VERSITA

Lesnícky časopis - Forestry Journal
volume 58, number 2, 2012, p. 75-91
Section: Forestry
DOI: 10.2478/v10114-011-0012-0

Pôvodné práce – *Original papers*

FOREST STRUCTURE, COMPETITION AND PLANT-HERBIVORE INTERACTION MODELLED WITH RELATIONAL GROWTH GRAMMARS

Winfried Kurth[1], Ole Kniemeyer[2] and Branislav Sloboda[1]

[1]*University of Göttingen, Department Ecoinformatics, Biometrics and Forest Growth, Büsgenweg 4, D – 37077 Göttingen, Germany, e-mail: wk@informatik.uni-goettingen.de*
[2]*MAXON Computer GmbH, Max-Planck-Str. 20, D – 61381 Friedrichsdorf, Germany*

Dedicated to Prof. Ing. Štefan Šmelko, DrSc.
an international expert in forest management and forest biometrics
who promoted partnership between Technical University Zvolen
and Georg-August University Göttingen.

Kurth, W., Kniemeyer, O. and Sloboda, B., 2011: Modelovanie štruktúry lesných porastov, konkurencie a interakcie medzi herbivormi pomocou relačných rastových tabuliek. Lesn. Čas. – Forestry Journal, **58**(2): 75–91, 2012, fig. 12, tab. 4, ref. 40, ISSN 0323 – 1046. Original paper.

Relačné rastové gramatiky sú systémy prepisovacích pravidiel (grafové gramatiky) s grafickou interpretáciou. Umožňujú spätnú väzbu z vytvorených virtuálnych 3D štruktúr do ďalšieho procesu aplikácie pravidiel. Ich použitím je možné kombinovať morfologické (geneticky viazané) rastové pravidlá s environmentálnym vplyvom a s funkciami hodnotiacimi konkurenčnú situáciu jednotlivých rastlín. Relačné rastové gramatiky sú preto ideálnym nástrojom na presnú špecifikáciu funkčno-štrukturálnych modelov rastu a architektúry rastlín. Dynamika vývoja porastu v takýchto modeloch vyplýva z čisto lokálnej aplikácie pravidiel. Predbežné výsledky sú ukázané na troch aplikáciách modelovania lesného ekosystému: (a) Tvorba nepravidelných porastových štruktúr, (b) simulácia vplyvov konkurencie na vývoj polomeru koruny a výslednú dynamiku porastu a (c) modelovanie interakcie medzi stromami a herbivormi, založené na energetických nárokoch individuálnych rastlín a živočíchov. Posledný model zahŕňa genetický prenos a evolúciu potravnej stratégie živočíchov. Softvérový systém GroIMP (Growth-grammar related Interactive Modelling Platform), open source projekt prístupný na www.grogra.de, bol navrhnutý na interpretáciu relačných rastových gramatík v objektovo orientovanom rámci. Tiež slúži na vizualizáciu výsledných priestorových štruktúr. Kód, spustiteľný v GroIMPe, je pre vyššie spomenuté modely kompletne dokumentovaný a vysvetlený. Dúfame, že uvedenými príkladmi budeme motivovať čitateľov k používaniu na pravidlách založených štrukturálnych modelov v ekológii lesa.

Kľúčové slová: *konkurencia, štruktúra, herbivory, model, gramatika, FSPM (functional-structural plant model)*

Relational Growth Grammars are systems of rewriting rules (graph grammars) with graphical interpretation. They allow a feedback from the created virtual 3-d structures to the subsequent rule-application process. Using them it is possible to combine morphological (genetically fixed) growth rules with environmental impact and with functions evaluating the competitive situation of individual plants. Relational Growth Grammars are thus an ideal tool for precise specification of functional-structural models of plant growth and architecture. The dynamics of stand development in such models results from

purely local rule application. Preliminary results are shown for three applications in forest-ecosystem modelling: (a) Creation of irregular stand structures, (b) simulation of competitive effects on crown radius development and resulting stand dynamics, and (c) modelling the interaction between trees and herbivores, based on the energy budgets of the individual plants and animals. The latter model includes genetical transfer and evolution of the foraging strategy of the animals. The software system GroIMP (Growth-grammar related Interactive Modelling Platform), an open source project available under www.grogra.de, was designed to interpret Relational Growth Grammars in an object-oriented framework. It also serves to visualize the resulting spatial structures. The code, executable by GroIMP, for the above-mentioned models is completely documented and explained. By our examples, we hope to motivate the readers to use rule-based structural models in forest ecology.

Keywords: *competition, structure, herbivores, model, grammar, FSPM (functional-structural plant model)*

1. Introduction

With the advance of information technology, simulations of large and complex biological systems, formerly considered intractable because of the amount of necessary calculation time and memory space, became possible. As a consequence, individual-based simulation models which are expected to reproduce and to predict forest structure and stand dynamics have received increasing attention in recent years (DeAngelis, Gross, 1992; Grimm, Railsback, 2005; Pretzsch 2001, 2009). Among different types of ecosystems, natural forests are characterized by a particularly high degree of spatial heterogeneity and complex structure. Moreover, changing the spatial structure is the main method used by foresters to manipulate the development of forest stands and of individual trees (Bormann, Likens, 1979). Individual-based models have the advantage that spatial structures of competing trees can be represented in a natural way.

When a researcher is confronted with the large number of existing complex models, there arises the need for short and precise model specifications. Whereas simple models of whole-stand dynamics, disregarding spatial structure, can often be expressed in terms of one or several equations, models involving spatial details usually need computer source code, written in a standard programming language (such as Fortran, C, C++ or Java), for their full specification. However, classical source code does usually involve many technical constructions distracting attention from the essentials of the model, and cannot be understood easily by users who are not computer scientists or professional programmers. Furthermore, the requirements of generalness and modular design of software (cf. Acock, Reynolds, 1997) are often violated by ad-hoc models implemented by biologists or agronomists who lack specific training in software development.

A way to overcome these difficulties is the design of a higher-level model specification language, adapted to the particular needs of tree and stand simulation and spatial interaction. When model specifications written in this language can be read and interpreted by a generic software, there will be no need to modify and re-compile the source code of the software each time some assumptions or relations in the model are changed. Instead,

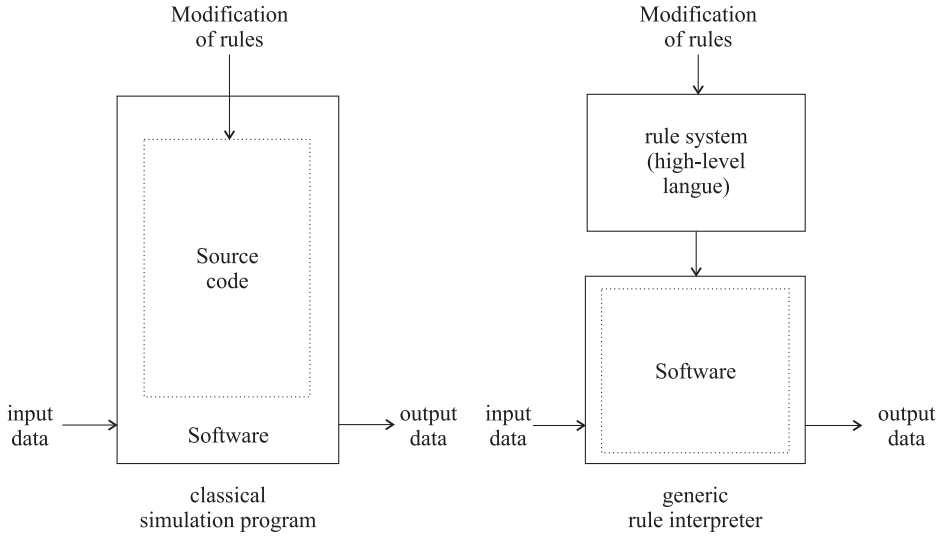

Fig. 1. Comparison of the architecture of a classical simulation model (left side), where each modification requires rewriting of the software source code, with a generic software shell for an advanced model-specification language (like Relational Growth Grammars; right side)

only the specifications made in the high-level language are to be modified, and these can be made open to "informed users" other than computer scientists (Figure 1). Comparisons of different models are easier if the basic software with its technical details remains the same.

A candidate formalism which could adopt the role of such a high-level specification language is the cellular automaton (CA). A CA is specified by a transition function which determines the state of a cell in a grid from the previous states of the same cell and its neighbours. CA has been used for a number of ecological models (Ermentrout, Edelstein-Keshet, 1993), including forest models. However, their inherent preferential treatment of certain directions, spatial and temporal scales restricts their use.

