How to develop a simple FSPM with GroIMP

Katarína Streit (Smoleňová), Gerhard Buck-Sorlin

ksmolen@gwdg.de
Departement Ecoinformatics, Biometry and Forest Growth
Georg-August-University Göttingen

September 18, 2013 / Prague

“Modelling of Ecosystems by Tools from Computer Science”

A simple FSPM in 9 steps

- In the first versions: purely structural model of a plant
- Later extended by functional part: light interception, photosynthesis, transport of assimilates in the plant
- No real species, more sort of a typical basic shape and development
- Mimics an annual plant (but can be modified and generalized)

Step 1

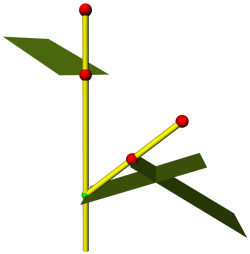
- A simple plant with leaves and branching

Step 1

```
// buds are red spheres
module Bud extends Sphere(0.1) {
    { setShader(RED); }
}

// nodes are green spheres
module Node extends Sphere(0.07) {
    { setShader(GREEN); }
}

// internodes are yellow cylinders
module Internode extends Cylinder(1, 0.05) {
    { setShader(YELLOW); }
}

// leaves are green rectangles
module Leaf extends Parallelogram(2, 1) {
    { setColor(0x82B417); }
}
```

Step 1

```
const float GOLDEN_ANGLE = 137.5;
const float BRANCH_ANGLE = 50;
const float LEAF_ANGLE = 70;

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

public void run()
[
    Bud ==>
        Internode Node
        [ RL(BRANCH_ANGLE) Bud ] [ RL(LEAF_ANGLE) Leaf ]
        RH(GOLDEN_ANGLE) Internode Bud
    ;
]
```

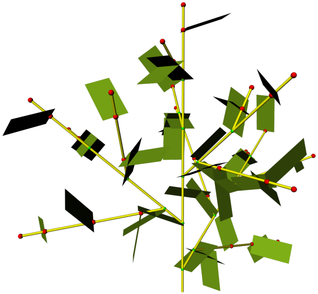


Step 2

- Restricting branching order

Step 2

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

 ...

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

public void run()
[
    Bud(o), (o < 4) ==>
        Internode Node
        [ RL(BRANCH_ANGLE) Bud(o+1) ] [ RL(LEAF_ANGLE) Leaf ]
        RH(GOLDEN_ANGLE) Internode Bud(o)
    ;
]
```



Step 3

- Introducing more precise timing for appearance of new metamers (internode, node, leaf)
- Phyllochron = time span between the appearance of new metamers
  (often used as synonym: plastochron, but this means in its proper sense the time span between two initiations of new metamers)
- Time count in the model in discrete steps
  (1 step = 1 parallel application of rules)

Step 3

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

 ...

const int PHYLLOCHRON = 25;

protected void init()
[
    Axiom ==> Bud(PHYLLOCHRON, 1);
]
```

Step 3

```
public void run()
[
    Bud(p, o), (p > 0) ==> Bud(p-1, o);

    Bud(p, o), (p == 0 && o < 4) ==>
        Internode Node
        [ RL(BRANCH_ANGLE) Bud(PHYLLOCHRON, o+1) ]
        [ RL(LEAF_ANGLE) Leaf ]
        RH(GOLDEN_ANGLE) Internode Bud(PHYLLOCHRON, o)
    ;
]
```

Step 4

- Introducing flowering
- Textured organs

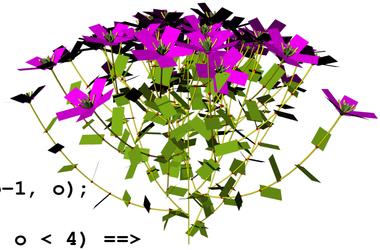
Step 4

```
module Flower ==>
    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)) ]
        )
    ]
;
```

Step 4

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

 ...

protected void init()
[
    Axiom ==> Bud(1, PHYLLOCHRON, 1);
]
```

Step 4



```
public void run()
[
    Bud(r, p, o), (p > 0) ==> Bud(r, p-1, o);

