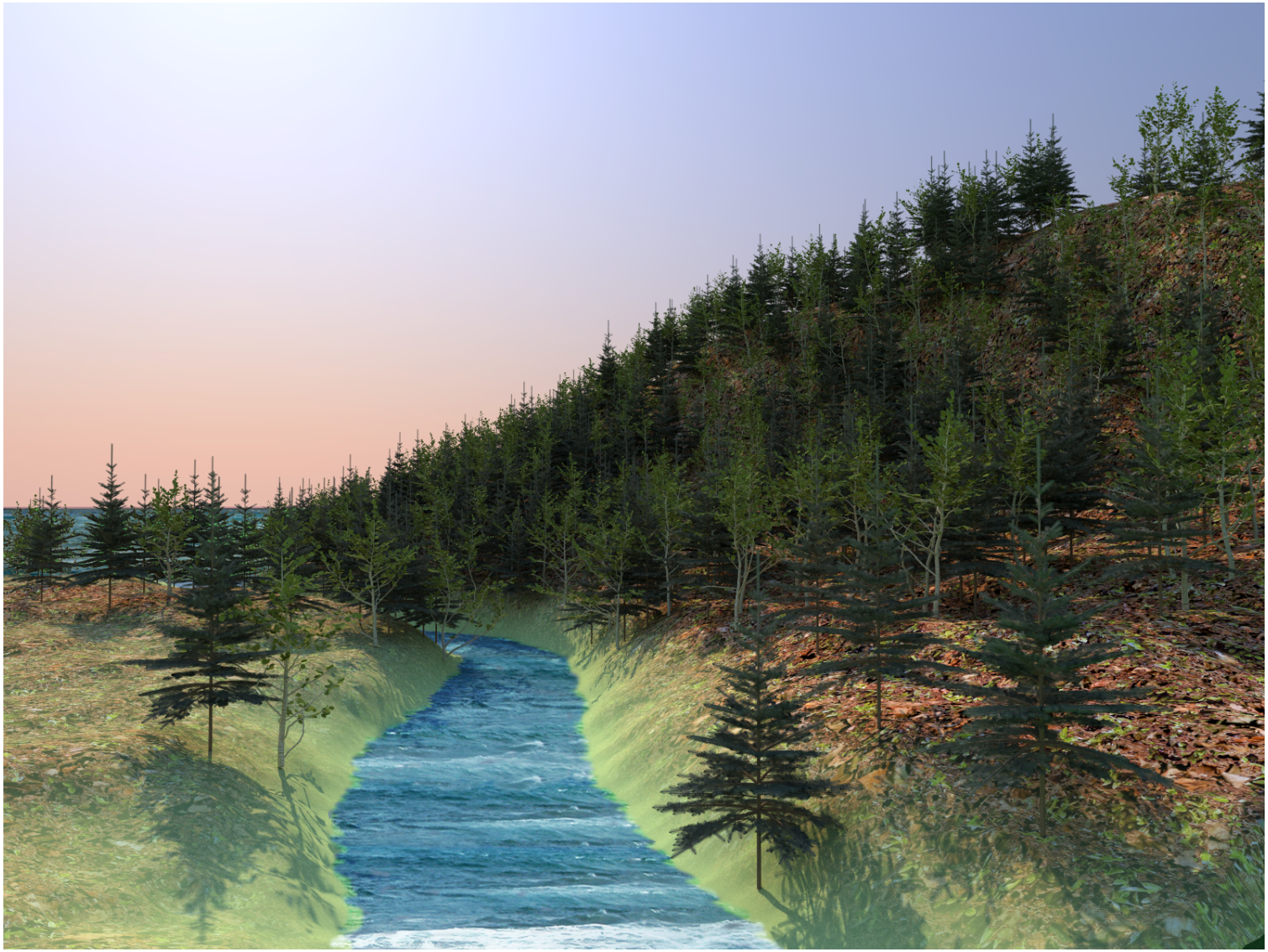
A model of a mixed stand, including competition for light

Reinhard Hemmerling

University of Göttingen, Göttingen, Germany

September 30th 2010



Overview

Files of the Model

`Main.xl`	– main program
`TreeModel.xl`	– tree model interface
`SpruceModel.xl`	– model of spruce, implements TreeModel
`BeechModel.xl`	– model of beech, implements TreeModel

Note: `*.xl` similar to `*.rgg`, but closer to Java

TreeModel.xl

Definition of a common interface for tree models

```
public interface TreeModel
{
  // initialise the tree model
  void init();

  // create a tree described by this model
  de.grogra.graph.impl.Node createTree();

  // perform a time step computation for
  // all trees created by this model
  void step();
}
```

Main.xl

Automatically find existing tree models and execute them

```
ArrayList models = new ArrayList();
protected void init() {
  // make sure list is empty
  models.clear();
  // get root of meta graph
  Node root = (Node) getGraph().getRoot(GraphManager.META_GRAPH);
  // add known instances of TreeModel to the list
  models.add((* root --> x:TreeModel *));
  // initialise every known model
  for (TreeModel tm : models) { tm.init(); }
}
public void run() {
  // execute tree models to advance growth by one year
  for (TreeModel tm : models) { tm.step(); }
}
```

