensitive grammars again and return to the modelling of spruce crowns (*Picea abies* (L.) Karst.) (cf. Example 6.4, p. 139). The following grammar is based on rough empirical observations of SEIBT [110] on research areas in the Solling (Germany), made at trees in the ages 85, 90 and 95 years. (The back-extrapolation to young trees which is undertaken here is not to be understood as empirically settled.) The regression for height growth which is used in line 46 below is a simplified version of an equation of HOULLIER and LEBAN [59] and was fitted to the values of [110], the three-sections approach for diameter increment (lines 51–54) comes from KOBAYASHI [67], and the needle area calculation (lines 87–103) again from [74] and [92], as in Example 6.5. The positions and dynamics of proventive shoot growth are oriented at results of GRUBER [49]. The purpose of this example is to illustrate how such empirical approaches and regressions can be included in a morphological model based on GROGRA.

```
 1  \const bh 1300,
 2  \const be -0.0113,
 3  \const g1 0.014,
 4  \const g2 0.2,
 5  \const d1 1.5,
 6  \const dm1 0.9,
 7  \const d2 0.8,
 8  \const d3 0.7,
 9  \const k 1E-8,
10  \const cb 0.9888,
11  \const sw 452,
12  \const tc3 4,
13  \const tp 5,
14  \const vf 0.01,
15  \const at 0.97,
16  \const m0 13.84,
17  \const m1 1.14,
18  \const m2 5.4,
19  \const m3 1.385,
20  \var t0 function 10 1,
21  \var f11 function 11 5,
22  \var g generation,
23  \var n1 normal 1 0.05,
24  \var n2 normal 0 4,
25  \var n3 normal 0 30,
26  \var n4 normal 0 10,
\var ba uniform 0.03 0.1,
\var me uniform 0.3 0.8,
\var su uniform 0.85 0.97,
\var lg length,
\var dh register 0,
\var kb register 1,
\var va register 2,
\var rf table 0.1 0.2 0.4 0.6 0.7 0.8 0.85 0.9 0.95 1,
\var tm table 1 1 1 2 2 2 2 2 3 3 4,
\var tc2 table 2 3 3 4 4 5 5 5 6 6 6 6 6 6 6 7,
\var kf table 2.5 2.5 2 2 1.5 1.5 1,
\var ro uniform 0 360,
\var ns distribution 0 0 0 0 0 0.2 0.5 0.2 0.1 0,
\var nm distribution 0 0 0 0.1 0.2 0.2 0.2 0.2 0.1 0,
\var i index,
\set D 4,
\set N 0,
\register 2 1,
* # bas(1) tk,
bas(t) # J0=(sw*rf*exp(be*t)) J2=(n1) bas(t+1),
J0=(x) # ,
J2=(x) # ,
tk # P4 t(0,0),
t(t, h) # L(dh) a(0,h,6,va) RH(ro)
   &(nm) < [ @(me) RH(i*360/nm_+n3) L*(0.4*kf) D w(0,n2+5) $ m(0) ] >
   RH(ro)
   &(ns) < [ @(su) RH(i*360/ns_+n4) L*(0.65*kf) D w(0,n2) $ s(0) ] >
   t(t+1,h+dh*va),
(t < t0(g)) a(t, h, d, y) # a(t+1,h,d+g1*dh,y),
(t >= t0(g)) a(t, h, d, y) # b(h,d+g1*dh,y) J1=(h),
(h >= bh) b(h, d, y) # b(h,d+g1*dh*exp((-k)*(kb-h)),y),
(h < bh) b(h, d, y) # b(h,d+g1*dh*exp((-k)*(kb-h))*(1+(bh-h)*g2/bh),y),
J1=(x) # ,
w(t, y) # w(t+1,y),
(t <= 3) s(t) # c1(t,0,va)
   g(0, vf) [ @(me) L*0.4 RH-15 RU60 $ sm2 ]
   [ @(me) L*0.4 RH15 RU-60 $ sm2 ]
   g(0, vf) [ @(su) L*0.7 RH-15 RU45 $ s2 ]
   [ @(su) L*0.7 RH15 RU-45 $ s2 ]
   L*(at) s(t+1),
58  (t > 3) s(t) # c1(t,0,va)
      h(0,0.001) [ @(ba) L*0.5 RH-60 RU80 $ p(3,0) ]
      [ @(ba) L*0.5 RH60 RU-80 $ p(3,0) ]
      h(0,0.008) [ @(su) L*0.6 RH-50 RU70 $ p(2,0) ]
      [ @(su) L*0.6 RH50 RU-70 $ p(2,0) ]
      h(0,0.02) [ @(ba) L*0.65 RH-30 RU60 $ p(1,0) ]
      [ @(ba) L*0.65 RH30 RU-60 $ p(1,0) ]
      g(0, vf*1.5) [ @(me) L*0.4 RH-15 RU60 $ sm2 ]
      [ @(me) L*0.4 RH15 RU-60 $ sm2 ]
      g(0, vf) [ @(su) L*0.7 RH-15 RU45 $ s2 ]
      [ @(su) L*0.7 RH15 RU-45 $ s2 ]
      L*(at) s(t+1),
59  m(t) # cm1(t,0,va)
      g(0,vf*1.5) [ @(su) L*0.7 RH-15 RU45 $ ms2 ]
      [ @(su) L*0.7 RH15 RU-45 $ ms2 ]
      L*(at) m(t+1),
60  (j==0 && t==t0(g)-3) c1(j,t,y) # c1(j,t+1,y) % ?0.1,
61  (j==0 && t==t0(g)-2) c1(j,t,y) # c1(j,t+1,y) % ?0.4,
62  (j==0 && t==t0(g)-1) c1(j,t,y) # c1(j,t+1,y) %,
63  (j==0 && t>=t0(g)+3) c1(j,t,y) # %,
64  c1(j,t,y) # c1(j,t+1,y),
65  (j==0 && t==tm-1) cm1(j,t,y) # cm1(j,t+1,y) % ?0.3,
66  (j==0 && t==tm) cm1(j,t,y) # cm1(j,t+1,y) %,
67  (j==0 && t>=tm+6) cm1(j,t,y) # %,
68  cm1(j,t,y) # cm1(j,t+1,y),
69  g(t, s) # g(t+1,s),
70  h(t, s) # h(t+1,s),
71  s2 # RV c2(0,tc2,va)
      g(0,vf) [ @(su) L*0.7 RH-5 RU45 $ s3 ]
      [ @(su) L*0.7 RH5 RU-45 $ s3 ]
      L*(at) s2 ?0.93,
72  s2 # RV c2(0,tc2,va)
      L*(at) s2 ?0.07,
73  ms2 # RV c2(0,3,va)
      g(0,vf) [ @(su) L*0.7 RH-5 RU45 $ s3 ]
      [ @(su) L*0.7 RH5 RU-45 $ s3 ]
      L*(at) ms2 ?0.6,
74  ms2 # RV c2(0,3,va)
      L*(at) ms2 ?0.4,
75  sm2 # RV c2(0,3,va)
      g(0,vf) [ @(su) L*0.7 RH-5 RU45 $ s3 ]
      [ @(su) L*0.7 RH5 RU-45 $ s3 ]
      L*(at) sm2 ?0.8,
76  sm2 # RV c2(0,3,va) L*(at) sm2 ?0.2,
77  (t < 4*k+2) p(k, t) # p(k, t+1),
78  (t >= 4*k+2) p(k, t) # RV c2(0,tp,va)
       L*(at) p(k,t+1) ?((1-k*0.07)*0.4),
79  (t >= 4*k+2) p(k, t) # RV c2(0,tp,va)
       h(0,0.03) [ @(su) L*0.7 RH-5 RU45 $ s3 ]
       [ @(su) L*0.7 RH5 RU-45 $ s3 ]
       L*(at) p(k,t+1) ?((1-k*0.1)*0.6),
80  (t >= 4*k+2) p(k, t) # ?(k*0.07),
81  s3 # RV c3(0,tc3,va) L*(at) s3 ?0.9,
82  s3 # ?0.1,
83  (t <= s) c2(t, s, y) # c2(t+1,s,y),
84  (t > s) c2(t, s, y) # %,
85  (t <= s) c3(t, s, y) # c3(t+1,s,y),
86  (t > s) c3(t, s, y) # %,
87  (t <= 2) a(t, h, d, y) ## P2 D(d) N(m0*lg^m1+m2*lg^m3) F*(y),
88  (t > 2 && t <= 5) a(t, h, d, y) ## P2 D(d)
       N((m0*lg^m1+m2*lg^m3)*(5-t)/3) F*(y),
89  (t > 5) a(t, h, d, y) ## P2 D(d) N0 F*(y),
90  b(h, d, y) ## D(d) N0 F*(y),
91  w(t, y) ## RU(atg(0.19*(t-3))/3+60+y),
92  (t <= 2) c1(j,t,y) ##
       V(f11(j,t,2.5/(j+t+10)^2,6/(j+t+10)^2,0.3-0.015*sqrt(j+t)))
       RV Dl+(d1*t) N(m0*lg^m1+m2*lg^m3) F*(y),
93  (t > 2 && t <= 7) c1(j,t,y) ##
       V(f11(j,t,2.5/(j+t+10)^2,6/(j+t+10)^2,0.3-0.015*sqrt(j+t)))
       RV Dl+(d1*t) N((m0*lg^m1+m2*lg^m3)*(7-t)/5) F*(y),
94  (t > 7) c1(j,t,y) ##
       V(f11(j,t,2.5/(j+t+10)^2,6/(j+t+10)^2,0.3-0.015*sqrt(j+t)))
       RV Dl+(d1*7+0.3*d1*(t-7)) N0 F*(y),
95  (t <= 2) cm1(j,t,y) ## V(f11(j,t,0.01,0.01,0.25)) RV Dl+(dm1*t)
       N(m0*lg^m1+m2*lg^m3) F*(y),
96  (t > 2 && t <= 7) cm1(j,t,y) ## V(f11(j,t,0.01,0.01,0.25))
       RV Dl+(dm1*t) N((m0*lg^m1+m2*lg^m3)*(7-t)/5) F*(y),
97  (t > 7) cm1(j,t,y) ## V(f11(j,t,0.01,0.01,0.25))
       RV Dl+(dm1*7+0.1*dm1*(t-7)) N0 F*(y),
98  (t <= 2) c2(t, s, y) ## Dl+(d2*t) N(m0*lg^m1+m2*lg^m3) F*(y),
99  (t > 2 && t <= 7) c2(t, s, y) ## Dl+(d2*t)
       N((m0*lg^m1+m2*lg^m3)*(7-t)/5) F*(y),
100 (t > 7) c2(t, s, y) ## Dl+(d2*t) N0 F*(y),
101 (t <= 2) c3(t, s, y) ## Dl+(d3*t) N(m0*lg^m1+m2*lg^m3) F*(y),
102 (t > 2 && t <= 7) c3(t, s, y) ## Dl+(d3*t)
       N((m0*lg^m1+m2*lg^m3)*(7-t)/5) F*(y),
103 (t > 7) c3(t, s, y) ## Dl+(d3*t) N0 F*(y),
104  g(t,s) ## V(s*(t*t−2*t+1)),
105  h(t,s) ## V(s*(t−1)),
```

