6. Introduction to rule-based simulation

Examples of processes which are studied by simulation on a computer:

- growth and crown development of a plant
- chemical reactions in a cell
- population dynamics of competing tree species
- foraging behaviour of ants
- water flow in the soil
- interception of photosynthetically-active radiation by a canopy
- dynamics of traffic on a road network
- economic decisions of traders on a market
- ...

Different formal systems, programming languages and software platforms are in use which support such simulations.

As an example, we demonstrate the usage of graph-grammar rules in the language XL to simulate the 3-dimensional development of plants.

XL = eXtended L-system language

L-systems (Lindenmayer systems): rules working on character strings, named after the botanist Aristid Lindenmayer (1925-1989)

L-systems (Lindenmayer systems)

rule systems for the replacement of character strings

in each derivation step *parallel* replacement of all characters for which there is one applicable rule

An L-system mathematically:

a triple (Σ, α, *R*) with:

Σ a set of characters, the *alphabet*,

α a string with characters from Σ, the *start word* (also "Axiom"),

R a set of rules of the form

character $\rightarrow$ string of characters;

with the characters taken from Σ.

A *derivation step* (rewriting) of a string consists of the replacement of all of its characters which occur in left-hand sides of rules by the corresponding right-hand sides.

Convention: characters for which no rule is applicable stay as they are.

Result:

Derivation chain of strings, developed from the start word by iterated rewriting.

Example:

alphabet {A, B}, start word A

set of rules:

A → B
B → AB

derivation chain:

A → B → AB → BAB → ABBAB → BABABBAB
→ ABBABBABABBAB → BABABBABABBABBABABBAB
→ ...

still missing for modelling biological structures in space:

a geometrical interpretation

Thus we add:

a function which assigns to each string a subset of 3-D space

„interpreted“ L-system processing

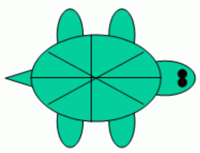
$$\alpha \to \sigma_1 \to \sigma_2 \to \sigma_3 \to ....$$
$$\downarrow \quad \downarrow \quad \downarrow$$
$$S_1 \quad S_2 \quad S_3 \quad ....$$

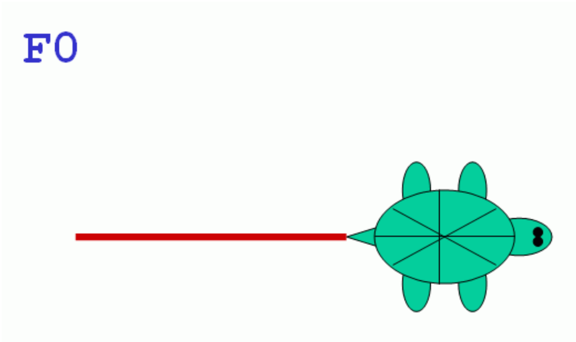
S_1, S_2, S_3, ... can be seen as developmental steps of an object, a scene or an organism.

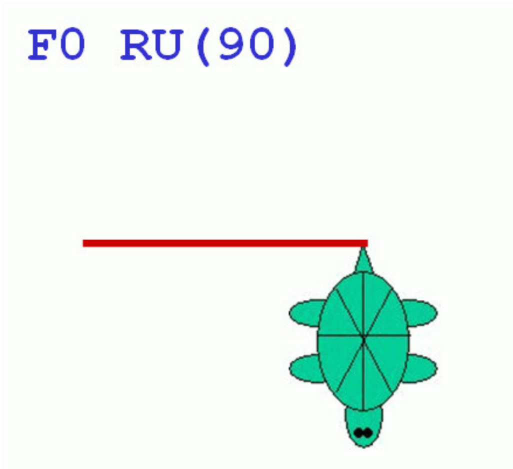
For the interpretation: *turtle geometry*

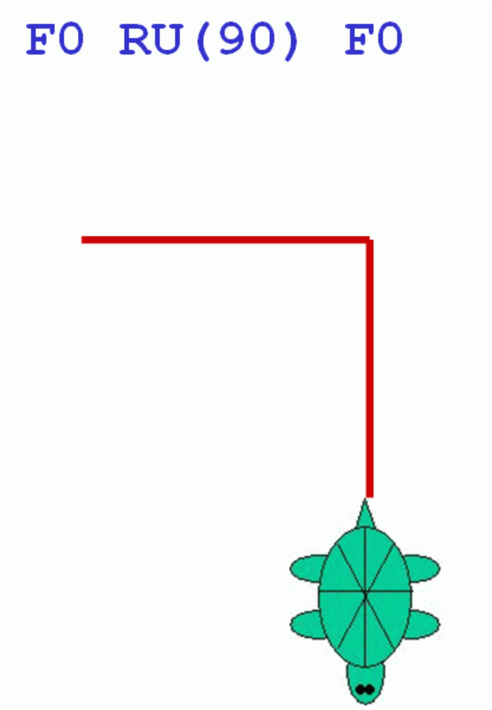
Turtle:

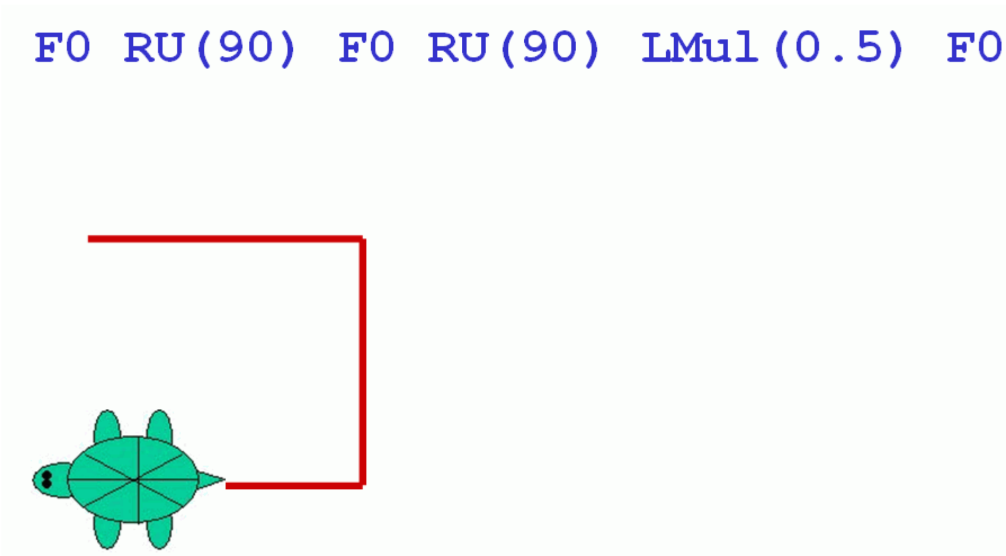
goes according to commands




F0


F0 RU(90)