A different general specification language for biological growth processes is the rule-based language of L-systems (Lindenmayer systems, named after the botanist Aristid Lindenmayer, 1925–1989). Originally devised to resemble growth rules of simple, filamentous plants (Lindenmayer, 1968), numerous extensions have been added to the formalism since then (Prusinkiewicz *et al.,* 1997; Kurth, 1999, Kurth, Sloboda, 1999a). However, the main field of application of this language is still the specification of the architecture of individual plants.

L-systems have been extended to a formalism termed Relational Growth Grammars (RGGs) (Kniemeyer *et al.,* 2004; Kurth *et al.,* 2005; Kurth, 2007). In contrast to L-systems, which operate on strings, RGGs transform general networks (graphs) and are thus more powerful. Furthermore, we have embedded RGGs into a full-scale object-oriented programming language called XL (eXtended L-system language). It enables a smooth combination of rule-based and classical (imperative) constructions (Kniemeyer, 2008; Kurth *et al.,* 2005; Kurth, Lanwert, 2011). Simulation models specified in XL can be compiled and run by the software GroIMP (Growth-grammar related Interactive Modelling Platform), which is available at www.grogra.de as an open-source project (Kniemeyer *et al.,* 2007).

In this paper, we aim to explore the possibilities of RGGs (expressed in the programming language XL) for spatial simulations at a lower scale of resolution. In our examples, the architecture of single trees will be highly simplified, but their spatial arrangement, competition and interaction with mobile herbivores will be taken into account. In the same way as for L-system based models of architectural development of single trees, the dynamics of development will result from purely local rule application. That means, no global curve of self-thinning or other aggregated description of stand growth is taken as input. Instead, the overall behaviour of the model will result as an emergent property from local rules (cf. Breckling, 1996).

2. Theoretical Background

2.1. L-Systems

An L-system consists of a set of symbols, a start symbol *a* and a set of *replacement rules*, each of the form "symbol → string of symbols". Additionally, there is a *geometrical interpretation* of the strings (i.e., a semantics) translating strings of symbols into structures in 3-d space. Usually, this interpretation is given by the conventions of *turtle geometry* (Abelson, diSessa, 1982): Some symbols are used as commands for a virtual device (the turtle) which is able to move (command "`M`"),

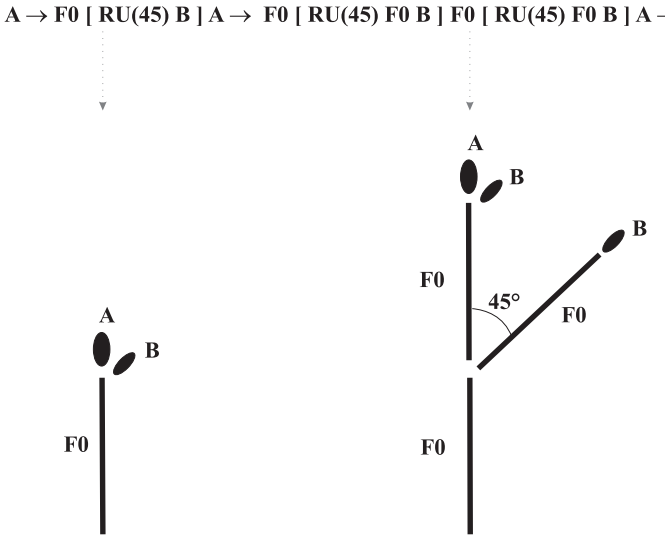

Fig. 2. Development of geometrical structures specified by the growth grammar `A ==> F0 [ RU(45) B ] A; B ==> F0 B.` Horizontal arrows stand for rule application, dotted vertical arrows for interpretation of the strings by the turtle

to produce cylindric elements while moving forward (command "F"), to rotate (command "RU") or to change internal parameters used for the next elements to be produced (commands "L" for length, "D" for diameter, and more; see KURTH (1994) for details). Brackets [...] are interpreted as delimiters for branches.

The rules of the L-system are applied in parallel to all symbols of a string at time *t* in order to get a new string at time *t*+1. This rewriting process is normally iterated several times. Thus one gets a (potentially infinite) sequence of turtle command strings s_0, s_1, s_2, ... Here, s_{t+1} is obtained from s_t by application of the rules, and $s_0 = a$. The string s_t is interpreted in terms of turtle geometry, resulting in a geometrical model of a single plant or stand at time *t*. By proceeding in discrete time steps, we obtain a developmental sequence of geometrical structures. In models of individual plants, the time step often corresponds to 1 year or even to shorter periods of growth. However, in our application examples the time step will represent a period of several years of stand development. Figure 2 shows the strings and geometrical structures resulting from the application of a very simple, classical L-system describing the growth of a branching system. The start symbol is "A". The L-system has only two rules: `A ==> F0 [ RU(45) B ] A` and `B ==> F0 B`. The symbols A and B, standing for apical buds of main and lateral branches, respectively, are normally not interpreted by the turtle. In Figure 2, we have visualized the corresponding buds by ovals. The L-system contains one symbol with a parameter: RU, with the subsequent number specifying the rotation angle in degrees. Parameters can also be attached to other symbols like A and B (PRUSINKIEWICZ, LINDENMAYER, 1990). F0 is a turtle command for movement and branch construction which has no (i.e., zero) parameters, in contrast to the F command in XL which expects (at least) the length of the next shoot as a parameter. Further commands will be explained in the subsequent examples where they are needed.

2.2. Relational Growth Grammars (RGGs)

RGGs are a proper extension of L-systems. All L-systems can also be realised as RGGs in the programming language XL. In an RGG, the symbols are interpreted as nodes of a graph. These nodes can be connected by arbitrary edges. In our examples, we use mostly one special type of edges, which correspond to the "successor" relation in L-system strings. They are automatically generated when a "blank" symbol is read between two symbols standing for nodes in XL. This convention enables a direct embedding of L-system notation in XL code.

RGGs in the language XL generalise L-systems in the following aspects:

– More than one symbol (node) can be replaced in a rule,
– further geometrical objects (polygons, spheres, cones, boxes, spline surfaces) are available as node types and can be used for modelling,
– the standard node types like `F, Sphere` etc. can be extended by additional, user-defined parameters (like, e.g., carbon content),
– queries for certain node types, e.g. (`* Tree *`) for all nodes of type "Tree", can be used to search in the structure generated during a simulation and, e.g., to count all trees or to calculate their total biomass,
– the order of the application of rules can be controlled using blocks of rules, conditions and loops,
– functions written in the object-oriented programming language Java (GOSLING *et al.,* 2005) can be used for calculations, and Java code enclosed in braces `{...}` can be inserted in the right-hand side of rules. This enables an execution of conventional, imperative code each time when the rule is applied. Particularly, a "sensitivity" of growing elements with respect to the environment or to competing objects in the neighbourhood can thus be simulated.

These extensions will be further clarified when they are used in the examples below. A complete language description of XL, including a precise definition of RGGs, is given by KNIEMEYER (2008). All pieces of code presented in the given examples will be readable by the software GroIMP. We refer to KURTH (2007), to the menu item "examples" in the software GroIMP itself and to www.grogra.de for further simple examples.

3. First Example: Specification of Irregular Stand Structures

The first application of the grammar formalism at stand level is the specification of patterns of tree positions, disregarding dynamic aspects. Models of stand structure have often used stochastic approaches like point processes (PENTTINEN *et al*., 1994) or heuristic algorithms (LEWANDOWSKI *et al.,* 1997). Extended L-systems provide a framework for a transparent specification of such models. Let us first focus on tree positions only, disregarding all other attributes such as height, diameter, tree species etc. The simplest grammar rule for an irregular stand generates a random pattern of tree seedlings:

```
Stand ==> for ((1:n))
( [move to random position Seedling
(random(10, 20)) ]);
```

where n is the number of seedlings dispersed over the stand area and `Seedling(length)` an object standing for a seedling of height "`length`". Firstly, this "seedling" node type must be defined as a user-defined extension of a standard node type, let us say, F (which generates an object of cylindrical shape):

```
module Seedling(super.length) extends
F(length, 5, 4);
```

where the "5" in the specification of `F` stands for a fixed diameter and "4" for a colour index. "`super.length`" means that the parameter "length" is taken from the "superior" (more general) node type `F`.

In our rule above, for `length` a uniformly distributed random number between 10 and 20 was inserted, making the seedlings vary in their height between these limits. Movement to a random position within, let us say, a rectangular area of extensions x_{extens} and y_{extens} can be specified by a translation command:

```
Translate(random(0, x_extens),
random(0, y_extens), 0)
```

which uses random numbers for x and y and does not make use of the third dimension ($z = 0$). When this command (which specifies, technically, also a node of the graph) is inserted into the above replacement rule instead of "*move to random position*", the result will be a random distribution of seedling positions with uniform distribution of x and y coordinates (i.e., a "Poisson forest").