    Bud(r, p, o), (r < 10 && p == 0 && o < 4) ==>
        RV(-0.1) Internode Node
        [ RL(BRANCH_ANGLE) Bud(r, PHYLLOCHRON, o+1) ]
        [ RL(LEAF_ANGLE) Leaf ]
        RH(GOLDEN_ANGLE) RV(-0.1) Internode Bud(r+1, PHYLLOCHRON, o)
    ;

    Bud(r, p, o), (r == 10) ==>
        RV(-0.1) Internode RV(-0.1) Internode Flower;
]
```

Step 4

After importing textures into GroIMP:

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

Step 4

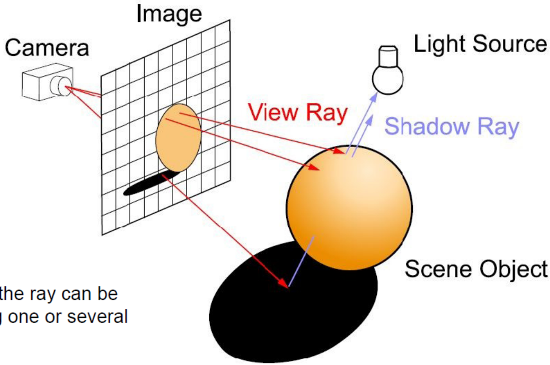
```
module Bud(int rank, int phyllo, int order) extends Sphere(0.1) {
    { setShader(nodemat); }
}

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

module Internode extends Cylinder(1, 0.05) {
    { setShader(internodemat); }
}

module Leaf extends Parallelogram(2, 1) {
    { setShader(leafmat); }
}

module Flower ==>
    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)) ]
        )
    ]
    ...
;
```

Step 5

- Introducing light interception of leaves
- Adding light sources into the scene
- Use of radiation model of GroIMP

Principle of radiation model

- Raytracing - a method from computer graphics



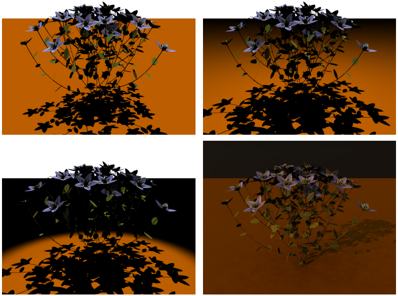
additionally the ray can be traced along one or several reflections

http://en.wikipedia.org/wiki/File:Ray_trace_diagram.svg (modif.)

- 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

Light sources in GroIMP

- `DirectionalLight`
- `PointLight`
- `SpotLight`
- `Sky` (SunSky Shader)

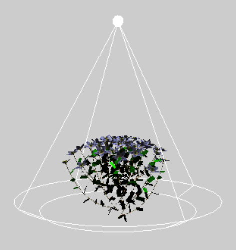


Adding a light source into the scene

```
// 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);
    }
}

// light specification with colour (R, G, B)
module MyLight extends LightNode(1.0, 1.0, 1.0) {
    { setLight(new MyLamp()); }
}
```

Adding a light source into the scene

```
protected void init()
[
    Axiom ==> Bud(1, PHYLLOCHRON, 1);

    // light source is placed
    // above the scene
    ==>> ^ M(50) RU(180) MyLight;
]
```



Adding a light model into the scene

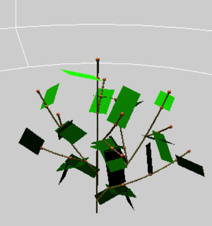
```
 ...

