

# TD 3 - UE

# Modélisation de l'architecture racinaire (RSA)

Christophe Pradal

[christophe.pradal@cirad.fr](mailto:christophe.pradal@cirad.fr)

CIRAD / Inria

# Formalisme: L-systems

- Introduit par A. Lindenmayer en 1968
  - Simulation of multi-cellular organisms
- Système dynamique à structure Dynamique [Giavitto, 01]
- Adapté à la modélisation de la croissance des plantes. Utilisé dans la communauté FSPM
- Contribution importante de Prusinkiewicz et al. en infographie / Modélisation



# L-systems

- Un L-systems se définit par alphabet  $V$ , un axiome  $w$  et un ensemble de règles de productions  $P$ .

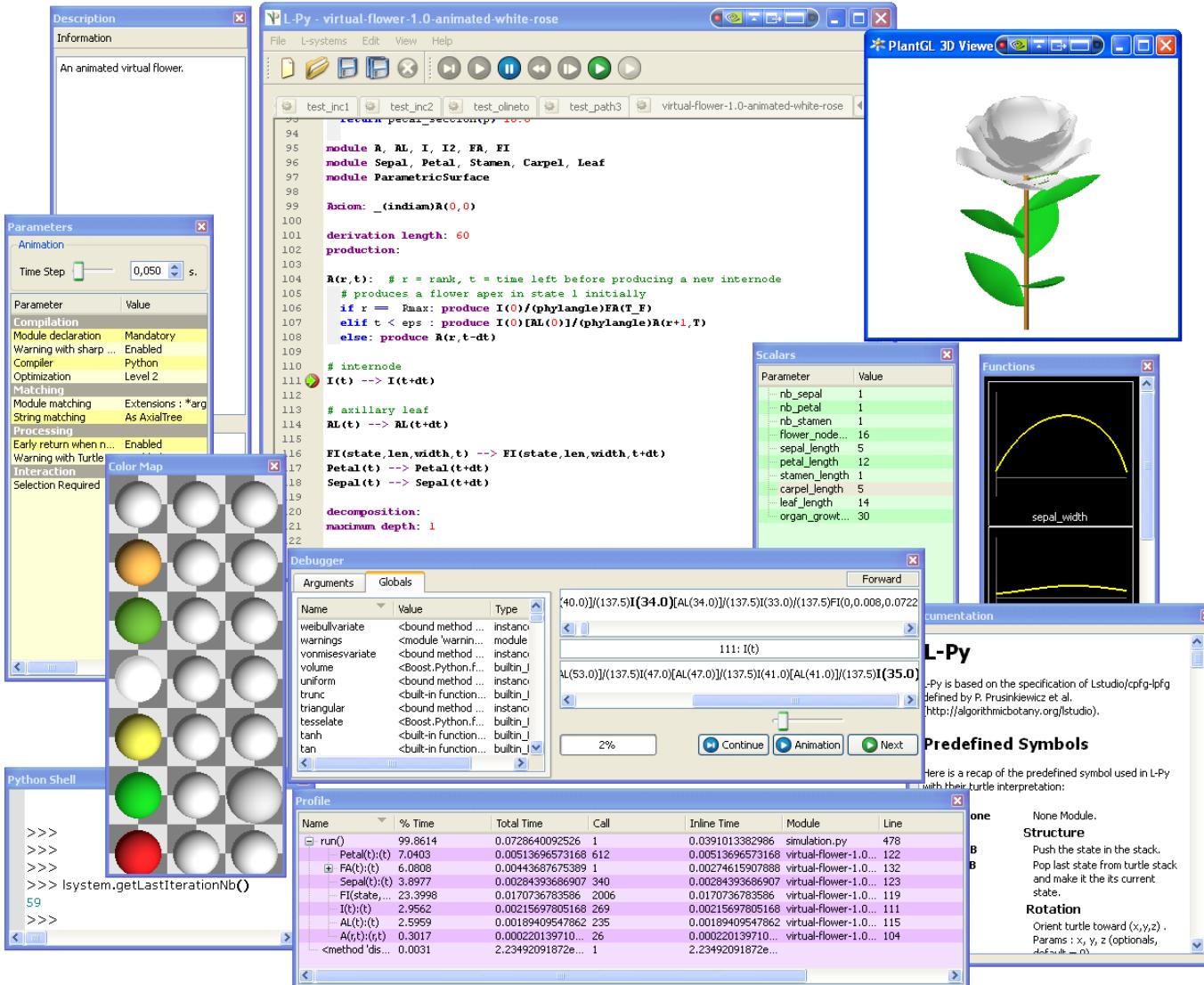
$$G = \langle V, w, P \rangle$$

- Les Productions sont appliquées sur une “string” en parallèle.