As in the example shoot.lsy above, only a few features of this complex growth grammar shall be commented here.

Register 0 (associated with variable dh, line 31) contains the *height increment* of the tree in the actual step (one developmental step representing one year). This value is actualized by the J command in line 46, using the constants sw and be from lines 2 and 11, and also a growth reduction factor rf (damping) for the young tree, specified explicitely as a table variable in line 34.
The height of the *crown basis* is given as a function of age (line 20). This height is involved in the calculation of *diameter increment* in lines 51–54. Stem segments in the living crown are symbolized by a, stem segments below by b. In the living crown, the yearly diameter increment is assumed to be linearly dependent from height increment dh and independent from the height of the segment ([67]; line 51: $d + g1 \ * \ dh$). In the branchless stem part, diameter increment is exponentially diminuishing with height (line 53). Below breastheight (bh), however, the diameter growth of the trunk is again stronger, this strengthening being reflected in line 54 (cf. [67]).
The development of the *branching angle* of the main branches with age is approximated by a fitting function in line 91, tending versus 90° when the branch grows old. A *stochastic variation* $n2$, however, disturbs the determinism slightly. A variation in the length increments of all new shoots is also included in the grammar, representing yearly climatic influences (register 2, represented by variable va; line 46).
The morphological development of the lateral *branching* is rather simply modelled in this grammar; only branching orders 1, 2 and 3 are taken into account, and, as in the case of Example 6.4, only the rough differentiation into “subapical” and “medial” shoots is made to model acrotony (symbols s, m, $s2$, $ms2$ etc.). The rule in line 50, specifying the distribution of main branches at the stem, is comparable to the rule in line 14 of ficht5.lsy (p. 139). The shoots of order 1, 2 and 3 are symbolized by $c1$, $cm1$, $c2$ and $c3$. Added is a growth from *proventive* (delayed) buds (p) in the basal region of first-order shoots (lines 58 and 77–80).
Branch bending is modelled with the help of V- and RV-commands. The main (i.e. first-order) branches are considered separately (lines 92–97) to ensure their characteristic form. Another spruce phenotype, distinguished by a more or less intensive hanging down of second-order branches ([47]), can easily be specified by a change in the second parameter (s) of the symbol g (lines 57–59 and 71–75), which is translated in line 104 in an age-dependent V-command.
The *shedding* of subapical main branches occurs around the crown base position, but is not completely deterministic (lines 60–63). Note that the shoot symbol $c1$ is itself transformed into the cut operator %, thus avoiding the use of an extra symbol for that purpose. Some dead stubs remain for 4 years. The medial

branches die much earlier (lines 65–67).