However, during growth, close neighbours will normally be outcompeted. To obtain a pattern where a minimum distance between the trees is respected, we can modify the grammar by introducing a function `notOutcompeted`, which checks if in a certain neighbourhood of a seedling another seedling with greater "`length`" parameter (i.e., with greater height) occurs. This must be a function of type "boolean" (i.e., the result is true or false). The investigation of the neighbourhood is done by a query for all seedlings different from the considered seedling and having a distance smaller than a threshold value, `inhib_r`:

```
boolean notOutcompeted(Seedling s)
  {
  return empty((* t:Seedling, ((t != s) &&
     (distance(s, t) <= inhib_r) &&
     (t[length] >= s[length]) ) *) );
  }
```

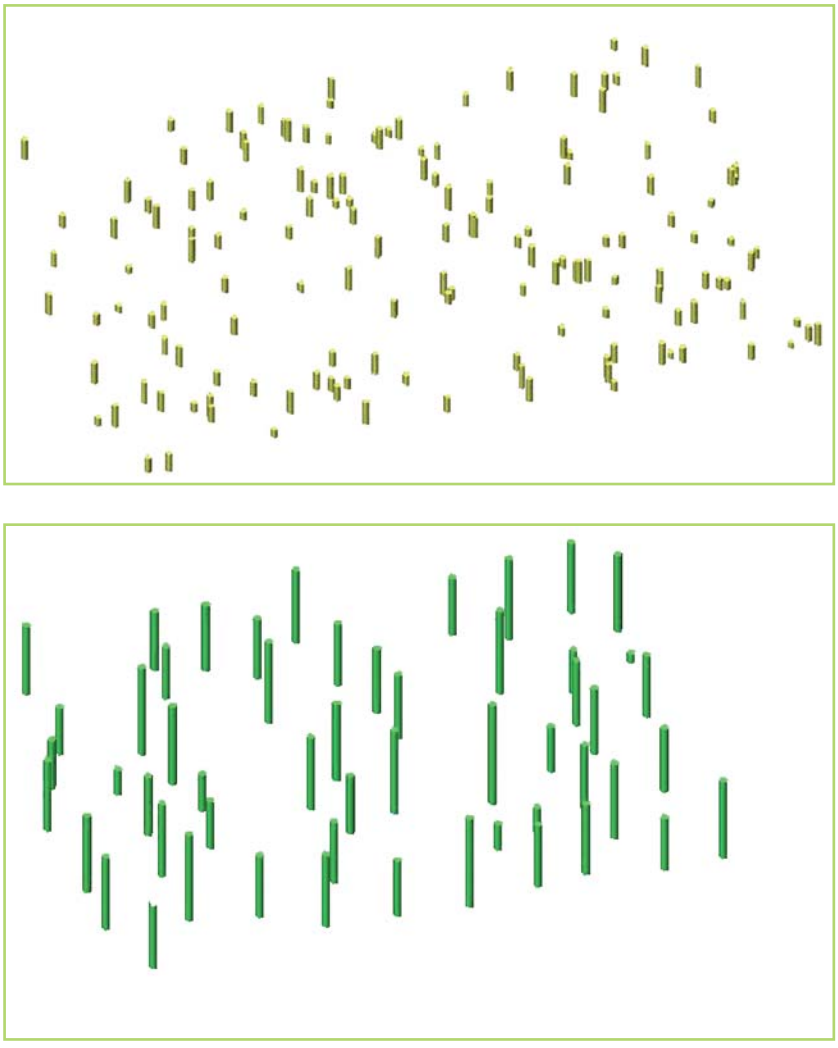

Fig. 3. Result of the growth grammar `irreg` (Table 1) after 2 steps (pattern of seedlings; upper part) and after 3 steps (pattern of mature trees, lower part). Slanted view, parallel projection

Table 1. The Relational Growth Grammar `irreg`

```
/* irreg.rgg: specification of an irregular stand structure
   in the 2D plane with random tree coordinates, but
   close neighbourhoods of larger individuals excluded */

module Stand;
module Seedling(super.length) extends F(length, 5, 4);
module Tree(super.length) extends F(length, 7, 2);

int n = 150;            /* initial number of seedlings */
double x_extens = 500;  /* extension of stand */
double y_extens = 350;
double inhib_r = 35;    /* distance which inhibits growth */
double factor = 4;      /* proportionality of seedling
                           and tree height */

boolean notOutcompeted(Seedling s)
   {
   return empty( (* t:Seedling,
      ( (t != s) && (distance(s, t) <= inhib_r) &&
        (t[length] >= s[length]) ) *) );
   }

protected void init()
   [ Axiom ==> Stand; ]

public void make()
   [
   Stand ==> for ((1 : n))
      ( [ Translate(random(0, x_extens), random(0, y_extens), 0)
          Seedling(random(10, 20)) ] );

   s:Seedling(h) ==> if (notOutcompeted(s) &&
      s[Location.X] >= 0 && s[Location.Y] >= 0 &&
      s[Location.X] <= x_extens && s[Location.Y] <= y_extens)
           ( Tree(factor * h + normal(0, 15)) )
      else ();
   ]
```

The seedling which is under consideration has the name `s`, whereas the potential competitor in the query is labelled with the name `t`. The symbols "`&&`" stand in Java (Gosling *et al.,* 2005) for the logical "and" operation. It is used to combine the three conditions for the existence of a dominating competitor, namely, that it is different from `s`, that its distance is below the threshold, and that its height is larger than that of `s`.

By the conditional rule

```
s:Seedling(h) ==>  if (notOutcompeted(s))
                      ( Tree(factor*h +
                      normal(0, 15)) )
                   else ();
```

we can specify that the development of the seedling to a mature tree (module "`Tree`") will only take place if there is no higher competitor inside a circle with radius "*inhib_r*". This results in an arrangement of mature trees where positions are random, but no two trees are closer than the minimal distance *inhib_r*. Furthermore, the height of the resulting tree is linearly dependent from that of the seedling, but with a random deviation following a normal distribution with mean 0 and standard deviation 15. This dependency models in a simple way the well-known phenomenon of rank preservation (Sloboda, 1983). "`factor`" is an empirical constant. With the statement "`else ();`", we ensure that outcompeted seedlings disappear.

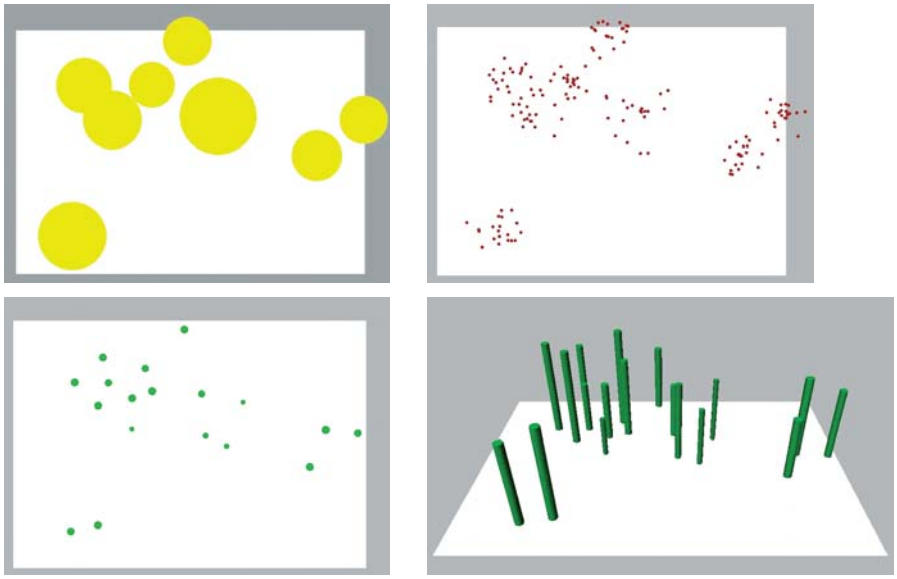

Fig. 4. Result of the growth grammar `irreg2` (Table 2) after 2, 3 and 4 steps, seen from above, and after 4 steps from oblique perspective

Figure 3 shows a typical resulting pattern of seedlings (without minimal distance; upper part) and the corresponding pattern of mature trees after one additional step of rule application (with minimal distance; lower part). The complete XL code with 9 declarations and 3 rules is given in Table 1.

In Table 1, we see the organisation of rules in blocks (delimited by pairs of brackets), here called "`init`" and "`make`", which makes the XL code more transparent. All rules of a block which is declared as "`public`" are automatically associated with an accessible button on GroIMP's RGG-execution panel, thus providing an interactive selection of the group of rules which the user wants to apply. The rules of the "`init`" block, which is declared as "`protected`", are, however, executed automatically when GroIMP loads the RGG file. In this case, there is only one "`init`" rule, which transforms the start node "`Axiom`" into a "`Stand`" node.

