Tutorial and workshop „Modelling with GroIMP and XL“ / Tutorial for beginners

University of Göttingen, 27 February, 2012

Gerhard Buck-Sorlin and Winfried Kurth

Using the radiation model of GroIMP

A simple FSPM in 9 steps

- no real species, more sort of a typical basic shape and development

- (cf. Goethe: „Urpflanze“)

- mimics an annual plant (but can be modified and generalized)

- functional part will be (in the end): light interception, photosynthesis, transport of assimilates in the plant

- in the first versions: purely structural model of a plant

first version (`sfspm01.rgg`):

```
/* Steps towards a simple FSPM. sfspm01.rgg:
   A simple plant with leaves and branching is generated.
   Original version by G. Buck-Sorlin; modified. */

module Bud extends Sphere(0.1)
   {{ setShader(RED); }};

module Node extends Sphere(0.07)
   {{ setShader(GREEN); }};

module Internode extends F;

/* leaves are rectangles: */
module Leaf extends Parallelogram(2, 1);

const float G_ANGLE = 137.5;  /* golden angle */


/* simple plant, with leaves and branches: */
protected void init()
   [
   Axiom ==> Bud;
   ]

public void run()
   [
   Bud ==> Internode Node [ RL(50) Bud ] [ RL(70) Leaf ]
           RH(G_ANGLE) Internode Bud;
   ]
```

second version (`sfspm02.rgg`):

```
module Bud(int order) extends Sphere(0.0001)
   {{ setShader(RED); setRadius(0.2); }};

module Node extends Sphere(0.07)
   {{ setShader(GREEN); }};

module Internode extends F;

/* leaves are rectangles: */
module Leaf extends Parallelogram(2, 1);

const float G_ANGLE = 137.5;  /* golden angle */


/* simple plant, with leaves and branches: */
protected void init()
   [
   Axiom ==> Bud(0);
   ]

public void run()
   [
   Bud(o), (o < 3) ==> Internode Node [ RL(50) Bud(o+1) ]
                       [ RL(70) Leaf] RH(G_ANGLE) Internode Bud(o);
   ]
```

third version:

more precise timing for appearance of new metamers (internode, node, leaf)

phyllochron = time span between the appearances of new metamers in apical position at the same shoot axis

(often used as synonym: plastochron, but this means in its proper sense the time span between two *initiations* of new metamers)

(phyllochron: notion does not depend on growth being preformed or neoformed)

Time count in the model in discrete steps
(1 step = 1 parallel application of rules)

sfspm03.rgg

```
module Bud (int ph, int order) extends Sphere(0.1)
   {{ setShader(RED); }};

module Node extends Sphere(0.07)
   {{ setShader(GREEN); }};

module Internode extends F;

module Leaf extends Parallelogram(2, 1);

const float G_ANGLE = 137.5;  /* golden angle */


/* introducing a phyllochron */
const int phyllo = 25;

protected void init()
   [
   Axiom ==> Bud(phyllo, 0);
   ]

public void run()
   [
   Bud(p, o), (p > 0) ==> Bud(p-1, o);  /* first parameter
                                           counted down until...*/
   Bud(p, o), (p == 0 && o <= 1) ==> Internode Node
                  [ RL(50) Bud(phyllo, o+1) ] [RL(70) Leaf]
                  RH(G_ANGLE) Internode Bud(phyllo, o);
                  /* (order restricted to 1 ... for efficiency)   */
   ]
```