Needle surface development is modelled by the regressions in lines 87, 88, 92–102, cf. Example 6.5. The replication of rules (92–94, 95–97, 98–100, 101–103) is due to the inclusion of needle losses with age. *Branchwood thickening* is assumed to be constant with time, but depends on branching order and position (commands *Dl*+ in lines 92–103).

Because of the two rules in lines 46 and 49 which precede the development of any visible structure, the developmental step index exceeds the age (in years) of the modelled tree by 2. Fig. 56 shows the developmental steps 7, 12, 17, 22 and 27, thus representing the same spruce tree in the ages of 5, 10, 15, 20 and 25 years.

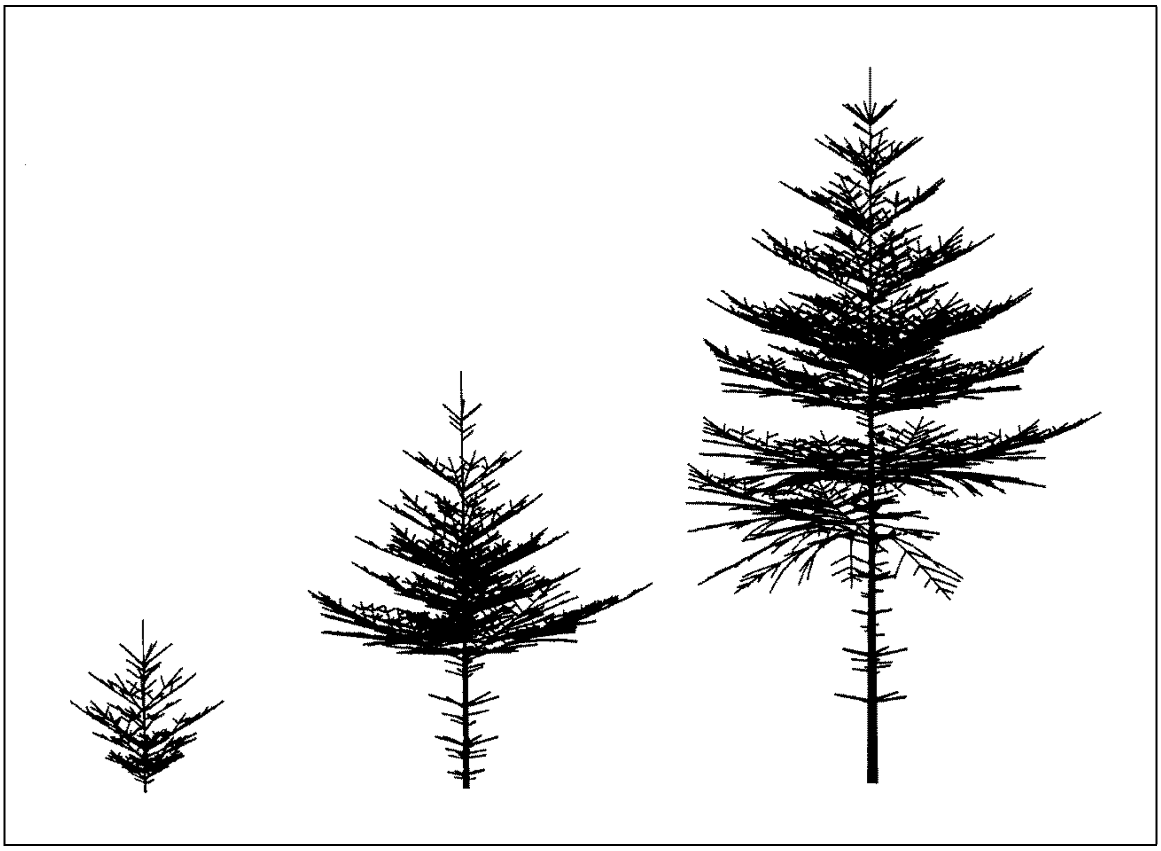


Fig. 56: pic0.lsy, steps 7, 12 and 17

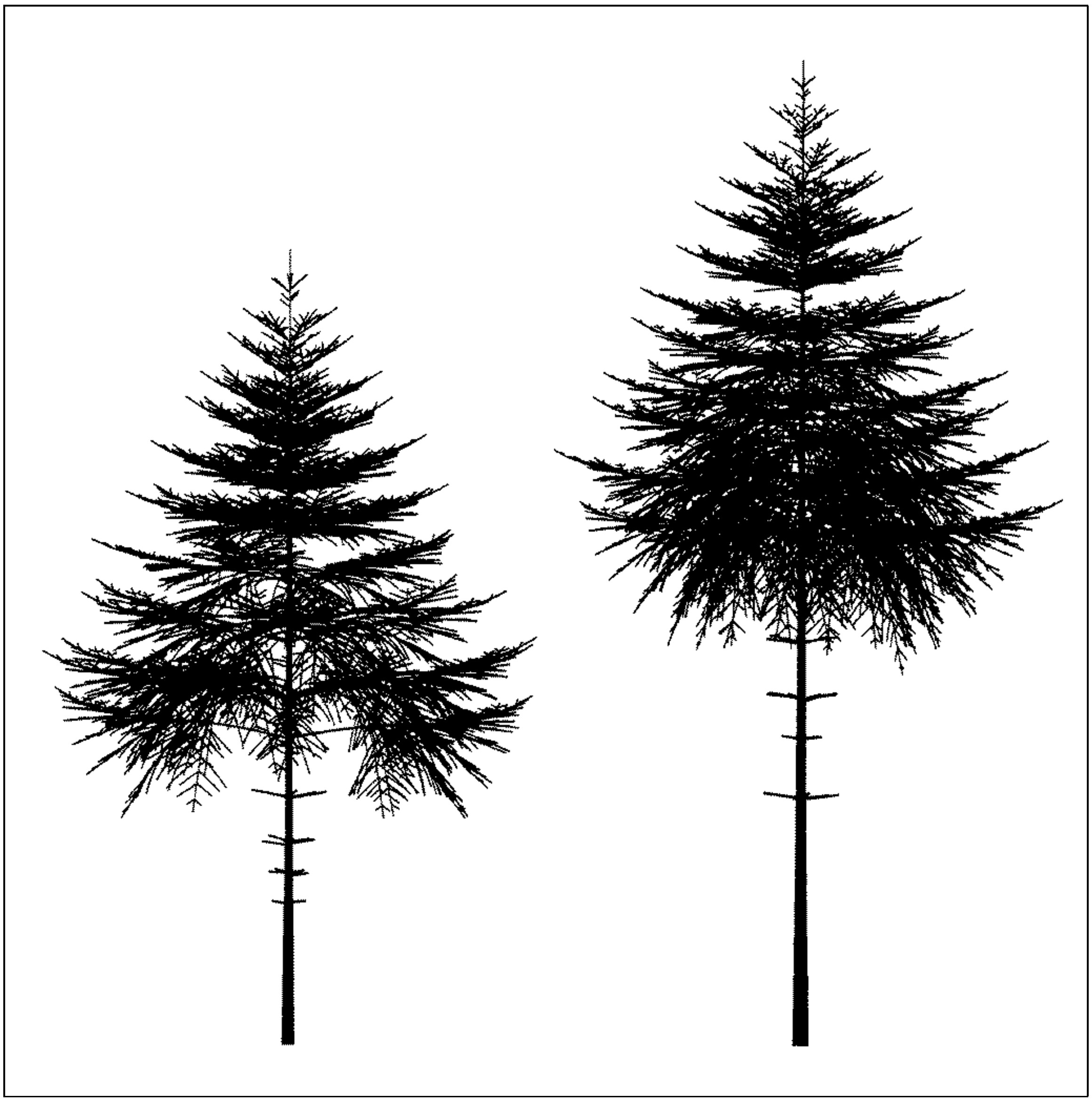


Fig. 56 (continued): pic0.lsy, steps 22 and 27

Appendix 1: The colour table of GROGRA

0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	brown
7	light grey
8	dark grey
9	light blue
10	light green
11	light cyan
12	light red
13	light magenta
14	yellow
15	white

Appendix 2: The turtle commands

In this table, x stands for a real and i for an integer.

Command	Effect
L	$\ell_L = \ell = \ell_g$
L(x)	$\ell_L = \ell = x$
L+(x)	$\ell_L = \ell = \ell + x$
L∗(x)	$\ell_L = \ell = \ell \cdot x$
Ll(x)	$\ell_L = x$
Ll+(x)	$\ell_L = \ell + x$
Ll∗(x)	$\ell_L = \ell \cdot x$
D	$d_L = d = d_g$
D(x)	$d_L = d = x$
D+(x)	$d_L = d = d + x$
D∗(x)	$d_L = d = d \cdot x$
Dl(x)	$d_L = x$
Dl+(x)	$d_L = d + x$
Dl∗(x)	$d_L = d \cdot x$
V	$v_L = v = v_g$
V(x)	$v_L = v = x$
V+(x)	$v_L = v = v + x$
V∗(x)	$v_L = v = v \cdot x$
Vl(x)	$v_L = x$
Vl+(x)	$v_L = v + x$
Vl∗(x)	$v_L = v \cdot x$
N	$n_L = n = n_g$
N(x)	$n_L = n = x$
N+(x)	$n_L = n = n + x$
N∗(x)	$n_L = n = n \cdot x$
Nl(x)	$n_L = x$
Nl+(x)	$n_L = n + x$
Nl∗(x)	$n_L = n \cdot x$
P	$p_L = p = p_g$
P(i)	$p_L = p = i$
Pl(i)	$p_L = i$

Command	Effect
F	$P = P + \ell_L \cdot H$, $q = 0$, $g = g + 1$, m actualized, unit construction
F(x)	$P = P + x \cdot H$, $q = 0$, $g = g + 1$, m actualized, unit construction
F+(x)	$P = P + (\ell_L + x) \cdot H$, $q = 0$, $g = g + 1$, m actualized, unit construction