F0 RU(90) F0


F0 RU(90) F0 RU(90) LMul(0.5) F0

„turtle": virtual device for drawing or construction in 2-D or 3-D space

- able to store information (graphical and non-graphical)
- equipped with a memory containing state information (important for branch construction)
- current turtle state contains e.g. current line thickness, step length, colour, further properties of the object which is constructed next

Turtle commands in XL (selection):

`F0` "Forward", with construction of an element (line segment, shoot, internode...), uses as length the current step size (the zero stands for „no explicit specification of length")

`M0` forward without construction (*Move*)

`L(x)` change current step size (length) to *x*

`LAdd(x)` increment the current step size to *x*

`LMul(x)` multiply the current step size by *x*

`D(x)`, `DAdd(x)`, `DMul(x)` analogously for current thickness

Repetition of substrings possible with "`for`"

e.g., `for ((1:3)) ( A B C )`

yields `A B C A B C A B C`

Exercise:

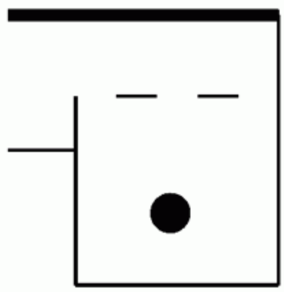
what is the result of the interpretation of

```
L(10) for ((1:6))
          ( F0 RU(90) LMul(0.8) )   ?
```

Example:

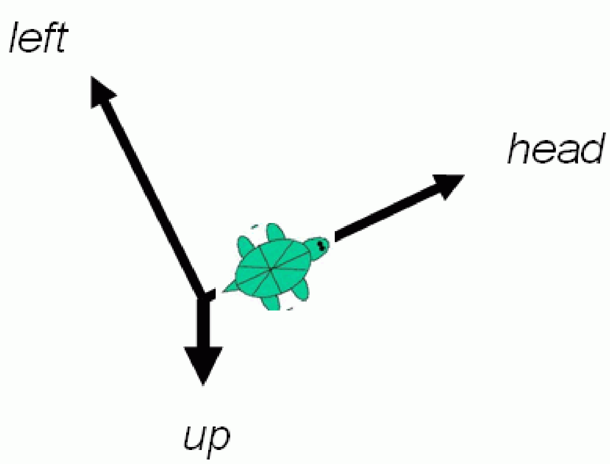
```
L(100) D(3) RU(-90) F(50) RU(90) M0 RU(90) D(10) F0 F0
       D(3) RU(90) F0 F0 RU(90) F(150) RU(90) F(140) RU(90)
       M(30) F(30) M(30) F(30) RU(120) M0 Sphere(15)
```

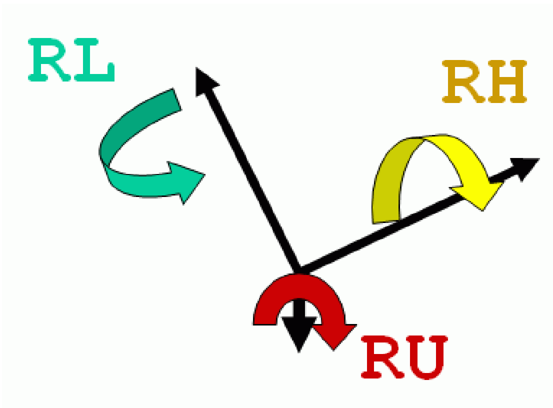
generates



Extension to 3-D graphics:

turtle rotations by 3 axes in space





3-D commands:

RU(45) rotation of the *turtle* around the "up" axis by 45°

RL(...) , **RH(...)** analogously by "left" and "head" axis

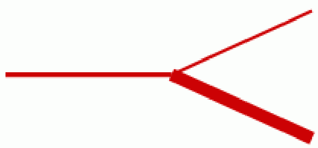
up-, *left*- and *head* axis form an orthogonal spatial coordinate system which is carried by the *turtle*

Branches:
realization with memory commands

[put current state on stack ("Ablage", Stack)

] take current state from stack and let it become the current state (thus: end of branch!)

```
F0 [ RU(-20) F0 ] RU(20) DMul(2) F0
```

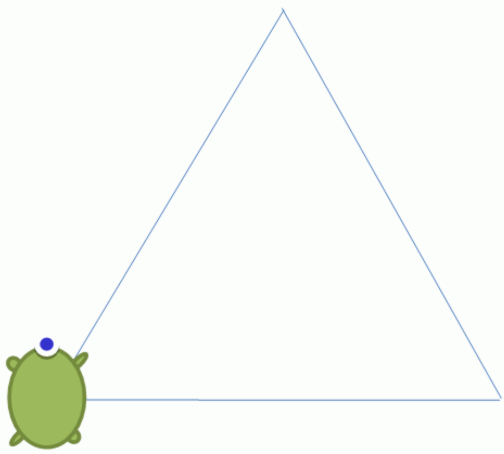


How to execute a turtle command sequence with GroIMP

write into a GroIMP project file (or into a file with filename extension `.rgg`):

```
protected void init()
    [
    Axiom ==> turtle command sequence ;
    ]
```

Example: Drawing a triangle



```
protected void init()
   [ Axiom ==> RU(30) F(10) RU(120) F(10) RU(120) F(10) ]
```