Into the second rule of the "`make`" block, which transforms the seedlings into trees, we have inserted additional conditions to ensure that no tree grows beyond the predefined limits of the stand. They just compare the location coordinates of the seedlings with the given threshold values.

A refinement of the model can be made by including a tendency of clustering. Instead of spreading the seedlings directly at random, we can simulate a two-phase stochastic process by first spreading clusters of seedlings:

```
Stand ==> for ((1:n))
   ( [move to random position
   Cluster(random(60, 120)) ] );
```

where the argument of the symbol "`Cluster`" specifies the diameter of a cluster. Then each cluster is expanded into a circular area where seedlings are distributed with random polar coordinates:

```
Cluster(d) ==> for ((1 : sd_per_cl))
   ( [ RH(random(0, 360)) RU(90) M
   (random(0, 0.5) * d)
   RU(-90) Seedling(random(10, 30)) ] );
```

Here, "`sd_per_cl`" is the number of seedlings per cluster, "`RH(random(0,360))`" denotes a random rotation, and "`M(random(0, 0.5)*d)`" a random movement in the interior of the circle with radius *d*/2, where the start point is the centre of the cluster. The turtle commands "`RU`" surrounding this movement command ensure that the randomized filling of the cluster takes place in the horizontal plane. Of course, "`Cluster`" must also be declared as a module; we define it (for the purpose of visualisation) as a flat, yellow disk which inherits its properties again from the standard node type "`F`":

```
module Cluster(super.diameter) extends
F(1, diameter, 14);
```

where 1 is the thickness of the disk and 14 a colour index, standing for "yellow".

Figure 4 shows the resulting distribution of clusters (second step; upper left part of the Figure), seedlings (third step; upper right part) and mature trees (fourth step; lower left part), all seen from above, and a view on the mature trees, represented as cylinders, from an oblique angle (lower right part). The stand area is modelled as a flat, white box; seedlings dispersed outside this area do not grow out to trees.

Similar to the previous example, the trees respect a minimum distance, which is smaller than the cluster ra-

Table 2. The Relational Growth Grammar `irreg2`

```
/* Stand with randomly distributed clusters of trees,
   close neighbourship excluded */

module Stand;
module Cluster(super.diameter) extends F(1, diameter, 14);
module Seedling(super.length) extends F(length, 5, 4);
module Tree(super.length, super.diameter) extends F(length,
       diameter, 2);

int sd_per_cl = 20;     /* number of seedlings per cluster */
int n = 8;              /* number of clusters */
double x_extens = 500;  /* extension of stand */
double y_extens = 350;
double inhib_r = 30;    /* distance which inhibits growth */
double factor = 4;      /* proportionality of seedling
                           and tree height */

boolean notOutcompeted(Seedling s)
   {
   return empty( (* t:Seedling,
      ( (t != s) && (distance(s, t) <= inhib_r) &&
        (t[length] >= s[length])) *) );
   }

protected void init()
   [
   Axiom ==>
      [ Translate(x_extens/2, y_extens/2, -1)
        Box(1, x_extens, y_extens).(setColor(-1)) ] /* soil */
      Stand;
   ]

public void make()
   [
   Stand ==> for ((1 : n))
      ( [ Translate(random(0, x_extens), random(0, y_extens), 0)
          Cluster(random(60, 120)) ] );
   Cluster(d) ==> for ((1 : sd_per_cl))
      ( [ RH(random(0, 360)) RU(90) M(random(0, 0.5) * d) RU(-90)
          Seedling(random(10, 30) ) ]);
   s:Seedling(h) ==> if (notOutcompeted(s) &&
       s[Location.X] >= 0 && s[Location.Y] >= 0 &&
       s[Location.X] <= x_extens && s[Location.Y] <= y_extens)
            ({double hnew = factor * h + normal(0, 20);}
              Tree(hnew, 0.4 * h) )
      else ();
   ]
```

dius. Height and crown radius of the mature tree, which are specified as parameters of the module `Tree`, will now both depend on the height `h` of the seedling. The complete grammar is shown in Table 2.

Instead of directly specifying the `Tree` object by a turtle command `F`, it is also possible to give a more or less complex rule which describes the topological and geometrical structure of a tree. The most simple variant is to compose a few standard objects, e.g., a cylinder for the stem and a cone for the crown (Figure 5a). To specify this geometrical model, there is just one additional rule for the module `Tree` to be inserted into the grammar of Table 2:

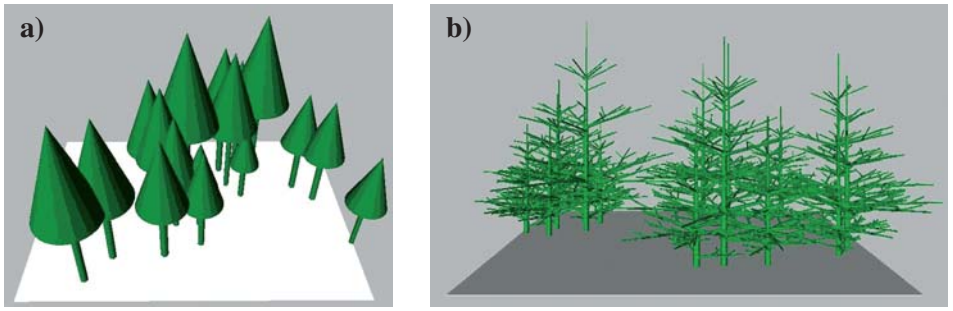

Fig. 5. (a) Result of an extended version of the grammar `irreg2` (see Table 2) where tree architecture is specified by a simple geometrical rule (see text). (b) Result of a similar grammar with trees grown according to L-system rules for spruce from KURTH (1999)

```
Tree(length, diameter) ==>
   F(length, diameter,
   2) Cone(length, length/3);
```

with `F` standing for the stem and `Cone` for the crown. In order to reduce the necessary amount of computer memory for all the tree compartments occurring in the scene, it is possible to use the technique of *object instancing* here: The objects `F` and `Cone` will be generated only once, and instances of them will be inserted into the scene at the position of each `Tree` object only in the moment when the output is generated (see DEUSSEN *et al.* (1998) for a related application of object instancing). For this purpose, the above rule for `Tree` has to be substituted by an *instantiation rule*, which must be specified in the XL code together with the declaration of the module `Tree`:

```
module Tree(float length, float diameter) ==>
   F(length, diameter,
   2) Cone(length, length/3);
```

thus replacing the fourth module declaration in Table 2. The visible result is the same as in Figure 5a.

It is also possible to use a more complex sub-grammar (not shown; see KURTH (1999) for examples) which captures the botanically-correct morphology of the branching structure of a certain tree species; we have tentatively inserted a grammar for spruce (Figure 5b). Combined with terrain data, textures and background images, this approach can be used to generate photo-realistic views of current or predicted forests and landscapes; see KNAUFT (2000) for an earlier study in this direction, using other tools.

4. Second Example: A Model of Crown Radius Dynamics Under Competition

The previous models contained only a very coarse representation of interactions between neighbours. If more detailed information about crown dimensions is available, a graph query can be used to simulate the reaction of the crown radius to the presence of competitors. In an ad-hoc model, i.e., without using a specification in terms of a growth grammar or other higher-level language, this approach was used by PRETZSCH (1992a,b). He described the dynamics of horizontal crown expansion, using 8 predefined directions, in dependence upon distance to neighbouring trees. To formalize this approach, we use a graph query which checks the distance to the next geometrical element, representing a crown sector of another tree, inside a 45° cone along the direction of the considered crown radius. If this distance is large enough, i.e., above a given threshold *ds*, then the considered crown radius continues to grow (parameter *growing* = 1, indicated also by a special colour *c*) and increases its length to *r*+1:

```
s:Sector(growing, r, c) (* ms:Marker *) ==>
   if (empty((* t:Sector mt:Marker,
      ( t != s && distance(ms, mt) <= ds &&
        mt in cone(s, true, 22.5)) *) ))
           ( Sector(1, r+1, 2) )
```

otherwise, a status of "shrinking" (*growing* = 0, *c* = 4) is assumed and the radius is shortened by 0.3 length units:

```
else  ( Sector(0, r-0.3, 4) );
```

Here, the objects called "`Marker`" simply represent the endpoints of the current crown radii. The XL function "`empty`" checks if the set of solutions of the subsequent query, which is enclosed by `(* ... *)`, is empty. It is the first object inside the query, here: the `Sector` object named `t`, which is looked for. This sector must fulfil three conditions in order to be accepted and to make the "`empty`" condition false: (1) it must be different from the crown sector `s` which is under consideration, (2) the distance of its endpoint (represented by the dimensionless marker object `mt`) to the endpoint of `s` must be smaller than the threshold `ds`, (3) its endpoint must be situated inside a cone with its central axis given by `s` and with an opening angle (measured from this axis) of 22.5 degrees.