F∗(x)	$P = P + \ell_L \cdot x \cdot H$, $q = 0$, $g = g + 1$, m actualized, unit construction
f	$P = P + \ell \cdot H$, $q = q - 1$
f(x)	$P = P + x \cdot H$, $q = q - x/\ell$
f+(x)	$P = P + (\ell + x) \cdot H$, $q = q - (\ell + x)/\ell$
f∗(x)	$P = P + \ell \cdot x \cdot H$, $q = q - x$
@(x)	$P = P + \ell \cdot (x - 1) \cdot H$, $q = 1 - x$
RH(x)	$L = L\ \cos x + U\ \sin x$, $U = -L\ \sin x + U\ \cos x$
RL(x)	$H = H\ \cos x + U\ \sin x$, $U = -H\ \sin x + U\ \cos x$
RU(x)	$H = H\ \cos x - L\ \sin x$, $L = H\ \sin x + L\ \cos x$
RV	$H = \mathit{normalized}(H - v_L \cdot (0,\ 0,\ 1))$, U also changed
RV(x)	$H = \mathit{normalized}(H - x \cdot (0,\ 0,\ 1))$, U also changed
RV+(x)	$H = \mathit{normalized}(H - (v_L + x) \cdot (0,\ 0,\ 1))$, U also changed
RV∗(x)	$H = \mathit{normalized}(H - v_L \cdot x \cdot (0,\ 0,\ 1))$, U also changed
RG	$H = (0,\ 0,\ -1)$, L and U also changed
+	same as RU(w_g)
−	same as RU($-w_g$)
$	corrects U and L for U pointing upwards as steeply as possible
[	puts current state on stack; $b = b + 1$
]	re-installs former state from stack
%	stops command execution until next] on same level (or till the end)
Ii =(x)	$r_i = x$
Ii+ =(x)	$r_i = r_i + x$
Ii∗ =(x)	$r_i = r_i \cdot x$
Ji =(x)	$r_i = x$ at generation time
Ji+ =(x)	$r_i = r_i + x$ at generation time
Ji∗ =(x)	$r_i = r_i \cdot x$ at generation time
Mi	execution of method i

Bibliography

[1] Abelson, H., and diSessa, A. A.: Turtle Geometry. M.I.T. Press, Cambridge 1982.

[2] Aono, Masaki, and Kunii, Tosiyasu L., Botanical tree image generation. *IEEE Computer Graphics and Applications,* **4** (1984), 10–34.

[3] Autodesk AG: AutoCAD 10 Benutzerhandbuch. Switzerland 1990.

[4] Barnsley, Michael F.: Fractals Everywhere. Associated Press, Boston 1988.

[5] Barthélemy, Daniel; Blaise, Frédéric; Guédon, Yann, et Reffye, Philippe de, Modélisation de l'architecture et du mode de floraison des bouts pendants de la vanille (*Vanilla planifolia*) à l'île de la Réunion. Convention Coopérative Agricole des Producteurs de Vanille / Laboratoire de modélisation du CIRAD/GERDAT, Montpellier, Déc. 1991.