*left context < predecessor > right context → successor*

# Le logiciel libre L-Py

[lpy.readthedocs.io](http://lpy.readthedocs.io)



# L-Py Syntax

- Combining L-systems and python

```
module A, I
Axiom: A
derivation length: 5
production:
A --> I / [+A] A
I --> II

interpretation:
I --> F
endlsystem
```

# L-Py Syntax

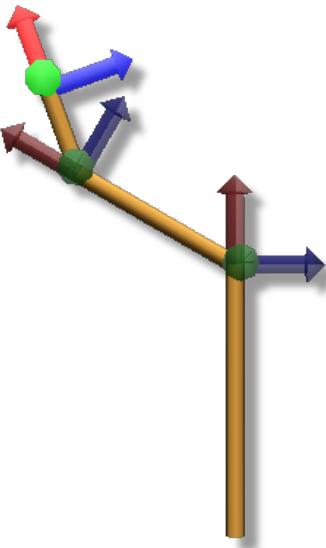
- Combining L-systems and python

```
from random import random
MAX_AGE, dr = 10, 0.03 # constants
module Apex(age), Internode(length, radius)
Axiom: Apex(0)
derivation length: 5
production:
Apex(age) :
    if age < MAX_AGE:
        produce Internode(1+ random(), 0.3)
            / (137) [+ (30) Apex(age+1) ] Apex(age+1)

Internode(l,r) --> Internode(l,r+dr)
interpretation:
Internode(l,r) --> _(r) F(l)
endlsystem
```

# Turtle Interpretation

- PlantGL Turtle



Documentation

## L-Py

L-Py is based on the specification of Lstudio/cpfg-lpfg defined by P. Prusinkiewicz et al. (<http://algorithmicbotany.org/lstudio>).

### Predefined Symbols

Here is a recap of the predefined symbol used in L-Py with their turtle interpretation:

	None	None Module.	Structure
[	SB	Push the state in the stack.	
]	EB	Pop last state from turtle stack and make it the its current state.	
<b>Pinpoint</b>			
<b>PinpointRel</b>			
@R			
+			
-			
^			
&			
/			
\			
iRollL			
iRollR			
@v			
<b>@M</b>			
<b>MoveRel</b>			
@Dd			
@Di			
@D			
F			
f			
@Gc			
@Ge			
{			
}			
<b>LineTo</b>			
<b>OLineTo</b>			
<b>SetHead</b>			
Left		Turn left around Up vector. Params : angle (optional), in degrees).	
Right		Turn right around Up vector. Params : angle (optional), in degrees).	
Up		Pitch up around Left vector. Params : angle (optional), in degrees).	
Down		Pitch down around Left vector. Params : angle (optional), in degrees).	
RollL		Roll left around Heading vector. Params : angle (optional), in degrees).	
RollR		Roll right around Heading vector. Params : angle (optional), in degrees).	
		Roll left intrinsically around Heading vector. Params : angle (optional), in degrees).	
		Roll right intrinsically around Heading vector. Params : angle (optional), in degrees).	
		Turn around 180deg the Up vector.	
		Turn to Vertical : Roll the turtle around the H axis so that H and U lie in a common vertical plane with U closest to up	
<b>Position</b>			
<b>MoveTo</b>		Set the turtle position. Params : x, y, z (optionals, default = 0).	
		Move relatively from current the turtle position. Params : x, y, z (optionals, default = 0).	
<b>DivScale</b>			
<b>MultScale</b>		Divides the current turtle scale by a scale factor. Params : scale factor (optional, default = 1.0).	
<b>SetScale</b>		Multiplies the current turtle scale by a scale factor. Params : scale factor (optional, default = 1.0).	
		Set the current turtle scale. Params : scale (optional, default = 1.0).	
<b>Primitive</b>			
F		Move forward and draw. Params: length , topRadius.	
f		Move forward and without draw. Params: length.	
StartGC		Start a new generalized cylinder.	
EndGC		Pop generalized cylinder from the stack and render it.	
BP		Start a new polygon.	
EP		Pop a polygon from the stack and render it.	
		Trace line to (x,y,z) without changing the orientation. Params : x, y, z (optionals, default = 0).	
		Trace line toward (x,y,z) and change the orientation. Params : x, y, z (optionals, default = 0).	