We can further make the assumption that the crown radii in the status of "shrinking" are counted for each tree, and if this number is 5 or greater, the tree is remo-

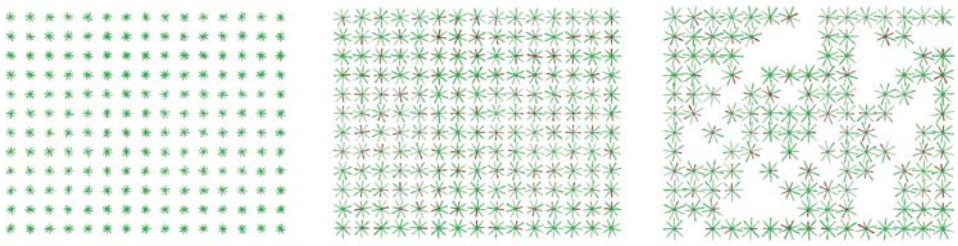

Fig. 6. Results of the growth grammar `radii` 2011: (Table 3) after 3, 6 and 8 steps. In the rightmost part, several trees have died because of too many shrinking radii due to competition, and other trees begin to invade the resulting gaps with their crowns. Adapted from KURTH (2002)

ved because of deadly suppression by its competitors. In the grammar, this mechanism is realized by an extra object called `Counter` at the basis of each tree, counting the shrinking crown radii (i.e., with crown radius status *growing* = 0) in a parameter `n`, and by a "cut operator" ("`cut`") which switches off the turtle interpretation of subsequent objects (technically, by isolating them from the rest of the graph). The counter at the basis of the tree can be accessed in the graph representation from each sector using the relation "`<-ancestor-`", which

Table 3. The Relational Growth Grammar `radii`

```
/* radii.rgg: specification of a competition model based on
   relations between representative crown radii of each tree */

module Tree;
module Sector(int growing, super.length, super.color)
   extends F(length, 0.1, color);
module Marker extends Null;
module Counter(int n);

double dp = 12;     /* distance between the planting positions */
double ds = 3;      /* threshold distance for competition */
double ang = 22.5;  /* opening angle of sensitive cone */

protected void init()
   [
   Axiom ==> for ((1:12))
      (
         [
         for ((1:15))  /* initial planting in rectangular array */
            ( [ Tree ] RU(90) M(dp) RU(-90) )
         ] RL(-90) M(dp) RL(90)
      );
   ]

public void make()
   [
   Tree ==> Counter(0) RU(90) RL(random(0, 360)) for (int i:(1:8))
              ( [ RL(i*45) Sector(1, 1, 2) Marker ] );
  (* x:Counter <-ancestor- *) s:Sector(g, r, c) (* ms:Marker *) ==>
      if (empty( (* t:Sector mt:Marker, ( t != s &&
          distance(ms, mt) <= ds && mt in cone(s, true, ang) ) *) ))
                        ( Sector(1, r+1, 2)   )
      else ( { x[n]++; } Sector(0, r-0.3, 4) );
   Counter(n), (n >= 5) ==> cut;
   ]
```

back traces the edges constituting the backbone of the tree representation.

Table 3 shows the complete grammar, and Figure 6 shows its application to a rectangular, regularly-spaced stand of 180 trees. Each tree is visibly represented by its 8 crown radii. The shrinking radii are marked with red colour.

Note that the only stochastic component in this model is the initial orientation of the "star" of 8 crown radii representing a tree. This is sufficient to generate random gaps in the aged stand.

5. Third Example: A Model of Plant-Herbivore Interaction

5.1. The Plant Submodel

The following example was inspired by a model constructed by Breckling (1990) (who did not use grammars), and is explained in further detail by Kurth (2000a, 2002). Going further beyond the previous examples, we now include the natural reproduction of plants by spreading of seed. A plant is represented by the module `Plant` and has two parameters, age *t* and size *r*. Size is assumed to be proportional to carbon content or energy content of the plant. Geometrically, a plant of size *r* is represented by a flat cylinder with radius *r*, giving the image of a circle when viewed from above. We use some heuristic rules for mortality. Our model does not intend to represent a refined model of carbon metabolism. The first rule is applied when the plant has reached a given maximal age, *pmaxage*:

```
Plant(t, r), (t > pmaxage) ==>;
```

The right-hand side of this rule is empty, i.e., the plant disappears (mineralisation and nutrient cycle are not represented in this model). The second rule has also an empty r.h.s.:

```
Plant(t, r), (r < 0) ==>;
```

This means that the plant dies because of negative carbon budget. The third rule represents the effect of competition. It searches for plants `q` in the neighbourhood which are larger than the plant under consideration and which cover its midpoint:

```
p:Plant, (* q:Plant *), (distance(p, q)
< q[radius]
   && p[radius] <= q[radius]) ==>;
```

We assume in this case, similar to the example `irreg` above, that the smaller plant is outcompeted. This model of competition for light is, of course, much simpler than many approaches which are described in the literature (e.g., Pfreundt, Sloboda, 1996). See Kurth, Sloboda (1999b), Hemmerling *et al.* (2008) for grammar representations of more detailed light competition models.

If none of the rules for mortality is applicable, the plant grows, and its age is increased by one:

```
Plant(t, r) ==> Plant(t+1, r+pgrow);
```

The amount of growth, `pgrow`, is a constant. Finally, there is a rule for reproduction. It is activated if one of two fixed (arbitrary) age stages is reached and if in the same time *r* is above a given threshold. This condition uses, besides `&&` (and), the logical operator `||` (or).

```
Plant(t, r), ((t == pgenage1 ||
t == pgenage2) &&
   r >= pminrad) ==>
      for ((1 : (int) (pgenfac*r)))
         ( [ RH(random(0, 360)) RU(90)
           M(random(distmin, distmax)) RU(-90)
           Plant(0, seed_rad) ] )
      Plant(t+1, r);
```

In the header of the loop, marked by the keyword "`for`", it is specified how many seeds are dispersed. Their number is proportional to the radius `r` of the plant. Spreading of seeds (which are simply represented by `Plant(0, seed_rad)`) is done in a similar

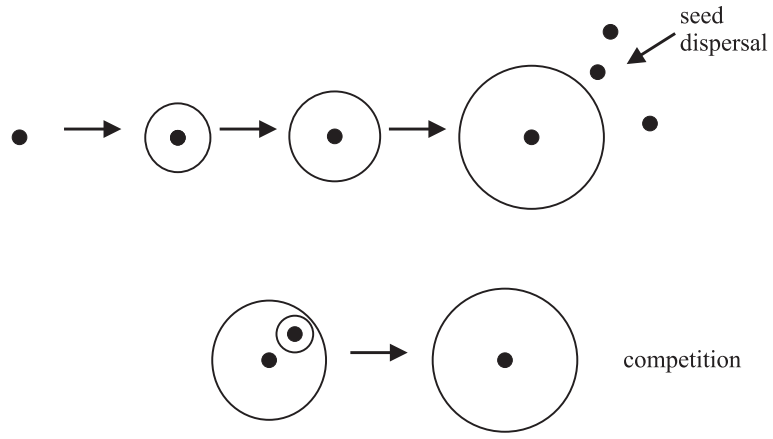

Fig. 7. Summary of the behaviour of plants in the "phytophag" grammar (see text)

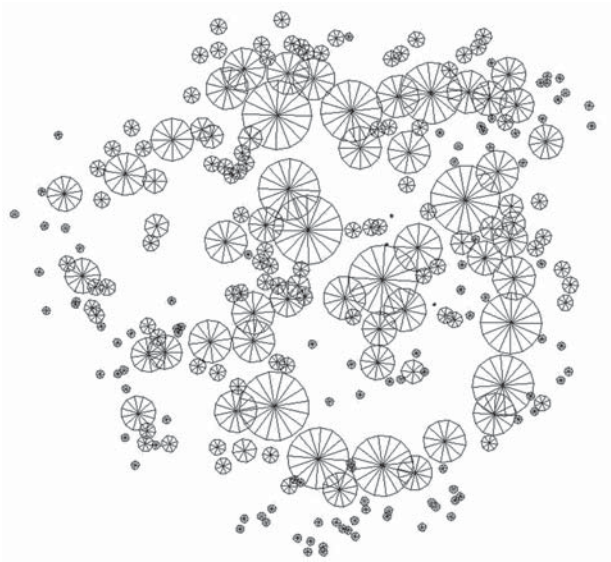

Fig. 8. Stand without herbivores after 80 steps of development, resulting from the plant grammar described in the text. Interaction between plants happens by the "competition" rule

way than in the example "`irreg2`" above. The distance from the mother tree is selected randomly between the limits `distmin` and `distmax`. The mother plant ages during this step, but does not grow (`Plant(t+1, r)`). In order to have higher priority than the rule for ordinary growth, the reproduction rule has to be inserted before that rule. In Figure 7, the key rules, that for growth and reproduction (upper part of picture) and that for competition (lower part), are visually summarized.