see file `sm09_e01.rgg`

now we make the turtle-generated patterns dynamic

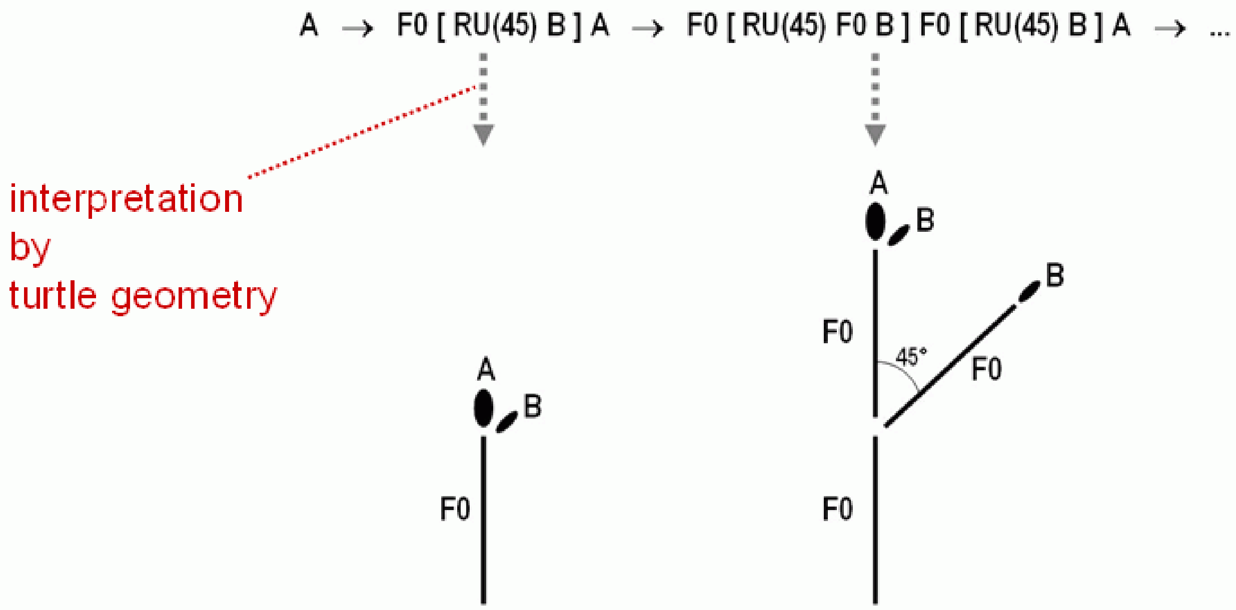
Interpreted L-system:
The alphabet of the L-system contains the turtle command language as a subset.

Example:

rules

```
A ==> F0 [ RU(45) B ] A ;
B ==> F0 B ;
```

start word **A**

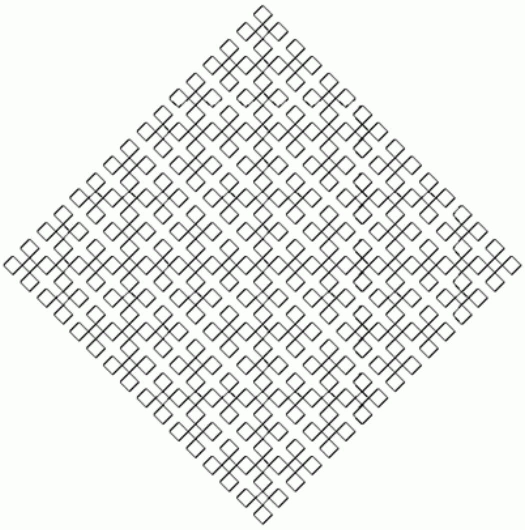


(**A** and **B** are normally not interpreted geometrically.)

also modelling of objects different from plants

example space filling curve:

```
Axiom ==> L(10) RU(-45) X RU(-45) F(1) RU(-45) X;

X ==> X F0 X RU(-45) F(1) RU(-45) X F0 X
```

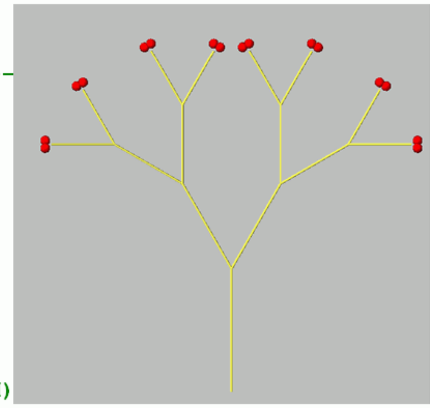


traditional Indian kolam
„Anklets of Krishna“

A simple plant with dichotomous branching:

sample file **sm09_e03.rgg** :

```
/* You learn at this example:
- how to construct a simple plant model (according to architectural model Schoute)
- how to specify branches with [ ]  */

// Example of a simple tree architecture (Schoute architecture)

//----------- Extensions to the standard alphabet ----------
//Shoot() is an extension of the turtle-command F() and stands for an annual shoot
module Shoot(float len) extends F(len);

// Bud is an extension of a sphere object and stands for a terminal bud
// its strength controls the length of the produced shoot in the next timestep
module Bud(float strength) extends Sphere(0.2)
{{ setShader(RED); setTransform(0, 0, 0.3); }};
//-------------------------------------------------------

protected void init ()
[  // start structure (a bud)
   Axiom ==> Bud(5);
]

public void run ()
[
   // a square bracket [] will indicate a branch
   // (daughter relation)
   // Rotation around upward axis (RU) and head axis (RH)
   // Decrease of strength of the Bud (each step by 20%)

   Bud(x) ==> Shoot(x) [ RU(30) Bud(0.8*x) ] [ RU(-30) Bud(0.8*x) ];
]
```



extension of the concept of symbol:

allow real-valued parameters not only for turtle commands like "`RU(45)`" and "`F(3)`", but for all characters

→ *parametric L-systems*

arbitrarily long, finite lists of parameters
parameters get values when the rule matches

Example:

rule `A(x, y) ==> F(7*x+10) B(y/2)`

current symbol is e.g.: `A(2, 6)`
after rule application: `F(24) B(3)`

parameters can be checked in conditions
(logical conditions with Java syntax):

`A(x, y) (x >= 17 && y != 0) ==>` ....

Stochastic L-systems

usage of pseudo-random numbers

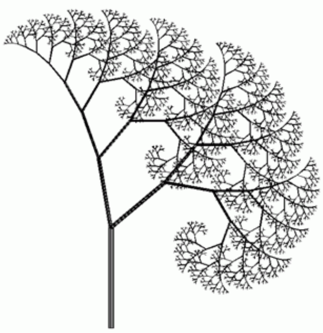
Example:

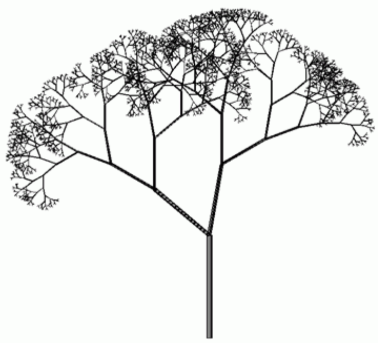
deterministic

```
Axiom ==> L(100) D(5) A;

A ==> F0 LMul(0.7) DMul(0.7)
  [ RU(50) A ] [ RU(-10) A ];
```

stochastic

```
Axiom ==> L(100) D(5) A;

A ==> F0 LMul(0.7) DMul(0.7)
  if (probability(0.5))
   ( [ RU(50) A ] [ RU(-10) A ] )
  else
   ( [ RU(-50) A ] [ RU(10) A ] );
```