// light model instance
// 100000: number of random rays
// 5: recursion depth (nb. of reflections)
LightModel lm = new LightModel(100000, 5);

 ...

public void grow() {
    // apply growth rules
    run();
    // compute light
    lm.compute();
    // calculate the amount of light
    // (integrated over the whole spectrum),
    // absorbed by a leaf
    absorb();
}
```

Adding a light model into the scene

```
 ...

module Leaf(float al) extends Parallelogram(2, 1) {
    { setShader(leafmat); }
}
 ...

protected void run() [
    ...
    Bud(r, p, o), (r < 10 && p == 0 && o < 4) ==> ...  Leaf(0) ... ;
    ...
]

protected void absorb() [
    lf:Leaf ::> {
        // 2.25 - conversion factor W -> PAR
        // (photosynthetically active radiation [mikro mol/m^2/s])
        lf[al] = lm.getAbsorbedPower3d(lf).integrate() * 2.25;
        //println(lf[al]);
    }
]
```

AlgorithmSwitchShader(guiShader, radiationShader)

```
module Leaf(float al) extends Parallelogram(2, 1) {
    {
        // the leaf gets a new shader
        // for the radiation model
        setShader(new AlgorithmSwitchShader(
            new RGBAShader(0, 1, 0), GREEN));
    }
}

protected void absorb() [
    lf:Leaf ::> {
        lf[al] = lm.getAbsorbedPower3d(lf).integrate() * 2.25;
        //println(lf[al]);

        lf.(setShader(new AlgorithmSwitchShader(
            new RGBAShader(lf[al]/5.0, lf[al]*2, lf[al]/100.0),
            GREEN
        )));
    }
]
```



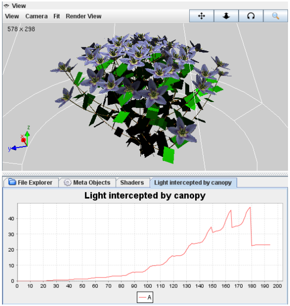
Step 6

- Plotting the absorbed light in a chart
- Leaf growth according to logistic function
- Internode growth according to logistic function

Plotting the absorbed light

```
 ...

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

protected void init()
[
    { initChart(); }
    ...
]

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

Plotting the absorbed light

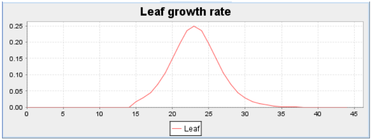



```
...

protected void initChart() {
    lightdata.clear();
    chart(lightdata, XY_PLOT);
}

protected void updateChart() {
    lightdata.addRow().set(0, sum((* Leaf *)[al]));
}
```

Panels → Explorers → Datasets

Leaf growth according to logistic function

```
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));
    }
}

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);
}
```



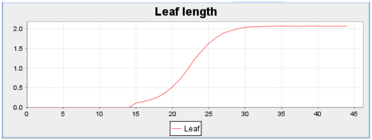
$$\frac{dW}{dt} = \frac{kW_{\max} e^{(-k(t-tm))}}{(e^{(-k(t-tm))}+1)^2}$$

Leaf growth according to logistic function

```
public void grow()
{
    run();
    lm.compute();
    absorbAndGrow();
    updateChart();
}

protected void run()
[
    ...

    Bud(r, p, o), (r < 10 && p == 0 && o < 4) ==>
        ... Leaf(0.1, 0.07, 0, 1) ... ;

    ...
]
```

Leaf growth according to logistic function

```
protected void absorbAndGrow()
[
    lf:Leaf ::> {

        ...

        lf[age]++;

        lf[length] += logistic(2, lf[age], 10, 0.5);
        lf[width] = lf[length] * 0.7;
    }
]
```



Internode growth according to logistic function

```
module Internode(super.length, int age) extends Cylinder(length, 0.05) {
    { setShader(internodemat); }
}
 ...
protected void run()
[
    ...
    Bud(r, p, o), (r < 10 && p == 0 && o < 4) ==>
        RV(-0.1) Internode(0.1, 1) Node
        [ RL(BRANCH_ANGLE) Bud(r, PHYLLOCHRON, o+1) ]
        [ RL(LEAF_ANGLE) Leaf(0.1, 0.07, 0, 1) ]
        RH(GOLDEN_ANGLE) RV(-0.1) Internode(0.1, 1) Bud(r+1, PHYLLOCHRON, o)
    ;

    Bud(r, p, o), (r == 10) ==>
        RV(-0.1) Internode(0.05, 1) RV(-0.1) Internode(0.05, 1) Flower;
]

protected void absorbAndGrow()
[
    lf:Leaf ::> { ...  }

    itn:Internode ::> {
        itn[age]++;
        itn[length] += logistic(1, itn[age], 10, 0.5);
    }
]
```