[6] Bassow, Susan L.; Ford, E. David, and Kiester, A. Ross, A critique of carbon-based tree growth models. Process Modeling of Forest Growth Responses to Environmental Stress (Eds.: R. K. Dixon et al.), Portland 1990, 50–57.

[7] Bell, Adrian, Computerized vegetative mobility in rhizomatous plants. Automata, Languages, Development. Eds.: A. Lindenmayer and G. Rozenberg. Amsterdam 1976, 3–14.

[8] Bell, Adrian D.; Roberts, D., and Smith, A., Branching patterns: The simulation of plant architecture. *Journal of Theoretical Biology,* **81** (1979), 351–375.

[9] Bell, Adrian D., A summary of the branching process in plants. Shape and Form in Plants and Fungi. The Linnean Society of London (ISBN 0-12-371035-9) 1994, 119–142.

[10] Berger, David S., Modification of a simple fractal tree growth scheme: Implications on growth, variation, and evolution. *Journal of Theoretical Biology,* **152** (1991), 513–529.

[11] Blaise, Frédéric: Simulation du parallélisme dans la croissance des plantes et applications. Thèse, Université Louis Pasteur, Strasbourg, Département d'Informatique 1991.

[12] Blaise, Frédéric, Simulation de couverts végétaux réalistes en 3-D. XVIIème Congrès I.S.P.R.S., Washington D.C., Aug. 1992.

[13] Boer, Martin J. M. de; Bunke, H.; Cuny, J.; Fracchia, F. David et al., Panel discussion: The use of graph grammars in applications. *Lecture Notes in Computer Science,* **532** (1991), p. 41 ff.

[14] Borel-Donohue, C. C.: Fractal models for the structure of trees. Models for Back Scattering of Millimeter Waves from Vegetative Canopies. Ph.D. Thesis, Amhurst 1988. Chapter 6, 145–176.

[15] Borland GmbH (Hersg.): Turbo C++ 3.0 Programmierhandbuch. Starnberg 1992.

[16] Bossel, Hartmut: TREEDYN 3 Forest Simulation Model. Mathematical model, program documentation, and simulation results. *Berichte des Forschungszentrums Waldökosysteme,* **B 35**, Göttingen 1994.

[17] Brauer, Wilfried: Automatentheorie. B. G. Teubner, Stuttgart 1984.

[18] Breckling, Broder: Contribution to the Workshop "Struktur- und Funktionsmodelle", Göttingen / Nienover, 22.–24. 6. 1994.

[19] Caraglio, Yves; Elguero, Eric; Mialet, Isabelle, et Rey, Hervé: Le Peuplier - Modélisation et simulation de son architecture. Convention IDF/CIRAD, Document no. 008 (Jan. 1990), Laboratoire de modélisation du GERDAT, Montpellier.

[20] Chen, S. G.; Mau, F.; Ceulemans, R., and Impens, I., Dynamic modelling approach for real tree architecture of short rotation intensively cultured *Populus* based on fractal theory. *Naturalia Monspeliensia, h.s.,* **1991**. L'Arbre. Biologie et Développement (Ed.: C. Edelin), Montpellier, 580–581.

[21] Chen, X., et Lienhardt, Pascal, Modélisation et programmation d'évolutions de subdivisions de surfaces. Journées Graphiques GROPLAN 1990, 28.–30. 11. 1990, Ecole des Mines de Saint-Etienne, Dép. Informatique.

[22] Cochrane, Lynda A., and Ford, E. David, Growth of a Sitka spruce plantation: Analysis and stochastic description of the development of the branching structure. *J. Applied Ecology,* **15** (1978), 227–244.

[23] Costes, Evelyne, et Reffye, Philippe de: Etude préliminaire sur la modélisation et la simulation de l'architecture de l'abricotier (*Prunus armeniaca* L.). Convention CTIFL/CIRAD, Laboratoire de Modélisation du CIRAD, Montpellier 1988.

[24] Costes, Evelyne, et Reffye, Philippe de: Modélisation de l'architecture de 3 clones d'Hévéa. Convention IRCA/GERDAT, Décembre 1990, Rapport no. 13/90, Montpellier.

[25] Dabadie, Pascal: Contribution à la modélisation et simulation de la croissance des végétaux. Thèse de Diplôme de Doctorat, Université Montpellier II, 1991.

[26] Dauzat, Jean, Simulated plants and radiative transfers simulations. Colloq. Structure du Couvert Végétal et Climat Lumineux: Méthodes de Caractérisation et Applications. Saumanc (France), 23.–27. 9. 1991.

[27] Dauzat, Jean, Communication at the Workshop on "Utilization of models of canopy architecture for simulations of light interception and photosynthesis", Göttingen, 13.–14. 4. 1993.

[28] Diggle, A. J., ROOTMAP — a model in three-dimensional coordinates of the growth and structure of fibrous root systems. *Plant and Soil,* **105** (1988), 169–178.

[29] Drexhage, Michael, oral communication.

[30] Ford, E. David, and Bassow, Susan L., Modeling the dependence of forest growth on environmental influences. *NATO ASI Ser. E,* **166**. Biomass Production by Fast-growing Trees. Eds.: J. S. Pereira and J. J. Landsberg. Dordrecht 1989, 209–229.

[31] Ford, E. David, and Kiester, A. Ross, Modeling the effects of pollutants on the processes of tree growth. Process Modeling of Forest Growth Responses to Environmental Stress (Eds.: R. K. Dixon et al.), Portland 1990, 324–337.

[32] Ford, E. David; Avery, Anne, and Ford, R., Simulation of branch growth in the Pinaceae: Interactions of morphology, phenology, foliage productivity, and the requirement for structural support, on the export of carbon. *J. Theor. Biol.,* **146** (1990), 15–36.

[33] Ford, R., and Ford, E. David, Structure and basic equations of a simulator for branch growth in the *Pinaceae. J. Theor. Biol.,* **146** (1990), 1–13.

[34] Forschungszentrum Waldökosysteme der Georg-August-Universität Göttingen: Antrag auf Förderung eines Verbundvorhabens "Veränderungsdynamik von Waldökosystemen", Teil II, Göttingen, Okt. 1993. There: Allgemeine Darstellung der Modellbildung im Rahmen des Forschungsansatzes des FZW, p. 302 ff.

[35] Fournier, Dominique: Modélisation de la croissance et de l'architecture du Merisier (*Prunus avium* L.). Diplôme d'Agronomie Approfondie, Ecole Nationale Superieure Agronomique de Montpellier. Réalisé a l'INRA et au CIRAD. Montpellier 1989.