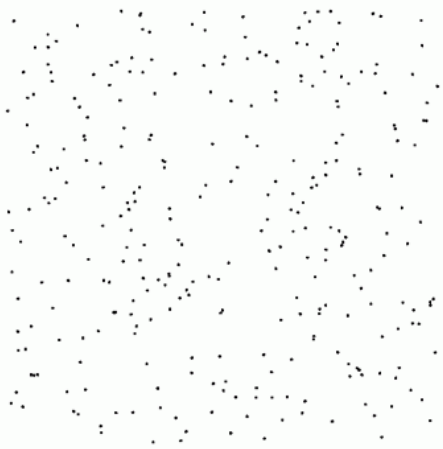
XL functions for pseudo-random numbers:

`Math.random()`	generates floating-point random number between 0 and 1
`random(a, b)`	generates floating point random number between **`a`** and **`b`**
`probability(x)`	gives 1 with probability x, 0 with probability 1–x

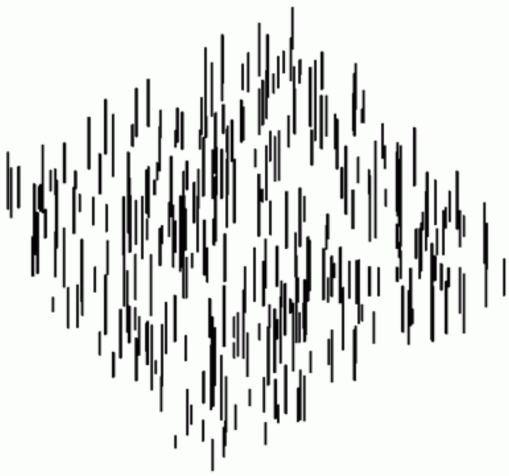
How to create a random distribution in the plane:

```
Axiom ==> D(0.5) for ((1:300))
            ( [ Translate(random(0, 100), random(0, 100), 0)
                F(random(5, 30)) ] );
```

view from above



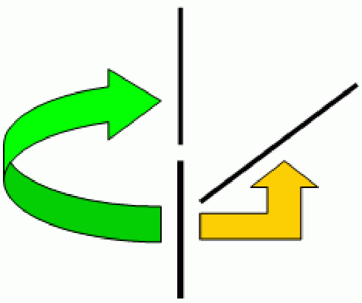
oblique view



The step towards graph grammars

drawback of L-systems:

• in L-systems with branches (by turtle commands) only 2 possible relations between objects: "direct successor" and "branch"

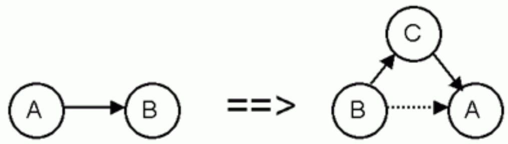


extensions:

• to permit additional types of relations
• to permit cycles

→ **graph grammar**

Example of a graph grammar rule:



- each left-hand side of a rule describes a subgraph (a pattern of nodes and edges, which is looked for in the whole graph), which is replaced when the rule is applied.

- each right-hand side of a rule defines a new subgraph which is inserted as substitute for the removed subgraph.

special variant of graph grammars:
Relational growth grammars (RGG)

- parallel application, same as for L-systems
- attributed vertices and edges
- vertex types with object hierarchy (a vertex type can inherit properties from another vertex type)

The language XL

specification: Kniemeyer (2008)

- extension of Java
- allows also specification of L-systems and RGGs (graph grammars) in an intuitive rule notation

imperative blocks, like in Java: { ... }

rule-oriented blocks (RGG blocks): [...]

During execution of an XL program, there is one graph (represented in the computer memory) which is transformed by the rules

- the nodes (vertices) of this graph are basically Java objects
  (they can also be geometrical objects)

Example:
rules for the fractal curve shown in Chapter 5

```
public void derivation()
  [
  Axiom ==> RU(90) F(10);
  F(x) ==> F(x/3) RU(-60) F(x/3) RU(120) F(x/3) RU(-60) F(x/3);
  ]
```

nodes of the graph

edges (type „successor“)

Queries in the graph

a query is enclosed by **(*   *)**

The elements are given in their expected order, e.g.:
(* A A B *) searches for a subgraph which consists of a sequence of nodes of the types **A A B**, connected by successor edges.

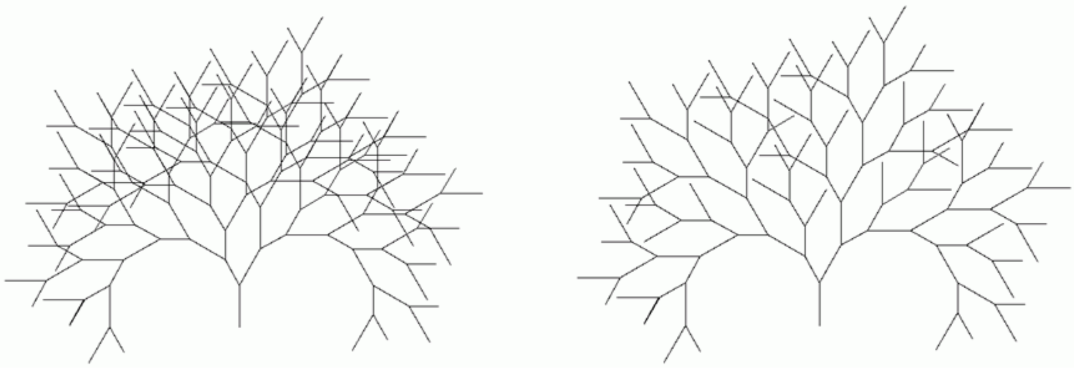
example for a graph query:

binary tree, growth shall start only if there is enough distance to other **F** objects

```
Axiom ==> F(100) [ RU(-30) A(70) ] RU(30) A(100);
a:A(s) ==> if ( forall(distance(a, (* F *)) > 60) )
           ( RH(180) F(s) [ RU(-30) A(70) ] RU(30) A(100) )
```

without the „if“ condition

with the „if“ condition



Example for modelling a "simple" plant:
a daisy

(following K. Smoleňová and R. Hemmerling)

Steps shown here:

1. Gather Data
2. Create Topology
3. Texturing
4. Parameter Calibration and Randomness

Results of data / knowledge collection about daisy (*Bellis perennis*):

- Small rounded or spoon-shaped evergreen leaves, 2-5 cm long, close to the ground, rosulate arrangement
- Leafless stem, 2-10 cm long
- Green bracts in two rows, usually 13
- Flower base, conical shape, 6 mm long, 5 mm in diameter
- White flowers, 11 mm long, 2 mm wide
- Yellow disc flowers

Definitions of the parts of the virtual plant (restricted to the above-ground part):

```
module Leaf;
module Stem;
module Bract;
module FlowerBase;
module Flower;
```

Definition of the corresponding parameters:

```
module Leaf(float length, float diameter);
module Stem(float length, float diameter);
module Bract(float lenth, float diameter);
module FlowerBase(float length, float diameter);
module Flower(float length, float diameter,
    int color);
```

How to assign a shape to a part?