Already with this simple grammar, consisting of 5 rules, a richness of spatial patterns emerges. We obtain clusters of smaller plants and larger gaps which are later again invaded by plants. Figure 8 shows, as an example, a stand which has evolved since 80 time steps from a single plant which was located near the centre of the picture.

5.2. The Herbivore Submodel

We will now add further rules to represent herbivores which take their energy for living from the plants. Such a herbivore is symbolized in the grammar by `Animal (t, e)`, where *t* is age and *e* the size. *e* is assumed to be proportional to the reserve of carbon or energy. The herbivores are represented graphically by small circles with a colour different from that of the plants. To have a simple reproduction rule, we assume that the herbivores reproduce by division. In the real world, we can think of numerous microorganisms (e.g. bacteria, fungi) behaving this way. This time, there is only one mortality rule:

```
Animal(t, e), (e <= 0) ==>;
```

(again, the right side is empty; organic matter from the dead herbivore is not fed back into the system.) In contrast to the plants, the herbivores are mobile; they perform a random walk which is influenced only by the presence of plants. If a herbivore is not in contact with a plant, it is in a "search" status and makes long steps (distance "`longstep`"), causing a loss of energy ("`respi`"):

```
Animal(t, e) ==> RH(random(0, 360)) RU(90)
  M(longstep) RU(-90) Animal(t+1,
  e - f_e*respi);
```

Here, `f_e` is a fixed proportionality factor between the energy of a herbivore and its radius. Preceding this rule, we specify a rule which is applied when the herbivore has come into contact with a plant. This condition is checked using a graph query looking for a plant `p` with a distance smaller than the radius of `p`:

```
a:Animal(t, e), (* p:Plant(u, r) *),
(distance(a, p) < p[radius]) ==>
  RH(random(0, 360)) RU(90)
  M(shortstep) RU(-90)
  Animal(t+1, e + f_e*eat - f_e*respi)
  { p[radius] :-= eat; };
```

Here, the step of movement (`shortstep`) is shorter than in the case of search for food. The energy budget of the herbivore is diminished by "`respi`" and increased by an amount "`eat`" which is taken from the plant. The reduction of the energy (and, at the same time, of the radius) of the plant is modelled using an imperative statement, enclosed in braces `{ ... }`. The colon-led assignment operator "`:-=`" ensures a delayed execution, i.e., the update of the energy budgets of the plants is executed after all possible herbivore-plant interactions are checked. (Otherwise, the outcome of a simulation could depend on the order in which the herbivores are visited during rule application.) We could interpret this "grazing" interaction as a sort of communication between two objects. Reproduction takes place when a herbivore is large enough, i.e., has more energy than a threshold `thr`:

```
Animal(t, e), (e > f_e*thr) ==>
  [ RH(random(0, 360)) RU(90)
  M(shortstep) RU(-90)
  Animal(0, e/2 - f_e*respi) ]
  RH(random(0, 360)) RU(90)
  M(shortstep) RU(-90)
  Animal(0, e/2 - f_e*respi);
```

Both offspring move away in random directions and get *e*/2, half of the energy content of the parent, diminished by respiration. Figure 9 summarizes the herbivore rules.

Only a start rule, a rule for delaying the appearance of the herbivores in the beginning, and some declarations of parameters have to be added to the given rules to complete the relational growth grammar `phytophag`, documented in Table 4. In order to have a more transpa-

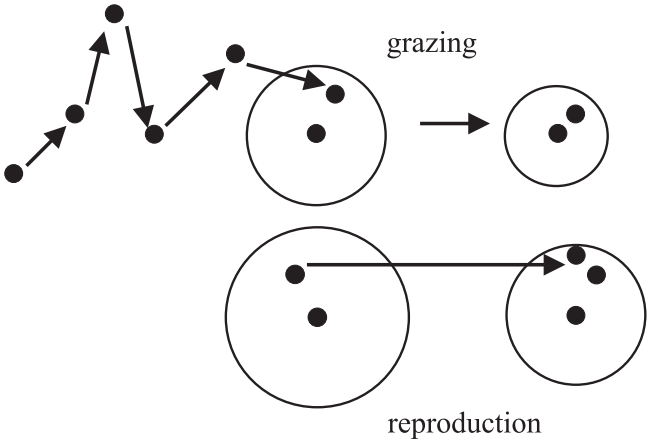

Fig. 9. Summary of the behaviour of herbivores in the "phytophag" grammar (see text)

rent code, the rules for plants and those for herbivores are separated in different blocks, "`growPlants`" and "`growAnimals`", using the possibility given in XL to organize the execution of rules with control structures and method calls as common in Java. Figure 10 shows two possible results of simulation runs after 125 steps, obtained with different parameterizations. Both simulation runs started with one plant and one herbivore. We see that complex spatial patterns can emerge. The dynamics in time does also depend on the choice of parameters. Figure 11 shows two examples: In the simulation run depicted on the left-hand side, the system collapses, i.e., the plants (and later, inevitably, also the herbivores) die out because of too much grazing. In the run depicted on the right-hand side, the plant population recovers after a while, and there is a long-term oscillation, as observed, e.g., in many real predator-prey systems.

6. Simulation of Evolving Strategies of Behaviour

The RGG rules offer the possibility to transmit genetic information concerning morphology and behaviour to the offspring. We demonstrate this at the example of the herbivores from the above model `phytophag`. We extend the `Animal` module by a third parameter `w` which represents the step width which is used for mo-

Fig. 10. Results of the grammar `phytophag` (described in the text) after 125 time steps. Both simulation runs started with one plant and one herbivore and differ in the parameterization. Wheel-like objects: plants, small points: herbivores

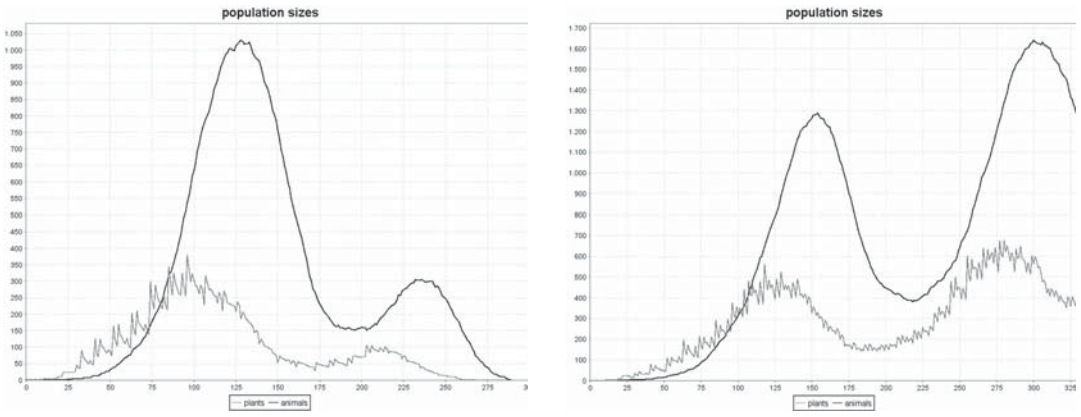

Fig. 11. Development of the numbers of individuals (smooth, dark line: herbivores, grey line: plants) in two simulation runs with the growth grammar `phytophag`, differing in their parameterization