sfspm04.gsz: with flowers; textured organs

```
const ShaderRef leafmat = new ShaderRef("Leafmat");
const ShaderRef petalmat = new ShaderRef("Petalmat");
const ShaderRef internodemat = new ShaderRef("Internodemat");
const ShaderRef nodemat = new ShaderRef("Nodemat");

module Bud(int time, int ph, int order) extends Sphere(0.1)
   {{ setShader(nodemat); }};
module Node extends Sphere(0.07)
   {{ setShader(GREEN); }};
module NiceNode extends Sphere(0.07)
   {{ setShader(nodemat); }};
module Internode extends F(1, 0.1, 7);
module NiceInternode extends Cylinder(1, 0.08)
   {{ setShader(internodemat); }};
module Leaf extends Parallelogram(2, 1)
   {{ setColor(0x82B417); }};
module NiceLeaf extends Parallelogram(2,2)
   {{ setShader(leafmat); }};

module Flower ==>                          /* instantiation rule */
   RU(180) Cone(0.3, 0.3).(setColor(0x82B417)) M(-0.25) RL(90)
   [ for (int i=1; i<=5; i++) ( [ RU(i*360/5) RL(20)
                     Parallelogram(2, 1).(setColor(0xFF00FF)) ] ) ] RU(45)
   [ for (int i=1; i<=5; i++) ( [ RU(i*360/5) RL(40) F(0.3, 0.1, 14) RV(-0.3)
                     F(0.3, 0.1, 14) RV(-0.3) F(0.3, 0.1, 14) ] ) ] RU(-45)
   [ for (int i=1; i<=5; i++) ( [ RU(i*360/5) RL(70)
                     Frustum(0.7, 0.2, 0.05).(setColor(0x8DAF58)) ] )

module NiceFlower ==>
   RU(180) Cone(0.3, 0.3).(setShader(internodemat)) M(-0.25) RL(90)
   [ for (int i=1; i<=5; i++) ( [ RU(i*360/5) RL(20)
                     Parallelogram(2, 1).(setShader(petalmat)) ] ) ] RU(45)
   [ for (int i=1; i<=2; i++) ( [ RU(i*360/3) RL(40) F(0.3, 0.1, 14) RV(-0.3)
                     F(0.3, 0.1, 14) RV(-0.3) F(0.3, 0.1, 14) ] ) ] RU(-45)
   [ for (int i=1; i<=5; i++) ( [ RU(i*360/5) RL(70)
                     Frustum(0.7, 0.2, 0.05).(setColor(0x8DAF58)) ] ) ];
```