**F(3) [+ (60) F(2) [-(40) F(1) ] ]**

Introspection

- Debugging PlantGL Turtle

**Frame F(3) [+ (60) Frame F(2) [-(40) Frame F(1) ] ]**

# Branch Geometry

- Use of tropism to change orientation of branch

```
length = 10
```

```
Axiom: Elasticity(0.02) Branch(length)
```

```
derivation length: 1
```

```
production:
```

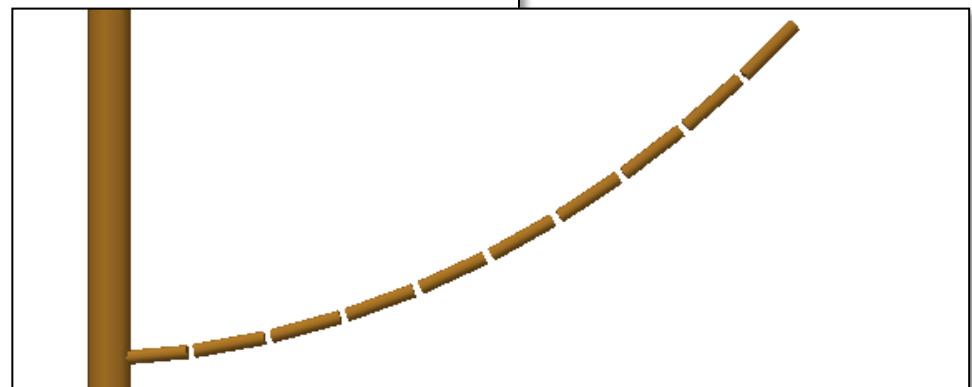
```
interpretation:
```

```
Branch(l) :
```

```
    for i in xrange(l):
```

```
        nproduce f(0.1) F(1)
```

```
endsystem
```



# Branch Geometry

- Use of predefined geometrical embedding

```
length = 10
```

**Axiom:** `SetGuide(path,length)` Branch(length)

**derivation length:** 1

**production:**

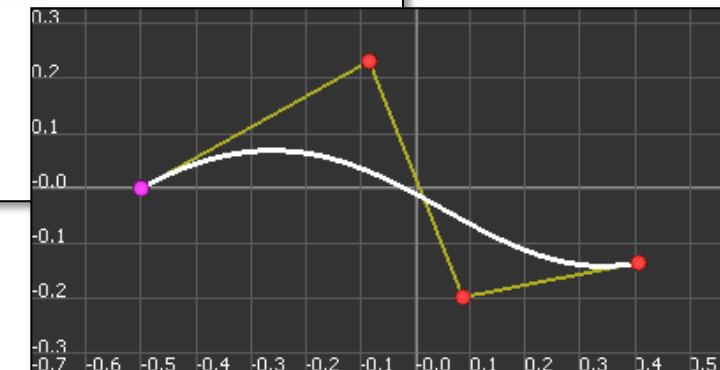
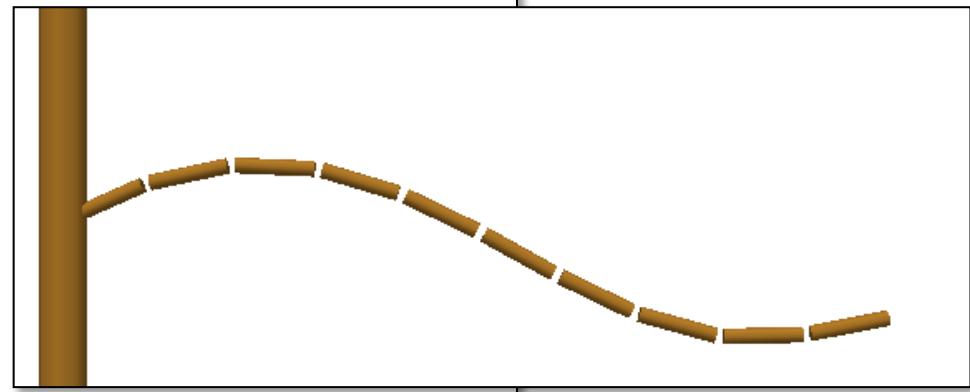
**interpretation:**