Table 4. The Relational Growth Grammar `phytophag`

```
/* phytophag.rgg: specification of a grazing and competition model
   with circular-shaped plants and herbivores */

module Plant(int t, super.radius) extends Cylinder(1, radius)
   {{setColor(0x00aa00);}}
module Animal(int t, super.radius) extends Cylinder(2, radius)
   {{setColor(0xff0000); setBaseOpen(true); setTopOpen(true);}};

double pgrow = 0.9;   /* regular growth increment per timestep */
double seed_rad = 0.1;/* initial radius of a plant */
int pmaxage = 30;     /* maximal age of a plant */
int pgenage1 = 10;    /* first reproductive age level */
int pgenage2 = 18;    /* second reproductive age level */
double distmin = 15;  /* minimal seed distance */
double distmax = 40;  /* maximal seed distance */
double pminrad = 9;   /* necessary plant radius for reproduction */
double pgenfac = 0.5; /* ratio #seeds/radius */

int lag = 15;          /* sleeping time for herbivore at start */
double shortstep = 0.4;/* movement of herbivores inside plant canopy */
double longstep = 15;  /* movement of herbivores outside */
double f_e = 0.2;      /* ratio radius / energy of herbivores */
double init_e = 4;     /* initial energy amount of herbivores */
double respi = 0.25;   /* energy cosumed by herbivores' respiration */
double thr = 7.6;      /* energy threshold for reproduction of herbivores */
double eat = 1.1;      /* energy transferred during grazing */

protected void init()
   [
   Axiom ==> Plant(0, seed_rad) [ RH(random(0, 360))
             RU(90) M(10) RU(-90) Animal(-lag, f_e*init_e) ];
   ]

public void make()
   { growAnimals(); derive(); growPlants(); }

public void growAnimals()
   [
   Animal(t, e), (t < 0) ==> Animal(t+1, e); /* start lag */
   Animal(t, e), (e <= 0) ==> ;
   Animal(t, e), (e > f_e*thr) ==>
     [ RH(random(0, 360)) RU(90) M(shortstep) RU(-90)
       Animal(0, e/2 - f_e*respi) ]
       RH(random(0, 360)) RU(90) M(shortstep) RU(-90)
       Animal(0, e/2 - f_e*respi);
   a:Animal(t, e), (* p:Plant(u, r) *),
  (distance(a, p) < p[radius]) ==>
       RH(random(0, 360)) RU(90) M(shortstep) RU(-90)
       Animal(t+1, e + f_e*eat - f_e*respi) { p[radius] :-= eat; };

   Animal(t, e) ==>
       RH(random(0, 360)) RU(90) M(longstep) RU(-90)
       Animal(t+1, e - f_e*respi);
   ]

public void growPlants()
   [
   Plant(t, r), (t > pmaxage) ==> ;
   Plant(t, r), (r < 0) ==> ;
   p:Plant, (* q:Plant *), (distance(p, q) < q[radius]
      && p[radius] <= q[radius]) ==> ;
   Plant(t, r), ((t == pgenage1 || t == pgenage2) && r >= pminrad)
      ==> for ((1 : (int) (pgenfac*r)))
            ( [ RH(random(0, 360)) RU(90) M(random(distmin, distmax))
                RU(-90) Plant(0, seed_rad) ] )
          Plant(t+1, r);
   Plant(t, r) ==> Plant(t+1, r+pgrow);
   ]
```

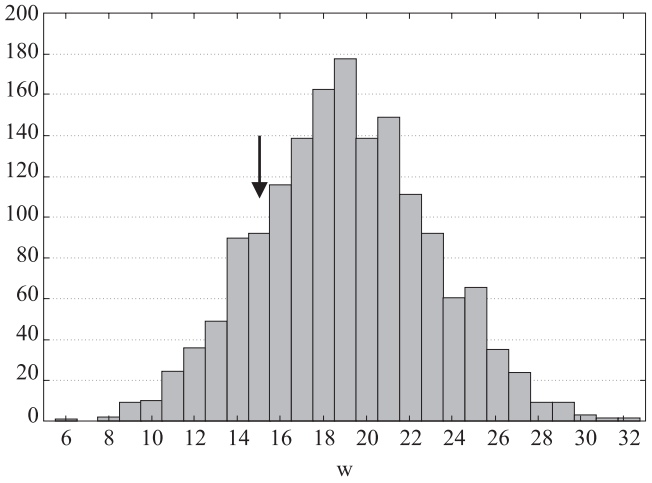

Fig. 12. Histogramme of the step lengths (mutated parameter w) in the herbivore population after 200 time steps, resulting from a simulation run of the grammar phytophag, extended by a random mutation of w in the reproduction rule (see text). The arrow marks the start value 15. The trend towards values larger than 15 is highly significant (*t*-test)

vements in the "search" mode (i.e., without contact to a plant). In the corresponding rule for movement, the constant `longstep` is thus replaced by this individual parameter `w`. We can interpret this parameter as a simple description of a strategy for foraging. In the rule for the reproduction of the herbivores, the value `w` is inherited by the offspring, but with a small random deviation (mutation), which we chose between –2 and 2 length units. In order to inhibit negative values, the resulting new value is compared with 0, and the maximum of both numbers is taken (using the standard Java function `Math.max`). The expression for the new value of parameter `w` for all offspring produced in the reproduction rule for herbivores is thus

```
Math.max(0, w + random(-2, 2))
```

In several simulation runs, this modification has led to a larger number of herbivores after 200 steps, compared with the model without mutation of step width. Furthermore, after this number of steps a significant shift of the value of parameter `w` from the initial value of 15 towards larger values can be observed (Figure 12). These are only preliminary results, but they demonstrate that it is relatively straightforward to simulate mechanisms of evolution by RGGs. In this connection it is worth to notice that in this example no fitness function was explicitly given. The fitness (chance for reproduction) of an individual results from the interaction with the simulated plants. In an extension of the model, a coevolution of plants and herbivores would be possible: To this purpose, the plants, too, would have to be equipped with genetic information and exposed to mutation.

7. Discussion

Further studies will be necessary to systematically explore the parameter spaces, i.e., the sets of all possible combinations of adjustable parameters of the presented models. This was not the aim here. Instead, it was intended to demonstrate the descriptive power of Relational Growth Grammars for the specification of various types of stand models – from simple descriptions of spatial patterns to population dynamics. Several advantages of the approach are obvious:

All results were obtained with one and the same software tool (GroIMP, see www.grogra.de, Kniemeyer, 2008) which had not to be recompiled for the different grammars. The grammars, which specify the essential features of the models including parameterization, are easy to manipulate.

To a certain degree, the rules are intuitively comprehensible and describe directly the behaviour of plants and herbivores (growth, reproduction, seed dispersal).

Not only the global behaviour of the simulated stand, but also "local histories" of certain trees or regions in the model plane can be investigated. Thus, comparisons with intensively-monitored research plots in real forests are possible. Furthermore, individual treatment of highly-valued trees can be simulated.

The appearance of singularities (catastrophes), e.g., breakdown of a stand from herbivore attack, can be studied in detail (cf. Breckling, 1990).

The universality of the approach is made plausible by the successful reimplementation of plant models from the literature which were originally not specified in terms of grammars and which can now all be studied using the same formal framework and software shell (cf. Kurth, 2000b).

Our example models belong to the class of individual-based models (IBMs). An overview about IBMs is given by Thiele *et al.* (2011), and their usage in ecology is extensively covered by deAngelis, Gross (1992), Grimm, Railsback (2005), and Railsback, Grimm (2011). In recent years, several generic software systems for IBMs have been developed and used, e.g., Repast (North *et al.,* 2007), MASON (Luke *et al.,* 2005), and NetLogo (Wilensky, Rand, in press). These systems and their corresponding model specification languages are adapted to handle the processes and behavioral properties, but they are not particularly well-suited for structural (architectural) properties of individuals. The latter are, however, especially important for trees and their interaction (Bormann, Likens, 1979). GroIMP and the language XL allow to combine individual-based modelling at the population scale with functional-structural modelling of the single tree (Hemmerling *et al.,* 2008). Furthermore, XL is distinguished by enabling the inclusion of L-systems, which have been widely used for structural plant models (cf. Prusinkiewicz *et al*., 1997).

The models which we have shown here have already been realised in L-system form in previous work (see the conference proceedings papers KURTH, SLOBODA, 2001 and KURTH, 2002), using the software Grogra (Growth grammar interpreter; KURTH, 1994). However, Grogra permits no general formalism of graph queries such as those possible in XL. Therefore, "sensitive functions" had to be used which were implemented in the Grogra source code and could not be modified in the grammars. Thus, the realisation in XL makes the models much more transparent and flexible. Furthermore, the fact that XL is an extension of Java makes it much easier to include procedurally-specified code, e.g., for process-based submodels, in a grammar-based structural model (c.f. HEMMERLING *et al.,* 2008).

Some extensions of the plant-herbivore model can easily be imagined:

– the inclusion of several trophic levels (predators),
– more refined rules for foraging and reproduction of herbivores,
– more detailed growth and competition models for the trees,
– a more detailed and realistic model of plant architecture, which can easily be realized with GroIMP – cf. the mixed-stand model presented in HEMMERLING *et al.* (2008), where coniferous and decidious trees compete for light,
– transmission of more complex genetic information in the reproduction rules,
– the inclusion of more realistic models of mutation, and also of recombination, which have already been realised in other RGG-based models of organisms (KNIEMEYER *et al.,* 2004; BUCK-SORLIN *et al.,* 2005).

But already in its current form, the implementation of the model in the language XL shows the appropriateness of this high-level language for a quite intuitive and transparent representation of mechanisms of competition, communication and foraging in ecological systems, particularly in forests.

Acknowledgements

The authors wish to thank Gerhard Buck-Sorlin, Helge Dzierzon, Broder Breckling, Marek Fabrika, Hans Pretzsch and Martin Schön for fruitful discussions and inspiration, and Katarína Smoleňová also for translation. Funding was obtained from BMBF and DFG. All support is gratefully acknowledged.