Two possibilities in XL:

- by inheritance from a predefined geometrical object (using the keyword "**`extends`**")
- by instantiation with one or more simpler objects (using the arrow "**==>**" in the module declaration)

```
module Leaf(float length, float diameter)
    ==> leaf(length, diameter);
module Stem(float length, float diameter)
    extends Cylinder(length, diameter/2);
module Bract(float lenth, float diameter)
    ==> leaf(length, diameter);
module FlowerBase(float length, float diameter)
    ==> Cone(length, diameter/2);
```

```
module Flower(float length, float diameter,
    int color)
    ==> if (color == YELLOW)
            (Cylinder(length, diameter/2))
        else if (color == WHITE)
            (leaf(length, diameter));
```

Derivation of the leaves at the base of the plant:

```
Axiom ==>
    // create rosette of 7 leaves,
    // diameter is half of their length
    for (int i:1:7)
    ( [
        RH(i * 137.5)
        M(i-1)
        RU(leafAngle)
        RH(90)
        { double r = 50 - i * 5; }
        Leaf(r, r/2)
    ] )
    ...
;
```

Derivation of the stem:

```
    ...

    // create the stem, 70 mm long,
    // diameter 2 mm
    Stem(70, 2)
```

Derivation of the bracts:

```
...

// create 13 bracts,
// each 9 mm long, 2 mm in width
for (int i:1:13)
( [
    M(-1)
    RH(360 * i / 13)
    RU(bractAngle)
    RH(90)
    Bract(9, 2)
] )
```

Derivation of the base of the inflorescence:

```
...

// create flower base,
// 6 mm long, 5 mm diameter
FlowerBase(6, 5)
```

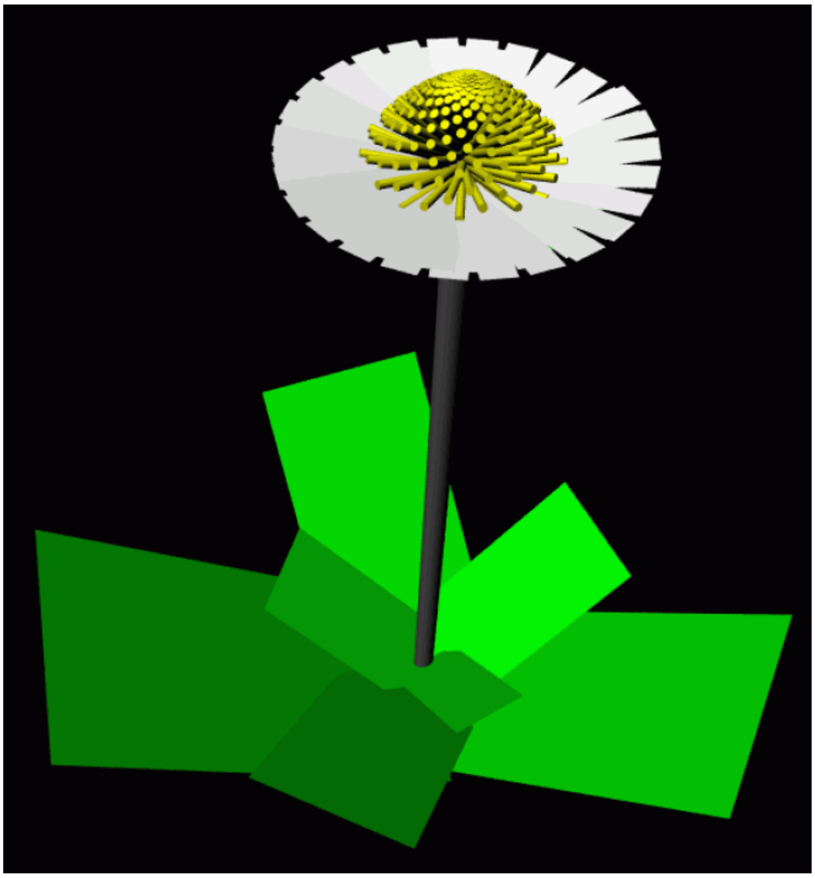
Derivation of the white flowers:

```
...

// create white flowers around flower base
for (int i:1:50)
( [
    M(-6 + i * 0.02)
    RH(i * 13.7)
    RU(whiteFlowerAngle)
    RH(90)
    Flower(11, 2, WHITE)
] )
```

Derivation of the yellow flowers:

```
...

// create yellow flowers around flower base
for (int i:1:250)
( [
    { float h = i * 0.02; }
    M(-h)
    RH(i * 137.5)
    Translate(h * k, 0, 0)
    RU(whiteFlowerAngle * i / 250)
    Flower(1.0 + 3.0 * (h / 5.0), 0.5, YELLOW)
] )
```

Result so far:



To obtain more visual realism, *textures* are needed.

Sources for surface textures of plants:

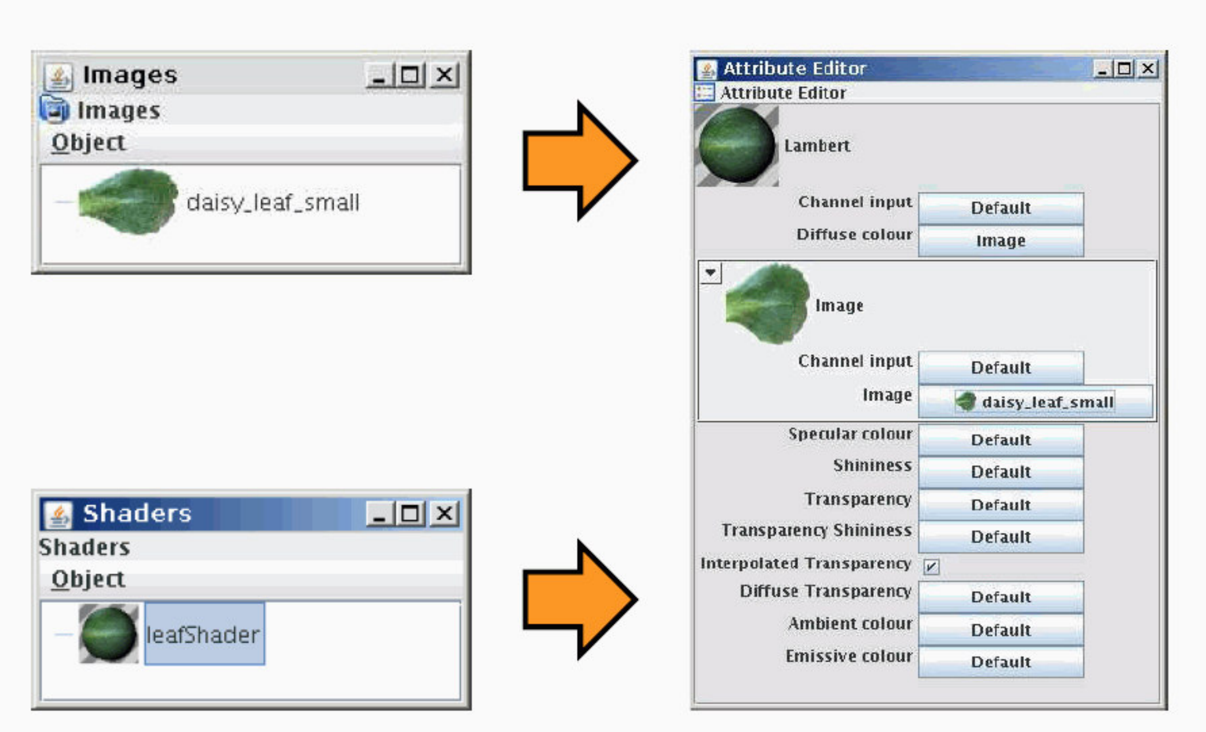
digital camera, scanner, existing images (from the web or from botanical books)

Preparation of the textures:

- adjust lighting
- cut out, make background transparent
- resize (avoid too memory-consuming textures)

examples of prepared daisy textures:

Import of textures into GroIMP: interactively



Application to an object (here: a leaf):

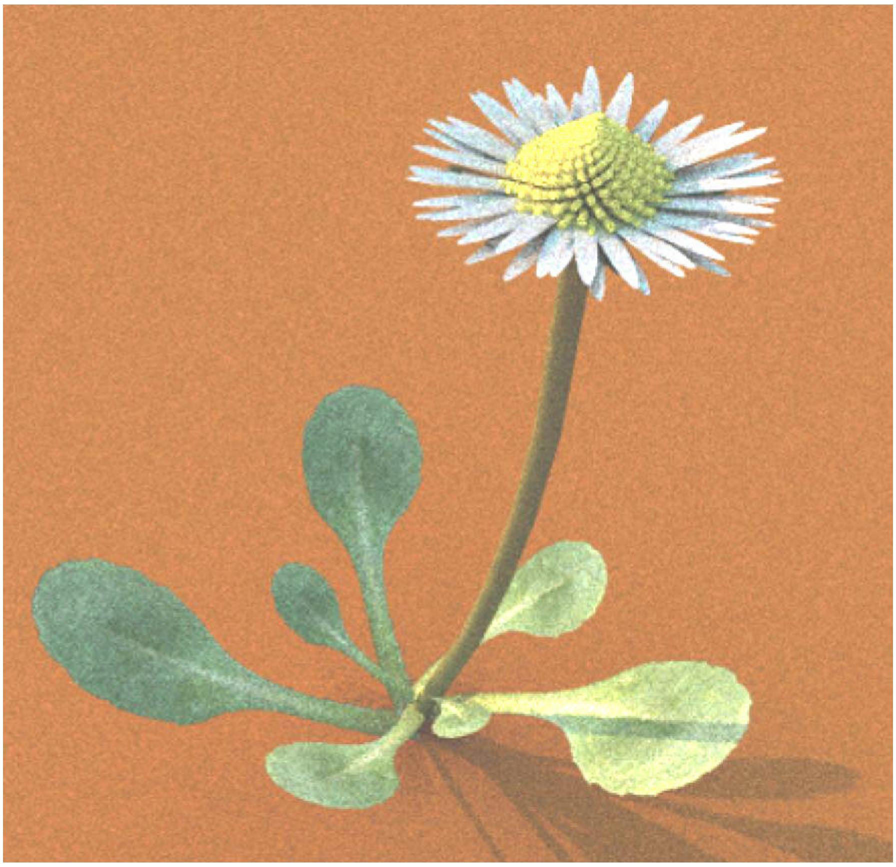
```
// obtain reference to named shader
ShaderRef leafShader = shader("leafShader");

// set shader during interpretation
module Leaf(float length, float diameter)
    ==> leaf(length, diameter).(
          setShader(leafShader));
```

Result of texturing the virtual plant:

Next steps:
Adjustment of parameters, introduction of variability

- Make plant look more natural by generating values (angle, length, diameter, ...) randomly
- Perform statistical analysis of the model followed by parameter adjustment until the model fits the observed data
- Perform statistical analysis of real plants to obtain mean and variance for stochastic generation of parameter values

Result with stochastic variations:



Deficiencies:
The daisy model is purely structural (has no processes like photosynthesis, respiration, uptake of water and nutrients...); there is no dynamics (growth, unfolding of organs, senescence...).

Next step would be:
Creation of a functional-structural plant model (FSPM) with rules describing ontogenesis.

A simple functional-structural plant model in XL:
see example file **`sfspm09.gsz`**

includes:

- light emitted from a lamp
- interception of light by the leaves of the plant
- a submodel for photosynthesis
- transport of assimilates along the plant axes
- formation of new internodes and leaves
- growth of the organs
- flowering

executable by GroIMP

The software GroIMP

GroIMP = "growth-grammar related interactive modelling platform"