// sfspm04.gsz, continued

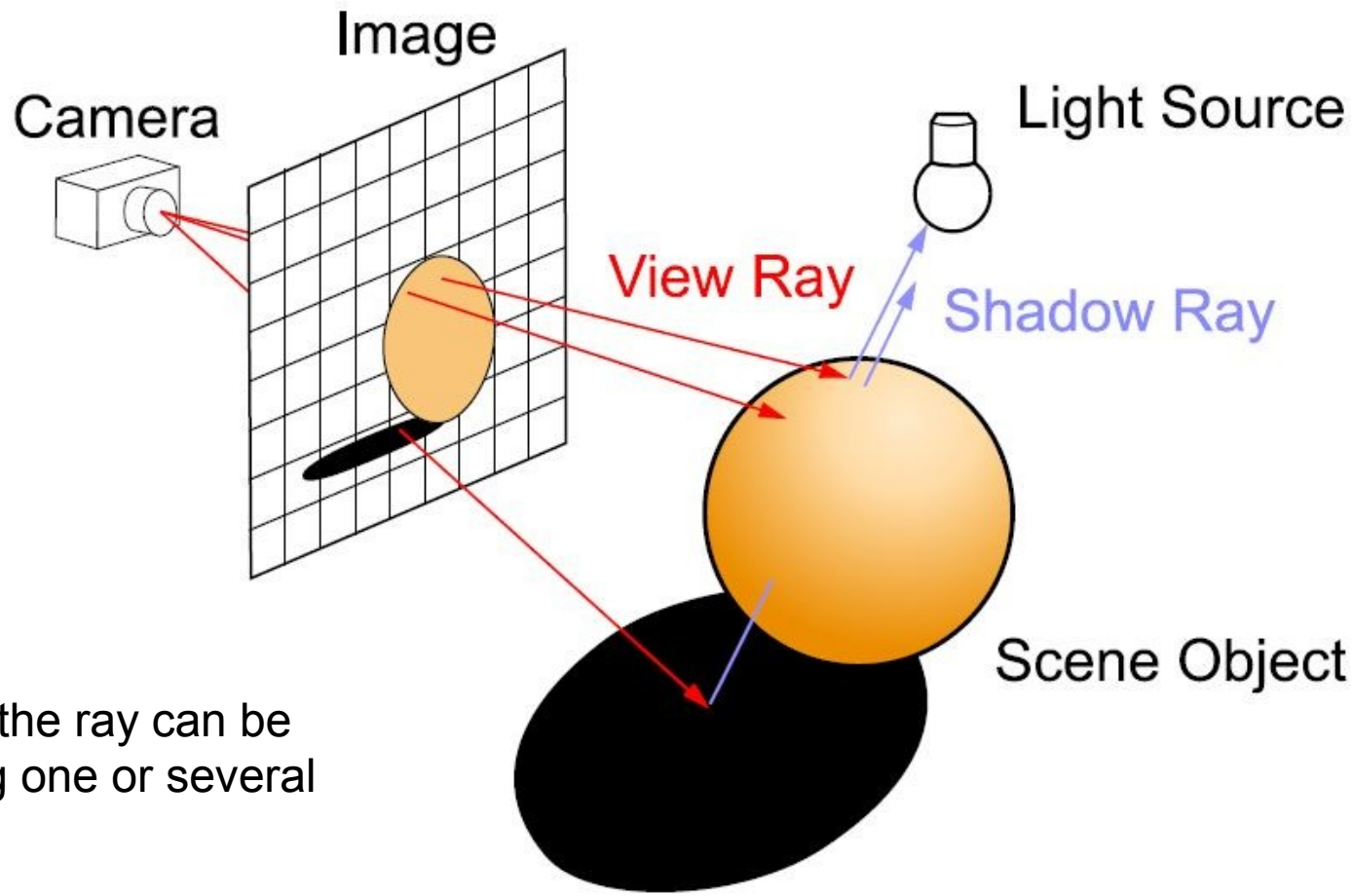
```
const float G_ANGLE = 137.5;  /* golden angle */

const int phyllo = 25;

protected void init()
   [
   Axiom ==> Bud(1, phyllo, 0);
   ]

public void run()
   [
   Bud(r, p, o), (p > 0)  ==> Bud(r, p-1, o);
   Bud(r, p, o), (r<10 && p==0 && o<=2) ==> RV(-0.1) NiceInternode NiceNode
         [ RL(50) Bud(r, phyllo, o+1) ] [ RL(70) NiceLeaf ] RH(G_ANGLE) RV(-0.1)
         NiceInternode Bud(r+1, phyllo, o);
   Bud(r, p, o), (r == 10) ==> RV(-0.1) NiceInternode RV(-0.1) NiceInternode NiceFlower;

   ]
```

additionally there are image files for the used textures, which are connected manually in GroIMP with the shader names „Leafmat“, „Petalmat“ etc. (see „Basic examples in XL, part 1“ = **`ws12_t02.ppt`**, last slides)

principle of radiation model

ray tracing – a method from computer graphics

basic idea:



additionally the ray can be traced along one or several reflections

from: http://en.wikipedia.org/wiki/File:Ray_trace_diagram.svg

we do not want to generate an image, but calculate for all leaves of the virtual plant the amount of intercepted light

$\rightarrow$reversal of the direction of the rays:

they run from the light sources to the objects. An extra shadow test is no longer necessary.