References

ABELSON H., DISESSA A.A., 1982: Turtle Geometry. Cambridge, MIT Press: 512 p.

ACOCK B., REYNOLDS J.F., 1997: Introduction: Modularity in plant models. Ecological Modelling, *94*: 1-6.

BORMANN F.H., LIKENS G.E., 1979: Pattern and Process in Forested Ecosystems. New York, Springer: 253 p.

BRECKLING B., 1990: Singularität und Reproduzierbarkeit in der Modellierung ökologischer Systeme. Dissertation, University of Bremen, FB2: 401 p.

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

BUCK-SORLIN G.H., KNIEMEYER O., KURTH W., 2005: Barley morphology, genetics and hormonal regulation of internode elongation modelled by a Relational Growth Grammar. New Phytologist, *166*(3): 859-867.

DEANGELIS D.L., GROSS L.J. (eds.), 1992: Individual-based Models and Approaches in Ecology: Populations, Communities and Ecosystems. New York, Chapman & Hall: 525 p.

DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P., 1998: Realistic modeling and rendering of plant ecosystems. SIGGRAPH 98 (Orlando, Fl., July 19–24, 1998), Computer Graphics Proceedings, Annual Conference Series, *1998*: 275-286.

ERMENTROUT G.B., EDELSTEIN-KESHET L., 1993. Cellular automata approaches to biological modelling. Journal of Theoretical Biology, *160*: 97-133.

GOSLING J., JOY B., STEELE G., BRACHNA G., 2005: The Java Language Specification. Third Edition. Reading, Addison-Wesley: 649 p. http://java.sun.com/docs/books/jls/ (last access: January 19, 2012).

GRIMM V., RAILSBACK S.F., 2005: Individual-based Modeling and Ecology. Princeton, N.J., Princeton University Press: 428 p.

HEMMERLING R., KNIEMEYER O., LANWERT D., KURTH W., BUCK-SORLIN G., 2008: The rule-based language XL and the modelling environment GroIMP illustrated with simulated tree competition. Functional Plant Biology, *35*: 739-750.

KNAUFT F.J., 2000: Entwicklung von Methoden zur GIS-gestützten Visualisierung von Waldentwicklungsszenarien. Ph.D. thesis, University of Göttingen, Faculty of Forest Sciences and Forest Ecology: 127 p., http://webdoc.sub.gwdg.de/diss/2000/knauft/inhalt.htm.

KNIEMEYER O., 2008: Design and Implementation of a Graph Grammar Based Language for Functional-structural Plant Modelling. Ph.D. thesis, University of Cottbus: 432 p., http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de: kobv:co1-opus-5937.

KNIEMEYER O., BUCK-SORLIN G.H., KURTH W., 2004: A graph grammar approach to Artificial Life. Artificial Life, *10* (4): 413-431.

KNIEMEYER O., BUCK-SORLIN G.H., KURTH W., 2007: GroIMP as a platform for functional-structural modelling of plants. In: VOS J., MARCELIS L.F.M., DEVISSER P.H.B., STRUIK P.C., EVERS J.B. (eds.): Functional-Structural Plant Modelling in Crop Production. Dordrecht, Springer: 43-52.

KURTH W., 1994: Growth Grammar Interpreter GROGRA 2.4. Berichte des Forschungszentrums Waldökosysteme der Universität Göttingen, Ser. B, *38*: 192 p.

KURTH W., 1999: Die Simulation der Baumarchitektur mit Wachstumsgrammatiken. Berlin, Wissenschaftlicher Verlag Berlin: 327 p.

KURTH W., SLOBODA B., 1999a: Tree and stand architecture described by formal grammars. I. Non-sensitive trees. J. Forest Sci., *45*: 16-30.

KURTH W., SLOBODA B., 1999b: Tree and stand architecture described by formal grammars. II. Sensitive trees and competition. J. Forest Sci., *45*: 53-63.

KURTH W., 2000a: Spezifikation räumlicher Bestandes- und Populationsmodelle mit sensitiven Grammatiken. In: SABOROWSKI J., SLOBODA B. (eds.): DVFFA Sektion Forstliche Biometrie u. Informatik, 12. Tagung 1999. Ljubljana, Biotechn. Fak.: 259-278.

KURTH W., 2000b: Towards universality of growth grammars: Models of Bell, Pagès, and Takenaka revisited. Ann. For. Sci., *57*: 543-554.

KURTH W., 2002: Spezifikation der Simulation der Struktur und Dynamik von Pflanzenbeständen und Tierpopulationen mit sen-

sitiven Wachstumsgrammatiken. In: Wittmann J., Gnauck A. (ed.): Simulation in Umwelt- und Geowissenschaften. Workshop Cottbus, 7.-8. 3. 2002. Aachen, Shaker: 37-51.

Kurth W., 2007: Specification of morphological models with L-systems and Relational Growth Grammars. Image – Journal of Interdisciplinary Image Science, *5* / Special Issue.

Kurth W., Kniemeyer O., Buck-Sorlin G.: Relational Growth Grammars – a graph rewriting approach to dynamical systems with a dynamical structure. In: Banâtre J.-P., Fradet P., Giavitto J.-L., Michel O. (eds.): Unconventional Programming Paradigms. Lecture Notes in Computer Science, *3566*, Berlin, Springer: 56-72.

Kurth W., Lanwert D., 2011: Grammar-based models and fractals. In: Jopp F., Reuter H., Breckling B. (eds.): Modelling Complex Ecological Dynamics. Berlin, Springer: 147-161.

Kurth W., Sloboda B., 2011: Sensitive growth grammars specifying models of forest structure, competition and plant-herbivore interaction. In: Rennolls K. (ed.): Proceedings of the IUFRO 4.11 Congress "Forest Biometry, Modelling and Information Science", Greenwich, UK (25.–29. 6. 2001), online publication, http://www.uni-forst.gwdg.de/~wkurth/cb/html/gree_tx.pdf (last access: May 6, 2012).

Lewandowski A., von Gadow K., 1997: Ein heuristischer Ansatz zur Reproduktion von Waldbeständen. Allg. Forst- u. Jagdzeitg., *168*: 170-174.

Lindenmayer A., 1968: Mathematical models for cellular interactions in development. Journal of Theoretical Biology, *18*: 280-299, 300-315.

Luke S., Cioffi-Revilla C., Panait L., Sullivan K., Balan G., 2005: MASON: A multi-agent simulation environment. Simulation, *82*: 517-527.

North M.J., Tatara E., Collier N.T., Ozik J., 2007: Visual agent-based model development with Repast Simphony. In: Agent 2007 Conference: Complex Interaction and Social Emergence, 15-17 Nov. 2007, Evanston, IL: 173-192.

Penttinen A., Stoyan D., Henttonen H.M., 1994: Marked point processes in forest statistics. For. Sci., *38*: 806-824.

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

Pretzsch H., 1992a: Kronenformen und ihre Veränderung unter Konkurrenz. In: Jahrestagung 1992 d. Sekt. Ertragskunde des Dt. Vb. Forstl. Forschungsanstalten, Grillenburg: 40-61.

Pretzsch H., 1992b. Modellierung der Kronenkonkurrenz von Fichte und Buche in Rein- und Mischbeständen. Allg. Forst- u. Jagdzeitg., *163*: 203–213.

Pretzsch H., 2001: Modellierung des Waldwachstums. Berlin, Parey: 341 p.

Pretzsch H., 2009: Forest Dynamics, Growth and Yield. Berlin, Springer: 664 p.

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

Prusinkiewicz P., Hammel M., Hanan J., Měch R., 1997: Visual models of plant development. In: Rozenberg G., Salomaa A. (eds.): Handbook of Formal Languages. Vol. 3: Beyond Words. Berlin, Springer: 535-597.

Railsback S.F., Grimm V., 2011: Agent-Based and Individual-Based Modeling: A Practical Introduction. Princeton, Princeton University Press: 352 p.

Sloboda B., 1983. Möglichkeiten der mathematischen Vorhersage der Holzproduktion im Wirtschaftswald. Forstarchiv, *54*: 134–142.

Thiele J.C., Kurth W., Grimm V., 2011: Agent- and individual-based modeling with NetLogo: Introduction and new NetLogo extensions. In: Römisch K., Nothdurft A., Wunn, U. (eds.): Die Grüne Reihe 22. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten und der Arbeitsgemeinschaft Ökologie und Umwelt der Internationalen Biometrischen Gesellschaft – Deutsche Region, 20-21 September 2010 in Göttingen (Germany): 68-101.

Wilensky U., Rand W., (in press): An Introduction to Agent-Based Modeling: Modeling Natural, Social and Engineered Complex Systems with NetLogo. Cambridge, MIT Press.