[36] Fournier, M.; Rogier, P.; Costes, Evelyne, et Jaeger, Marc, Modélisation mécanique des vibrations propres d'un arbre soumis aux vents, en fonction de sa morphologie. *Annales des Sciences Forestières,* **50** (1993), 401–412.

[37] Franco, M., The influence of neighbours on the growth of modular organisms with an example from trees. *Phil. Trans. Royal Soc. London, B,* **313** (1986), 209–225.

[38] Françon, Jean: Sur la modélisation informatique de l'architecture et du développement des végétaux. Université Louis Pasteur, Strasbourg, Département d'Informatique, Document no. R90/12 (1990), 1–19.

[39] Frijters, D., and Lindenmayer, Aristid, A model for the growth and flowering of Aster novae-angliae on the basis of table $< 1, 0 >$ L-systems. *Lecture Notes in Computer Science,* **15**. L Systems. Eds.: Grzegorz Rozenberg, Arto Salomaa. Berlin 1974, 24–52.

[40] Früh, Thomas, C-Bilanz auf der Basis der Pipe-Modell-Theorie, Produktionsmodell. *Berichte des Forschungszentrums Waldökosysteme,* **B 31**. Stabilitätsbedingungen von Waldökosystemen. Zwischenbericht von 1989 bis 1991, Göttingen 1992, 355–367.

[41] Früh, Thomas: Dissertation (in Vorbereitung). Abteilung für forstliche Biometrie und Informatik, Göttingen.

[42] Goel, Narendra S.; Knox, Lee B., and Norman, John M., From artificial life to real life: Computer simulation of plant growth. *International Journal of General Systems,* **18** (1991), 291–319.

[43] Goel, Narendra S., and Rozehnal, Ivan, Some non-biological applications of L-systems. *International Journal of General Systems,* **18** (1991), 321–405 + color plates.

[44] Gravenhorst, Gode; Myneni, Ranga B., and Metzing, Thomas, Strahlungsfeld und Mikroklima in Pappelanpflanzungen. Abschlußbericht BMFT-Programm "Biotechnologische Erzeugung und Verwertung von Industrieholz", Teilprojekt 2.1.2.3. (1. 7. 1987 – 30. 6. 1990), Förderkennzeichen 0318938A, Göttingen 1990.

[45] Gravenhorst, Gode; Knyazikhin, Yuri; Kranigk, Jörn, and Schulze, Dietmar, Abschlußbericht zum BMFT-Teilprojekt P6.3.3.4: Strahlungsverteilung

und bioklimatologische Standortfaktoren in der Langen Bramke. *Berichte des Forschungszentrums Waldökosysteme,* **B 37**. Göttingen 1994, 421–430.

[46] Greene, Ned, Voxel space automata: Modeling with stochastic growth processes in voxel space. *Computer Graphics (ACM/SIGGRAPH),* **23 (3)** (1989), 175–184.

[47] Gruber, Franz: Das Verzweigungssystem und der Nadelfall der Fichte (*Picea abies* (L.) Karst.) als Grundlage zur Beurteilung von Waldschäden. *Berichte des Forschungszentrums Waldökosysteme/Waldsterben,* **A 26**. Göttingen 1987.

[48] Gruber, Franz: Beiträge zum morphogenetischen Zyklus der Knospe, zur Phyllotaxis und zum Triebwachstum der Fichte (*Picea abies* (L.) Karst.) auf unterschiedlichen Standorten. *Berichte des Forschungszentrums Waldökosysteme/Waldsterben,* **A 25**. Göttingen 1987.

[49] Gruber, Franz: Dynamik und Regeneration der Gehölze. Habilitationsschrift. Forstbotanisches Institut der Universität Göttingen 1991.

[50] Hallé, Francis, Modular growth in seed plants. *Phil. Trans. Royal Soc. London, B,* **313** (1986), 77–87.

[51] Harper, John L., and Bell, Adrian D., The population dynamics of growth form in organisms with modular construction. Population Dynamics (Eds.: R. M. Anderson, B. D. Turner, L. R. Taylor), Oxford 1979, 29–52.

[52] Hauhs, Michael: Der Wasser- und Stoffhaushalt von Waldökosystemen aus informationstheoretischer Sicht. Habilitationsschrift, Institut für Bodenkunde und Waldernährung, Göttingen 1991.

[53] Hauhs, Michael; Rost-Siebert, K.; Kastner-Maresch, A., and Lange, H.: A new model relating forest growth to input and output fluxes of energy and matter of the corresponding ecosystem: TRAGIC (Tree response to acidification of groundwater in catchments). Final report of EC project EV4V-0032-D(B), Institut für Bodenkunde und Waldernährung, Göttingen 1993.

[54] Henderson, Robin; Ford, E. David; Renshaw, E., and Deans, J. D., Morphology of the structural root system of Sitka spruce. 1. Analysis and quantitative description. *Forestry,* **56** (1983), 121–135.

[55] Henderson, Robin; Ford, E. David, and Renshaw, E., Morphology of the structural root system of Sitka spruce. 2. Computer simulation of rooting patterns. *Forestry,* **56** (1983), 137–153.

[56] Hoffmann, Friedrich, FAGUS, a model for growth and development of beech. *Ecological Modelling* (submitted).

[57] Honda, Hisao; Tomlinson, P. B., and Fisher, Jack B., Computer simulation of branch interaction and regulation by unequal flow rates in botanical trees. *Amer. J. Botany,* **68** (1981), 569–585.

[58] Host, George E.; Rauscher, H. Michael; Isebrands, J. G., and Michael, Donald A., Validation of photosynthate production in ECOPHYS, an ecophysiological growth process model of Populus. *Tree Physiology,* **7** (1990), 283–296.

[59] Houllier, François, et Leban, Jean-Michel: Modèle théorique de croissance des arbres en peuplement equienne et monospecifique. ENGREF-INRA, Dynamique des Systèmes forestiers - INRA, Station de Recherches sur la Qualité des Bois, Champenoux, Document no. 01/91 (1991), 1–13.

[60] Jaeger, Marc: Représentation et simulation de croissance des végétaux. Thèse, Université Louis Pasteur, Strasbourg, Département d'Informatique, No. d'ordre 328 (1987).

[61] Jaeger, Marc, and Reffye, Philippe de, Basic concepts of computer simulation of plant growth. *Journal of Biosciences,* **17** (1992), 275–291.

[62] Jürgensen, H., Probabilistic L systems. Automata, Languages, Development. Eds.: A. Lindenmayer and G. Rozenberg. Amsterdam 1976, 211–225.

[63] Jürgensen, H., and Matthews, D. E., Stochastic 0L systems and formal power series. The Book of L. Eds.: G. Rozenberg, A. Salomaa. Berlin 1986, 167–177.