see **`http://www.grogra.de`**,

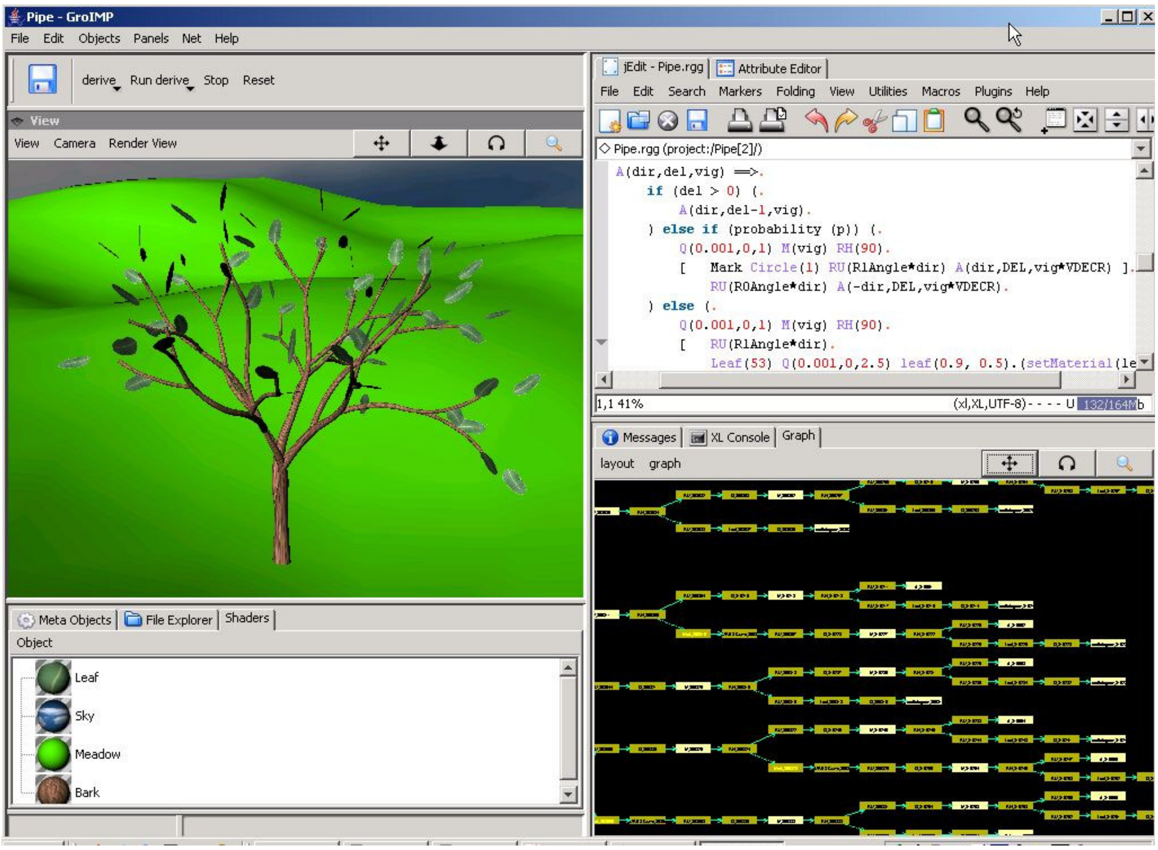
there you find also the link to the download site **`http://sourceforge.net/projects/groimp/`** and a gallery of examples.

See also the learning units about GroIMP (author: K. Petersen, M.Sc. Forest Science), available in StudIP.

GroIMP is an open source project. It combines:

- XL compiler and interpreter
- a development environment for XL
- an interactive 3-d modeller
- several 3-d renderers
- a 2-d graph visualization tool
- an editor for 3-d objects and attributes
- tools for texture generation
- an interface for measured tree architecture data
- a simulation tool for radiation in scenes
- support for solving differential equations in a numerically stable way (for submodels)
- interfaces for data formats like dxf, obj, mtg, pdb
- ...

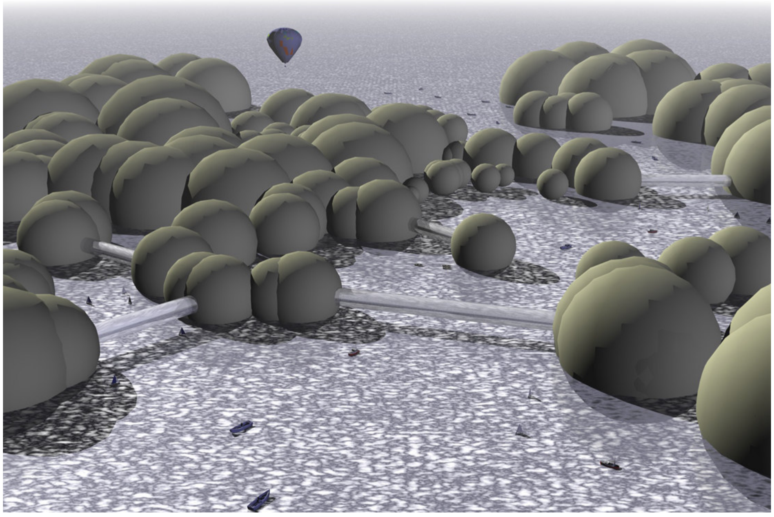
screenshot:



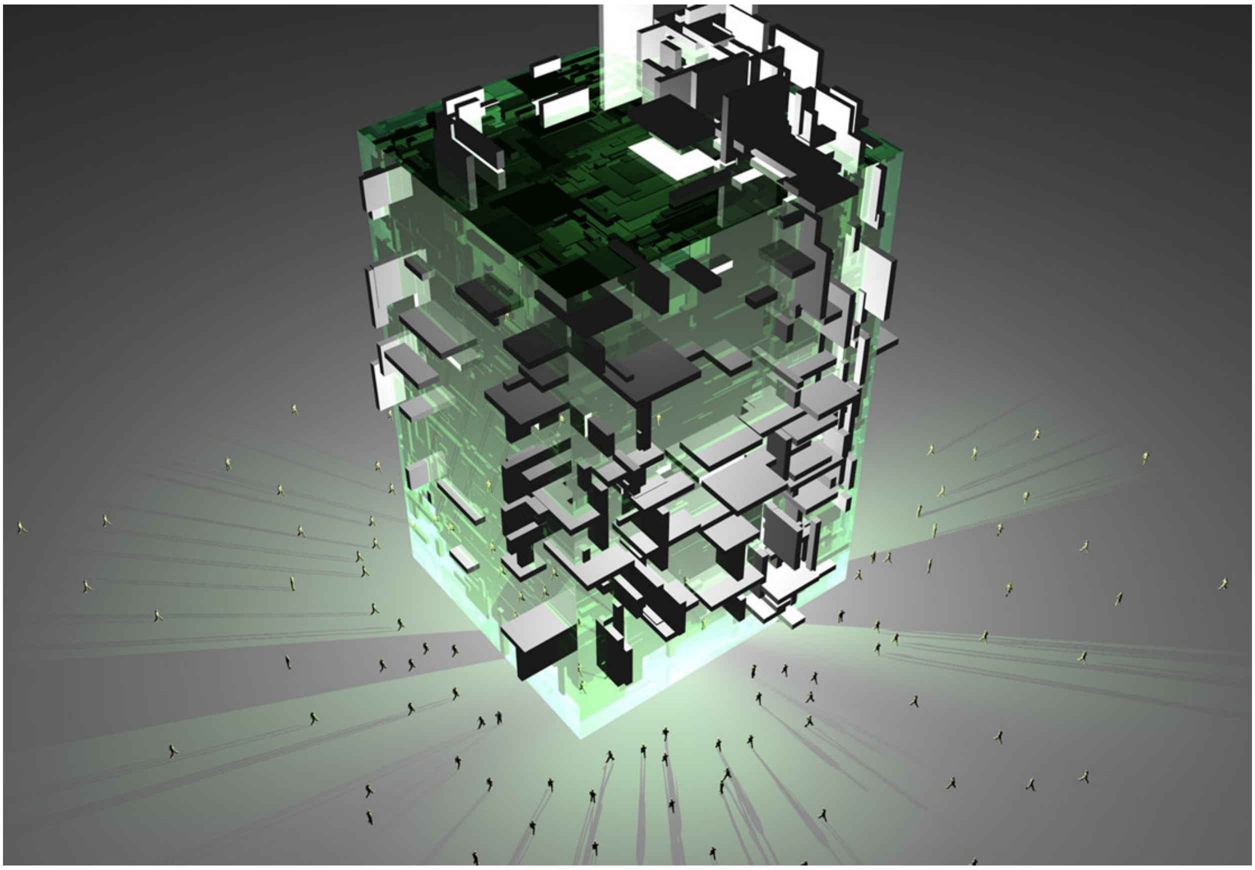
example applications:

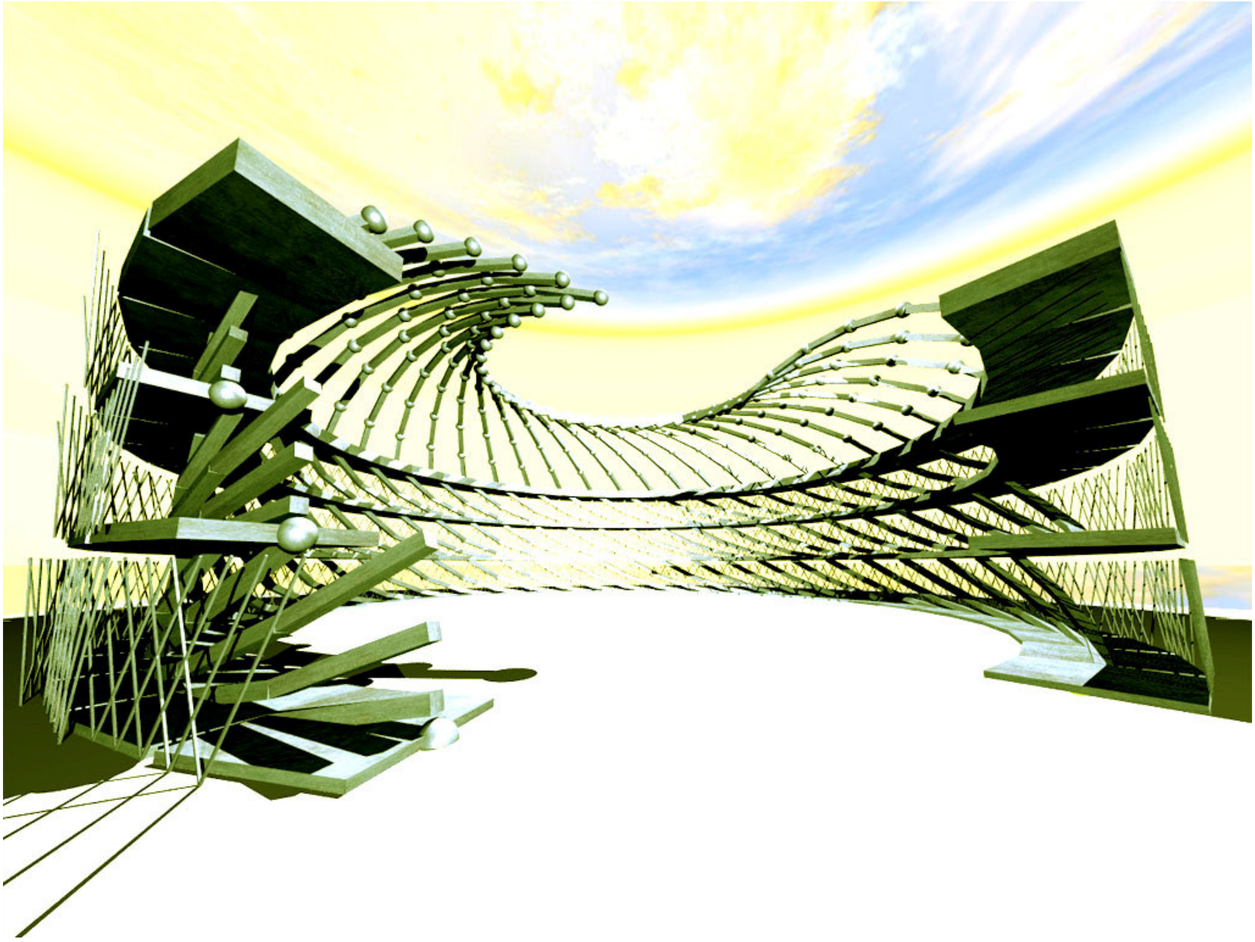
virtual barley
(Buck-Sorlin 2006)

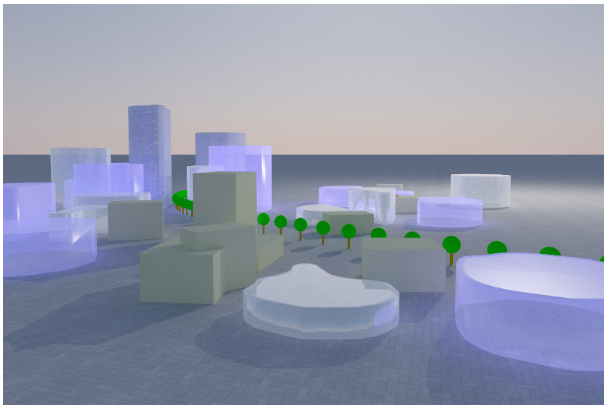
virtual Black Alder tree, generated with GroIMP, in a VRML scene (for Branitz Park Foundation, Cottbus; Rogge & Moschner 2007)



This and next images: students' results from architecture seminar, BTU Cottbus 2007







virtual landscape with beech-spruce mixed stand
(Hemmerling et al. 2008)