A large number of rays with random directions is generated: „Monte-Carlo ray tracing“

accumulation of the intercepted power of radiation (in the unit W = Watt) is possible for each object

Condition: there has to be a light source in the scene

`DirectionalLight, PointLight, SpotLight, Sky`

sfspm05.gsz *(only the new parts of the model are displayed:)*

```
// ..... module definitions .....

/* the leaf collects light and gets a new shader for the radiation model: */

module Leaf(float al) extends Parallelogram(2, 1)
   {{ setShader(new AlgorithmSwitchShader(new RGBAShader(0, 1, 0), GREEN)); }};

// ..... further module definitions .....


/* the light source: */

module MyLamp extends SpotLight
   {{
   setPower(200.0);                 /* power in W */
   setAttenuationDistance(50.0);    /* in m */
   setAttenuationExponent(0.0);
   setInnerAngle(22.5*Math.PI/180.0);
   setOuterAngle(30.0*Math.PI/180.0);
   }};

module MyLight extends LightNode(1.0, 1.0, 1.0)  /* R, G, B */
   {{ setLight(new MyLamp()); }};


/* the radiation model is defined */

LightModel lm = new LightModel(100000, 5);

   /* 100000: number of random rays, 5: recursion depth (nb. of reflections) */
```

sfspm05.gsz *(nur neue Teile des Modells dargestellt; Fortsetzung:)*

```
protected void init()
   [
   Axiom ==> Bud(1, phyllo, 0);
   ==> ^ M(50) RU(180) MyLight;    /* Light source is placed above the scene */
   ]

public void grow()
   {
   run();
   lm.compute();
   absorb();
   }

protected void run()
   [
   Bud(r, p, o), (p>0) ==> ..........
// further rules....
   ]

protected void absorb()
   [
   lf:Leaf ::>
      {
      lf[al] = lm.getAbsorbedPower3d(lf).integrate() * 2.25;
      lf.(setShader(new AlgorithmSwitchShader(
          new RGBAShader((float) lf[al]/5.0, (float) lf[al]*2, (float) lf[al]/100.0),
                                GREEN)));
      println(lf[al]);
      }
   ]
```

sfspm06.gsz: leaf growth according to logistic function, plotting the absorbed light in a chart *(new parts of model only:)*

```
/* the leaf is modelled as a 3-d box now and gets new parameters: */

module Leaf(super.length, super.width, float al, int age)
   extends Box(length, width, 0.01)
   {{ setShader(new AlgorithmSwitchShader(new RGBAShader(0, 1, 0), GREEN)); }};
// ............

/* Introducing leaf growth: */

/* derivative of logistic function */
public float logistic(float maxdim, int time, float phylloM, float slope)
   {
   return (slope * maxdim * Math.exp(-slope*(time-phylloM))) /
           ((Math.exp(-slope*(time-phylloM))+1)**2);
   }

/* Table for absorbed light values: */

const DatasetRef lightdata = new DatasetRef("Light intercepted by canopy");


protected void init()
   [
      {
      lightdata.clear();            /* the chart is initialized */
      chart(lightdata, XY_PLOT);
      }
   Axiom ==> Bud(1, phyllo, 0);
   ==> ^ M(50) RU(180) MyLight;     /* Light source is placed above the scene */
   ]
```

sfspm06.gsz: leaf growth according to logistic function, plotting the absorbed light in a chart *(new parts of model only; continued:)*

```
public void grow()
   {
   run();
   lm.compute();
   absorb_and_growleaf();
   lightdata.addRow().set(0, sum((* Leaf *)[al]));
   }

protected void run()
   [
   Bud(r, p, o), (p>0) ==> ..........
// further rules....
   ]

protected void absorb_and_growleaf()
   [
   lf:Leaf ::>
      {
      lf[al] = lm.getAbsorbedPower3d(lf).integrate();
      lf.(setShader(new AlgorithmSwitchShader(
          new RGBAShader((float) lf[al]/5.0, (float) lf[al]*2, (float) lf[al]/100.0),
                              GREEN)));
      println(lf[al]);
      lf[age]++;                                        /* the leaf is ageing */
      lf[length] += logistic(2, lf[age], 10, 0.5);  /* logistic growth    */
      lf[width] = lf[length]*0.7;
      }
   ]
```