[64] Kaye, Brian H.: A Random Walk Through Fractal Dimensions. VCH, Weinheim 1989.

[65] Kiester, A. Ross, et al.: Simple Whole Tree. NAPAP Report 17. Development and Use of Tree and Forest Response Models. Acidic Deposition: State of Science and Technology, Vol. III, Report 17. Principal Author: A. Ross Kiester. Washington, D.C. 1990, 35–127.

[66] Knyazikhin, Yuri; Marshak, A.; Schulze, Dietmar; Myneni, Ranga B. et al., Optimization of solar radiation input in forest canopy as a tool for planting / cutting of trees. *Transport Theory and Statistical Physics,* **23** (1994), 671–700.

[67] Kobayashi, Shogo, Studies on the simulation model of stand growth of japanese larch (*Larix leptolepis* Gord.) plantation. *Bulletin of the Hokkaido Forest Experiment Station,* **15**, Supplement. March 1978, Bibai, Hokkaido.

[68] Kranigk, Jörn, und Gravenhorst, Gode, Ein dreidimensionales Modell für Fichtenkronen. *Allgemeine Forst- und Jagdzeitung,* **164** (1993), 146–149.

[69] Kupka, Ingbert, Van Wijngaarden grammars as a special information processing model. *Lecture Notes in Computer Science,* **88** (1980), 387–401.

[70] Kurth, Winfried, Morphological models of plant growth: Possibilities and ecological relevance. ISEM's 8th International Conference on the State-of-the-Art in Ecological Modelling, 28. 9. - 2. 10. 1992, Kiel. *Ecological Modelling,* **75/76** (1994), 299–308.

[71] Lallement, Gérard: Semigroups and Combinatorial Applications. Wiley, New York 1979.

[72] Langton, Christopher G. (Ed.): Artificial Life. Proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems held Sept. 1987 in Los Alamos (New Mexico). Santa Fe Institute Studies in the Sciences of Complexity, Vol. VI. Addison-Wesley, Redwood City (Cal.) 1989.

[73] Langton, Christopher G.; Taylor, Charles; Farmer, J. Doyne, and Rasmussen, Steen (Eds.): Artificial Life II. Proceedings of the Workshop on Artificial Life held February, 1990 in Santa Fe (New Mexico). Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Vol. X. Addison-Wesley, Reading (Mass.) 1992.

[74] Lanwert, Dirk: Morphologische Untersuchungen an den Kronen junger Fichten (*Picea abies* (L.) Karst.) und ihre Darstellung in einem auf CAD basierenden Modell. Diplomarbeit, Abteilung für forstliche Biometrie und Informatik, Göttingen 1994.

[75] Ledig, Stéphane: Représentation graphique d'épicéas. INRA, Centre de Nancy, Recherche Forestière. Mention Complémentaire en informatique pour l'Instrumentation Scientifique et Médicale. Année 1990–1991 (1991).

[76] Lindenmayer, Aristid, Adding continuous components to L-systems. *Lecture Notes in Computer Science,* **15**. L Systems. Eds.: Grzegorz Rozenberg, Arto Salomaa. Berlin 1974, 53–68.

[77] Lindenmayer, Aristid, Developmental algorithms for multicellular organisms: A survey of L-systems. *Journal of Theoretical Biology,* **54** (1975), 3–22.

[78] Long, Charles A., Leonardo da Vinci's rule and fractal complexity in dichotomous trees. *Journal of Theoretical Biology,* **167** (1994), 107–113.

[79] Lück, Jacqueline, and Lück, Hermann B., From 0L and 1L map systems to indeterminate and determinate growth in plant morphogenesis. *Lecture Notes in Computer Science,* **291** (1987), 393–410.

[80] Magnenat-Thalmann, Nadia, and Thalmann, Daniel: Image Synthesis. Springer, Tokyo 1987.

[81] Maillette, Lucie, Structural dynamics of silver birch. I. The fates of buds. *Journal of Applied Ecology,* **19** (1982), 203–218.

[82] Mandelbrot, Benoit B.: The Fractal Geometry of Nature. W. H. Freeman, New York 1982.

[83] Mattheck, Claus, Why they grow, how they grow: The mechanics of trees. *Arboricultural Journal,* **14** (1990), 1–17.

[84] McMahon, Thomas A., and Kronauer, Richard E., Tree structures: Deducing the principle of mechanical design. *Journal of Theoretical Biology,* **59** (1976), 443–466.

[85] Möhring, Bernhard: Über den Zusammenhang zwischen Kronenform und Schneebruchanfälligkeit bei Fichte. Diplomarbeit, Institut für Forsteinrichtung und Ertragskunde, Universität Göttingen 1980.

[86] Morelli, R. A.; Walde, R. E.; Akstin, E., and Schneider, C. W., L-system representation of speciation in the red algal genus *Dipterosiphonia* (Ceramiales, Rhodomelaceae). *Journal of Theoretical Biology,* **149** (1991), 453–465.

[87] Neureither, Matthias: Modellierung geometrisch-topologischer Daten zur Beschreibung und Berechnung netzartiger und flächenhafter Strukturen. Dissertation, Universität Stuttgart, Fakultät 2 für Bauingenieur- und Vermessungswesen. Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften, Reihe C, Heft 387, München 1992.

[88] Nishida, Taishin, K0L-system simulating almost but not exactly the same development. *Memoirs of the Faculty of Science, Kyoto Univ., Ser. Biol.,* **8** (1980), 97–122.

[89] O'Neill, R. V.; de Angelis, D. L.; Waide, J. B. and Allen, T. F. H.: A Hierarchical Concept of Ecosystems. Princeton University Press, New Jersey 1986.

[90] Pagès, Loïc, et Ariès, Franck, SARAH: Modèle de simulation de la croissance, du développement et de l'architecture des systèmes racinaires. *Agronomie,* **8** (1988), 889–896.

[91] Pagès, Loïc, and Kervella, Jocelyne, Growth and development of root systems: Geometrical and structural aspects. *Acta Biotheoretica,* **38** (1990), 289–302.

[92] Perterer, J., und Körner, Ch., Das Problem der Bezugsgröße bei physiologisch-ökologischen Untersuchungen an Koniferennadeln. *Forstwissenschaftliches Centralblatt,* **109** (1990), 220–241.

[93] Perttunen, Jari; Saarenmaa, Hannu; Sievänen, Risto; Salminen, Hannu et al., Propagating the effects of herbivore attack in an object-oriented model of

a tree. W. J. Mattson (Ed.), Proc. of the IUFRO symposium “Mechanisms of Plant Resistance against Herbivores”, Maui (Hawaii), 2.–6. 2. 1994 (in print).