Main.xl

Planting trees

```
for (int y : (1:MAX_ROWS)) {
  for (int x : (1:MAX_COLS)) {
    double z = height(volume(namedNode("Terrain")), x, y);
    TreeModel tm = chooseModel(x, y);
    if (probability(0.3)) [
      ==>> ^ Translate(x, y, z) tm.createTree();
    ]
  }
}
```

SpruceModel.xl

Implements an empirical model by Winfried Kurth[1]

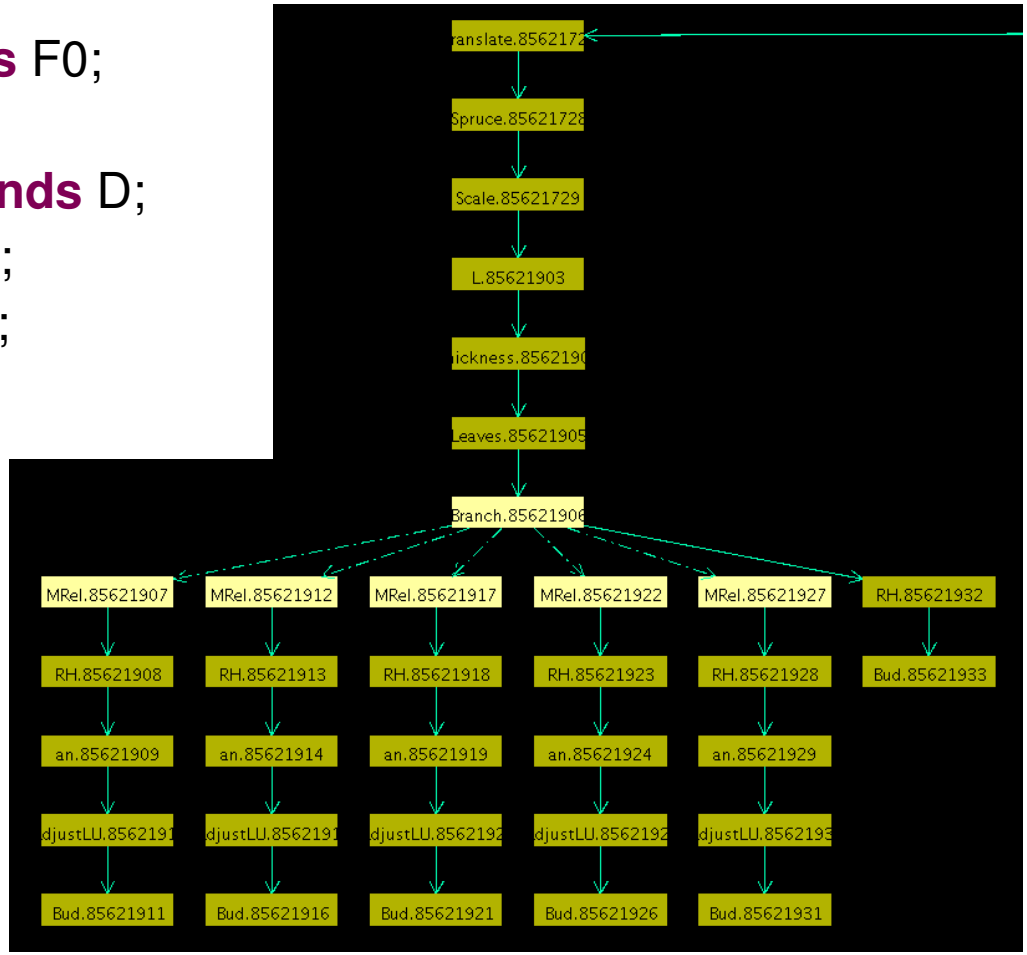
```
@de.grogra.xl.lang.ImplicitDoubleToFloat
public class SpruceModel
  extends RGG // base class of relational growth grammars
  implements TreeModel
{
        protected boolean isMainRGG ()
        {
                return false;
        }

        ...
}
```

[1] Winfried Kurth (1999) Die Simulation der Baumarchitektur mit Wachstumsgrammatiken: stochastische, sensitive L-Systeme als formale Basis für dynamische, morphologische Modelle der Verzweigungsstruktur von Gehölzen. Wiss. Verlag Berlin

SpruceModel.xl

Modules used in the spruce model:

module Branch(...) **extends** F0;
module Bud(...);
module Thickness(...) **extends** D;
module an(...) **extends** RU;
module bn(...) **extends** RV;
module Leaves;