Step 7

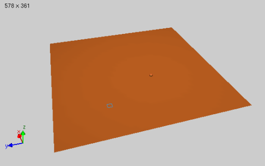
- Determination of the light arriving at the soil

Determination of the light arriving at the soil

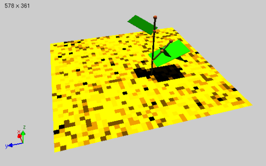
```
// light absorbing tile
module Tile(float len, float wid)
    extends Parallelogram(len, wid) {
    float al;
}
```

Determination of the light arriving at the soil

```
protected void init() [
    ...
    Axiom ==>
        [
            RL(90) M(4) RU(90) M(-4)
            for ((1:40)) (
                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, PHYLLOCHRON, 1)
    ;
    ...
]
```



Determination of the light arriving at the soil




```
protected void absorbAndGrow()
[
    lf:Leaf ::> { ...  }

    itn:Internode ::> { ...  }

    t:Tile ::> {
        t[al] = lm.getAbsorbedPower3d(t).integrate();
        //println(t[al]);

        t.(setShader(new AlgorithmSwitchShader(
            new RGBAShader(t[al]*300, t[al]*200, t[al]),
            new RGBAShader(0.6, 0.3, 0.1)
        )));
    }
]
```

Step 8

- Introducing leaf photosynthesis
- Simple model of photosynthesis: linear relationship between the absorbed light and the amount of assimilates produced in the leaf
- Ageing of flowers

Linear model of photosynthesis

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

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

protected void run()
[
    ...
    Bud(r, p, o), (r < 10 && p == 0 && o < 4) ==>
        ...  Leaf(0.1, 0.07, 0, 1, 0) ...  ;
    ...
]
```

Linear model of photosynthesis

```
protected void absorbAndGrow()
[
    lf:Leaf ::> {
        ...

        // amount of assimilates
        lf[as] = lf[al] * CONV_FACTOR;

        // amount of assimilates of all leaves
        float lfas = sum((* Leaf *)[as]);

        // dependency of growth on availability of assimilates
        if (lfas > 0) {
            lf[length] += logistic(2, lf[age], 10, 0.5);
        }

        lf[width] = lf[length] * 0.7;
    }
]
```

Aging of flowers

```
module Flower(int age, int max_age) ==> ... ;

 ...

public void run()
[
    ...

    Bud(r, p, o), (r == 10) ==> ...  Flower(1, irandom(10, 15));

    // ageing of flower
    Flower(t, m), (t < m) ==> Flower(t+1, m);

    // flower death
    Flower(t, m), (t >= m) ==> ;
]
```

Step 9

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

Photosynthesis in the leaves with improved model

- CO2 exchange rate (*CER*): saturation curve in dependence of photon flux density (*ppfd*) according to:

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

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

Units: CER, $ppfd$, RD, F_{max}: $\mu mol\ m^{-2} s^{-1}$; PE: dimensionless

```
const float FMAX = 20.0;
const float PHOTO_EFFICIENCY = 0.15;
const float DARK_RESPIRATION_RATE = 0.5;

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

Assimilate production of a leaf (in kg)

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

```
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
}
```

```
// leaf form factor
const float LEAF_FF = 0.7;
// conversion factor from absorbed power (W)
// to photon flux (umol/(m^2 s))
const float PPFD_FACTOR = 0.575;
// duration of a timestep in sec, here: 8 hours
const float DURATION = 8 * 60 * 60;
 ...

protected void absorbAndGrow() [
    lf:Leaf ::> {
        ...
        float area = LEAF_FF * lf[length] * lf[width];
        // area converted from cm^2 to m^2
        area /= 10000.0;

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

Distribution of the assimilates

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

```
module Internode(super.length, int age)
    extends Cylinder(length, 0.05) {
    { setShader(internodemat); }

    float as = 0;
}

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

```
public void grow() {
    ...
    transport();
    updateChart();
}

protected void transport() [
    // transport 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;
    }
]
```

Thank you for your attention!