sfspm07.gsz: determination of the light arriving at the soil

(new parts of model only:)

```
// .....
/* a single tile (will be positioned on the ground): */
module Tile(float len, float wid) extends Parallelogram(len, wid)
   { float al; };

// .....
protected void init()
   [
      {
      lightdata.clear();           /* the chart is initialized */
      chart(lightdata, XY_PLOT);
      }
   Axiom ==> [ RL(90) M(4) RU(90) M(-4) for ((1:40))   /* paving the ground */
                ( for ((1:40))
                        ( Tile(0.25, 0.25).(setShader(new RGBAShader(0.6, 0.3, 0.1))) )
                  M(-10) RU(90) M(0.25) RU(-90)
                ) ]
             Bud(1, phyllo, 0);
   ==> ^ M(50) RU(180) MyLight;    /* Light source is placed above the scene */
   ]
// .....
protected void absorb_and_grow()
   [
   lf:Leaf ::> ......
   p:Tile ::>
      {
      p[al] = lm.getAbsorbedPower3d(p).integrate();
      println(p[al]);
      p.(setShader(new AlgorithmSwitchShader(new RGBAShader(
            p[al]*300, p[al]*200, p[al]), new RGBAShader(0.6, 0.3, 0.1))));
      }
   ]
```

Most simple model of photosynthesis:

Assumption of a linear relationship between the absorbed light and the amount of assimilates produced in the leaf

- conversion factor CONV_FACTOR
- **`Leaf`** gets new property „**`as`**“ (produced amount of assimilates)

sfspm08.gsz: Usage of the linear model of photosynthesis

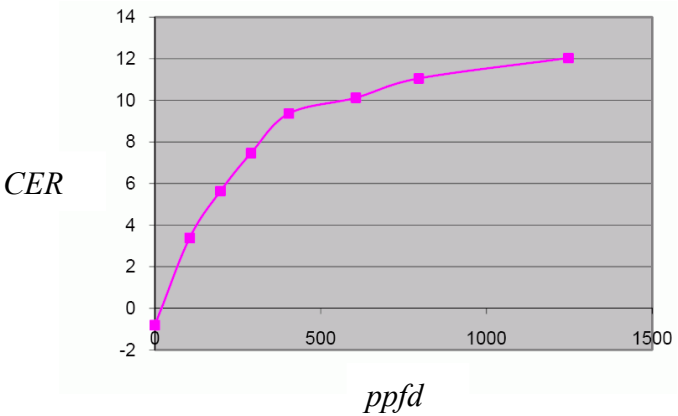
(new parts of model only:)

```
// .....
const float CONV_FACTOR = 0.2;  /* conversion factor light->assimilates */
// .....

protected void absorb_and_grow()
   [
   lf:Leaf ::>
      {
      lf[al] = lm.getAbsorbedPower3d(lf).integrate()*2.25;
// ...................
      lf[as] = lf[al] * CONV_FACTOR;                /* amount of assimilates */

      float lfas = sum((* Leaf *)[as]);               /* ... of all leaves  */
      if (lfas > 0)   /* dependency of growth on availability of assimilates */
         {
         lf[length] += logistic(2, lf[age], 10, 0.5);  /* logistic growth    */
         }
      lf[width] = lf[length]*0.7;
      }
   itn:Internode ::> // ......
   ]
```

Inclusion of a more realistic (non-linear) model of photosynthesis

CO_2 exchange rate (*CER*): saturation curve in dependence of photon flux densitiy (*ppfd*) according to

$$CER = \frac{\left(F_{\max} + RD\right) \cdot PE \cdot ppfd}{PE \cdot ppfd + F_{\max} + RD} - RD$$



with *RD* = dark respiration
PE = photosynthetic efficiency
F_{max} = maximal photosynthesis