[94] Pfreundt, Jürgen: Modellierung der räumlichen Verteilung von Strahlung, Photosynthesekapazität und Produktion in einem Fichtenbestand und ihrer Beziehung zur Bestandesstruktur. *Berichte des Forschungszentrums Waldökosysteme/Waldsterben,* **A 39**. Göttingen 1988.

[95] Porter, J. R., Modules, models and meristems in plant architecture. G. Russell, B. Marshall, P.G. Jarvis (Eds.), Plant Canopies: Their Growth, Form and Function. Cambridge 1989, 143–159.

[96] Pretzsch, Hans, Analyse der Bestandesstruktur als Grundlage für die Entwicklung eines Wachstumssimulators für Mischbestände. *Tagungsberichte der Arbeitsgruppe Biometrie in der Ökologie,* **1** (1990). AG-Tagung beim 35. Biometrischen Kolloquium in Celle, 1. 3. 1989, und 2. Herbstkolloquium in Wuppertal, 28.–29. 9. 1989, 52–71.

[97] Prusinkiewicz, Przemyslaw, and Hanan, James: Lindenmayer Systems, Fractals, and Plants. *Lecture Notes in Biomathematics,* **79**. Springer, New York 1989.

[98] Prusinkiewicz, Przemyslaw, and Lindenmayer, Aristid: The Algorithmic Beauty of Plants. Springer, New York 1990.

[99] Reffye, Philippe de, et Snoeck, J., Modèle mathématique de base pour l’étude et la simulation de la croissance et de l’architecture du Coffea robusta. *Café Cacao Thé,* **20** (1976), 11–32.

[100] Reffye, Philippe de: Modélisation de l’architecture des arbres par des processus stochastiques. Simulation spatiale des modèles tropicaux sous l’effet de la pesanteur. Application au *Coffea robusta.* Thèse, Université de Paris-Sud, Centre d’Orsay, No. d’ordre 2193 (1979).

[101] Reffye, Philippe de; Blaise, Frédéric; Houllier, François; Fourcaud, Th., et Castera, P., Essai sur les relations entre l’architecture d’un arbre et la grosseur de ses axes végétatifs. Colloque INRA - CIRAD “Modélisation et Simulation de l’Architecture des Arbres Fruitiers et Forestiers”, 23.–25. 11. 1993, Montpellier.

[102] Reffye, Philippe de; Cognée, M.; Jaeger, Marc, et Traoré, B., Modélisation stochastique de la croissance et de l’architecture du cotonnier. I. Tiges principales et branches fructifères primaires. *Cot. Fib. Trop.,* **43** (1988), 269–291.

[103] Reffye, Philippe de; Dinouard, Pierre, and Jaeger, Marc, Basic concepts of computer plants growth simulation. NICOGRAPH’90, Seminar-9; Computer

Graphics: "Where do we go now that we've arrived?", Tokyo, Nov. 1990, 219–234.

[104] Reffye, Philippe de; Dinouard, Pierre, et Barthélémy, Daniel, Modélisation et simulation de l'architecture de l'orme du japon *Zelkova serrata* (Thunb.) Makino (*Ulmaceae*): La notion d'axe de référence. *Naturalia Monspeliensia, h.s.* L'Arbre. Biologie et Developpement. Ed.: C. Edelin. Montpellier 1991, 251–266.

[105] Reffye, Philippe de; Elguero, E., and Costes, Evelyne, Growth units construction in trees: A stochastic approach. *Acta Biotheoretica,* **39** (1991), 325–342.

[106] Rigaut, Jean-Paul, Fractals, semi-fractals et biométrie. Fractals — Dimensions Non Entières et Applications. Ed.: G. Cherbit. Paris 1987, 231–281.

[107] Roloff, Andreas, oral communication.

[108] Salminen, H.; Saarenmaa, H.; Perttunen, J.; Sievänen, R. et al., Modelling trees with an object oriented scheme. *Resource Technology*, submitted.

[109] Salomaa, Arto: Computation and Automata. Cambridge University Press, Cambridge 1985.

[110] Seibt, G.: Die Buchen- und Fichtenbestände der Probeflächen des Sollingprojektes der Deutschen Forschungsgemeinschaft. *Schriften aus der Forstlichen Fakultät der Universität Göttingen und der Niedersächsischen Forstlichen Versuchsanstalt,* **72**. J.D. Sauerländer's Verlag, Frankfurt a.M. 1981.

[111] Sloboda, Branislav, and Pfreundt, Jürgen, Tree and stand growth. Artificial Intelligence and Growth Models for Forest Management Decisions. Publ. No. FWS-1-89, School of Forestry and Wildlife Resources, Blacksburg, Virginia 1989, 119–153.

[112] Smith, Alvy Ray, Plants, fractals, and formal languages. *Computer Graphics (ACM/SIGGRAPH),* **18 (3)** (1984), 1–10.

[113] Smith, Harry F., A garden of fractals. Fractals in the Fundamental and Applied Sciences (Eds.: H.-O. Peitgen, J. M. Henriques, L. F. Penedo), Amsterdam 1991, 407–424.

[114] Stickan, Walter; Gansert, Dirk; Neemann, Gerd, and Rees, Ulrich, Modelling the influence of climatic variability on carbon and water budgets of beech saplings (*Fagus sylvatica* L.) based on field data. ISEM's 8th International Conference on the State-of-the-Art in Ecological Modelling, 28. 9. - 2. 10. 1992, Kiel. *Ecological Modelling*, in press.

[115] Tragut, Vincenz, Graphtale Pflanzen. *c't,* **1989 (4)**, 210–225.

[116] Ulrich, Bernhard, Process hierarchy in forest ecosystems — an integrating ecosystem theory. A. Hüttermann / D. Godbold (Eds.), Effects of Acid Rain on Forest Processes. New York 1993.

[117] Viennot, Xavier Gérard; Eyrolles, Georges; Janey, Nicolas, and Arquès, Didier, Combinatorial analysis of ramified patterns and computer imagery of trees. *Computer Graphics (ACM/SIGGRAPH),* **23 (3)** (1989), 31–40.

[118] Viennot, Xavier Gérard, Trees everywhere. *Lecture Notes in Computer Science,* **431** (1990), 18–41.

[119] Waller, Donald M., and Steingraeber, David A., Branching and modular growth: Theoretical models and empirical patterns. Symposium on the Biology of Clonal Organisms, Yale University, 15. - 17. 2. 1982. Eds.: L. Buss, R. Cook and J. Jackson. Yale University Press 1983, 85–108.

[120] Zeide, Boris, and Pfeifer, Peter, A method for estimation of fractal dimension of tree crowns. *Forest Science,* **37** (1991), 1253–1265.

Index

Author's address:

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