SpruceModel.xl

Growth of the spruce (reduced)

```
Bud(v, o, p, t, r, s) ==>
  if (o == 0) ( // is main branch ?
    // create new segment of the branch
    L Thickness Leaves Branch
    // create new buds on top of the branch
    for (int i: (0:N−1)) ( [ MRel RH an AdjustLU Bud ] )
    // continue main branch
    RH Bud(v, o, p, t+1, r, s)
  ) else if (v < 2) ( // vitality too low ? => die
  ) else if (probability(1.06 − 0.0105∗v)) ( // branch stops growing
    L Thickness Leaves Branch
  ) else ( // branch continues growth
    L Thickness Leaves bn Branch
    for (int i: (0:M−1)) ( [ MRel cluster ])
    Bud
  );
```

SpruceModel.xl

Growth of the spruce (reduced)

```
Bud(v, o, p, t, r, s) ==>
  ...
    for (int i: (0:M−1)) ( [ MRel cluster ])
    Bud
  );

static void cluster(RGGProducer prod, long j, double v, ...)
{
  // produce some buds
  prod ==> ... [ M RU Bud ] ...;
}
```

Radiation model

Main.xl:

```
class Main {
  public static LightModel lm = new LightModel(LM_RAYS, LM_DEPTH);
  public void run() {
    lm.compute(); // run the light model
    for (TreeModel tm : models) tm.step(); // execute tree models
  }
}
```

SpruceModel.xl:

```
class SpruceModel {
  float calcLight(Bud bud) {
    Leaves l = (Leaves) ancestor(bud, Leaves.class);
    if (l == null) return 0;
    return Main.lm.getAbsorbedPower(l).integrate() / l.area();
  }
}
```

BeechModel.xl

```
module BeechLeaf extends Parallelogram(
  normal(0.06, 0.015), normal(0.043, 0.01)) {
    { setShader(leafShader); }
    float assimilatedCarbon;
}
```

- `leafShader` is scanned image of leaf as texture
- size of leaf is random (normal distribution)
- `assimilatedCarbon` stores amount of carbon assimilates in a single step (in mol C)

BeechModel.xl

```
abstract module Organ(super.length, int order)
  extends Cylinder(length, 0.0001) {
    float allocatedCarbon;
    float exportedCarbon;
    float preference = 1;
}
```

- `Organ` is superclass for internodes and single root compartment
- growth in length not explicitly modelled (flush growth)
- growth in diameter/girth in annual steps
- `allocatedCarbon` contains amount of carbon allocated by the organ as part of the basipetal transport of carbon
- `exportedCarbon` represents exportet part of carbon that is transported to the next organ in basipetal direction
- `preference` is used to compute sink strength

BeechModel.xl

```
module Beech extends Organ(0.01, 0);
module Internode(super.length, super.order) extends Organ;
module Bud(int order, float vitality, int sign,
  Internode internode, BeechLeaf leaf, Beech beech);
```

- `vitality` is a measure of the vitality of the bud and results from carbon assimilation and transport
- `sign` alternates between 1 and -1 to implement phyllotaxy

BeechModel.xl

Photosynthesis

```
public float assimilateCarbon() {
  float i = radiation.getAbsorbedPower(this).integrate() * TO_PPFD / AREA;
  float P = (αi+Pmax)/(2M) - sqrt(((αi+Pmax)/(2M))^2 - αiPmax/M);
  float c = AREA * Math.max(0, DURATION * P -
    SPECIFIC_MASS * C_FRACTION /
    (C_MASS * GROWTH_RESPIRATION_FACTOR_LEAF));
  this[assimilatedCarbon] = c;
  return c;
}
```

BeechModel.xl

Allocation and Transport

```
void transportCarbon(float imported, float assimilated) {
  float total = imported + assimilated;
  float q = radius / 0.0025;
  float exported = total * Math.exp(-0.7 * length * (1+q) / q);
  this[allocatedCarbon] = total - exported;
  this[exportedCarbon] = exported;
}

Beech -basipetal(Organ.class, children,
    BeechLeaf.class, leaves)-> x:Organ ::>
x.transportCarbon(
  sum(children[:][exportedCarbon]),
  sum(leaves[:].assimilateCarbon()));
```

BeechModel.xl

Allocation and Transport – Secondary Growth

```
float distributed = 0.7 * b[exportedCarbon];
b[allocatedCarbon] += b[exportedCarbon] - distributed;

float ds = sum((* b -descendants-> Organ *).distributionWeight());
(* b -descendants-> o:Organ *).grow(distributed * o.distributionWeight / ds);

void grow(float distributedCarbon) {
  float input = allocatedCarbon + distributedCarbon - maintenanceRespiration();
  if (input >= 0) {
    float m = input * GROWTH_RESPIRATION_FACTOR_WOOD *
      C_MASS / C_FRACTION;
    this[radius] = Math.sqrt(this[radius]**2 + m / (DENSITY * Math.PI * length));
  }
  this[mark] = (order > 0) && (input < 0);
}
```

BeechModel.xl

Self Pruning in Autumn

o:Organ, (o[mark]) ==>> ;
BeechLeaf ==>> ;

THANK YOU FOR YOUR ATTENTION