Units:
CER, *ppfd*, *RD*, F_{max} : µmol · m^{-1}· s^{-1} ; *PE* : dimensionless

sfspm09.gsz: Photosynthesis in the leaves with improved model

(calculation of photosynthesis only:)

```
/* function calculateCER gives the instantaneous CO2 fixation rate
   (micromol CO2 m-2 s-1) depending on light intensity (ppfd).
   Dependency on temperature is not included. */

float calculateCER(float ppfd)
   {
   return (float) ( (FMAX+DARK_RESPIRATION_RATE) * PHOTO_EFFICIENCY * ppfd) /
      (PHOTO_EFFICIENCY*ppfd + FMAX + DARK_RESPIRATION_RATE) - DARK_RESPIRATION_RATE;
   }
```

Conversion of the amount of assimilates in kg for a leaf of a certain area and during a given time span:

```
/* function calculatePS gives the assimilate production (in kg) of a leaf,
   depending on leaf area a (in m**2), ppfd (in umol/(m**2 s)) and duration
   d (in seconds) of light interception.
   Dependency on temperature is not included. */

float calculatePS(float a, float ppfd, float d)
   {
   return
      calculateCER(ppfd) * a * d
                 * 44.01e-6            /* mass of 1 umol CO2 in g   */
                 * (180.162/264.06)    /* conversion CO2 -> Glucose */
                 / 1000.0;             /* conversion g -> kg        */
   }
```

conversion of photon flux from W (power) in µmol · s^{-1} :

```
const float PPFD_FACTOR = 0.575;     /* conversion factor from absorbed
                                        power (W) to photon flux (umol/s);
                                        unit: umol/J;   after Kniemeyer 2008 */
// ............

lf:Leaf ::>
      {
      lf[al] = lm.getAbsorbedPower3d(lf).integrate();
//    ................
      float area = LEAF_FF * lf[length] * lf[width] / 10000.0;
                                          /* converted from cm**2 to m**2 */

      /* calculation of photosynthetic production of the leaf: */
      lf[as] += calculatePS(area, PPFD_FACTOR * lf[al] / area, DURATION);
//    ................
      }
```

required for the distribution of the assimilates:

modelling of transport processes

model assumption: substrate flows from elements with high concentration to elements with low concentration (principle of *diffusion*)

example:

sm09_e41.rgg (concentration of a substrate is visualized by the diameter here)

```
module Internode(super.diameter) extends F(100, diameter);

protected void init()
    [
    Axiom ==> P(14) Internode(1) P(2) Internode(1)
              P(4) Internode(1) P(15) Internode(60);
    ]

public void transport()
    [
    i_above:Internode < < i_below:Internode ::>
       {
       float r = 0.1 * (i_below[diameter] - i_above[diameter]);
       i_below[diameter] :-= r;
       i_above[diameter]  :+= r;
       }
    ]
```

(two reverse successor edges after the other)

modelling of transport in `sfspm09.gsz` :

```
const float DIFF_CONST = 0.01;        /* diffusion constant for transport
                                         of assimilates */
// ..........

public void grow()
   {
   run();
   lm.setSeed(irandom(1,100000));
   lm.compute();
   absorb_and_grow();
   for (apply(5)) transport();      /* 5 iterations of transport per step */
// ..........
   }

protected void transport()
   [
   /* transport of assimilates from a leaf to the supporting internode: */
   lf:Leaf <-minDescendants- itn:Internode ::>
      {
      float r = DIFF_CONST * (lf[as] - itn[as]);
      lf[as] :-= r;
      itn[as] :+= r;
      }

   /* exchange between successive internodes: */
   i_top:Internode <-minDescendants- i_bottom:Internode ::>
      {
      float r = DIFF_CONST * (i_bottom[as] - i_top[as]);
      i_bottom[as] :-= r;
      i_top[as] :+= r;
      }
   ]
```

open questions / deficiencies of the model:

- is this transport mechanism realistic?

- how are the conditions in the very beginning?

- what about buds which do not yet produce assimilates but need some for extension growth?

- For growth and photosynthesis, nitrogen (N) is needed, too. This is delivered by the roots. How would a transport model for N differ from that for C?