**Branch(1) :**

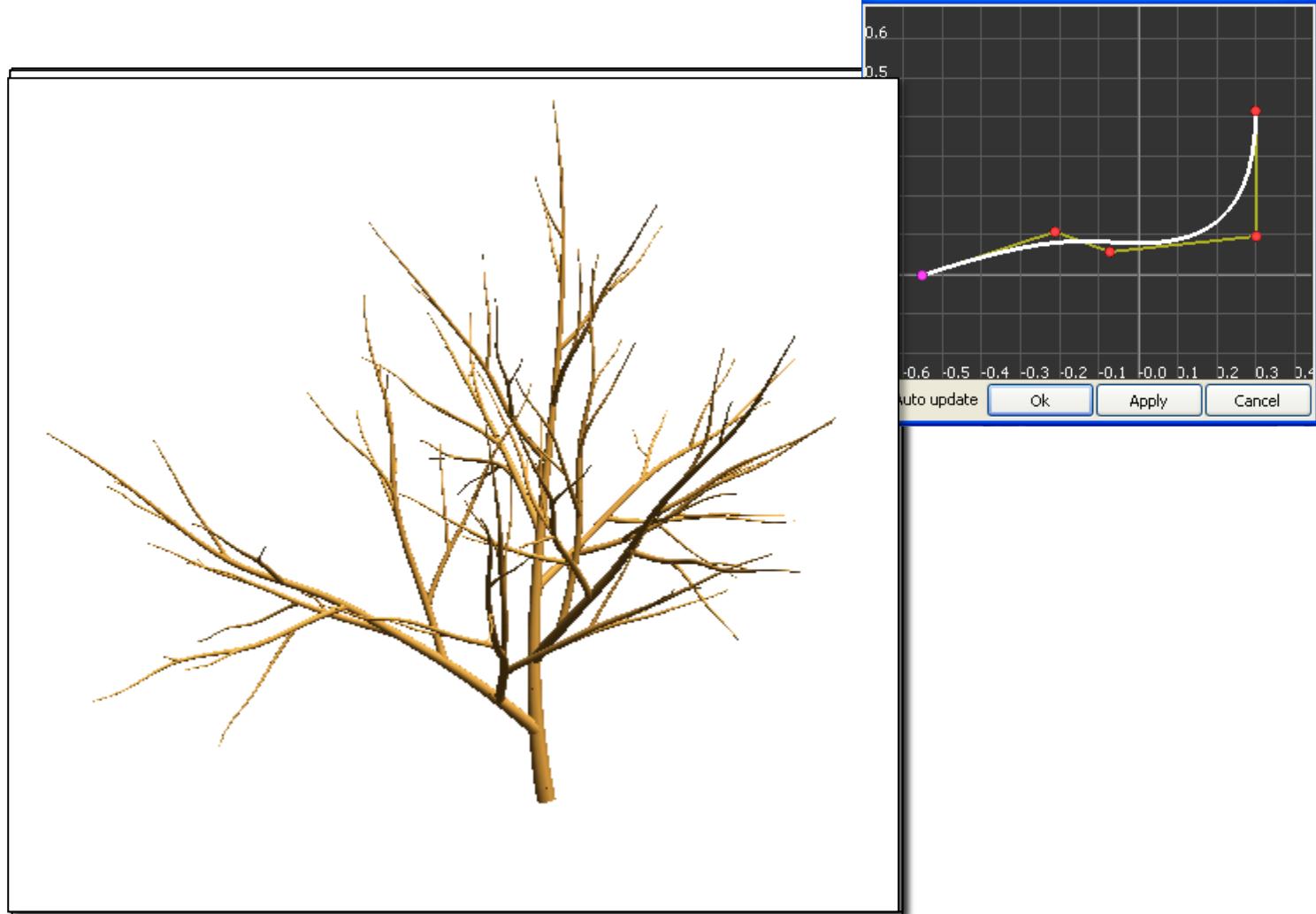
```
for i in range(l):
```

**nproduce f(0.1) F(1)**

**endsystem**



# Example on a simple tree structure



# L-systems features

- Parametric L-systems
- Contexts and new contexts sensitivity
- Stochastic L-systems

```
A(x) :  
  if x < 3:  
    produce B(2*x)
```

```
C(z) < B(y) << A(x) > D(w) :
```

```
A(x) :  
  if random() < prob1 :  
    produce B(2*x)  
  else : produce C(0)
```

# L-systems features

- Parametric L-systems
- Contexts and new contexts sensitivity
- Stochastic L-systems
- Environmental Interaction
- Pruning
- Group of rules
- Forward and Backward application, ...

```
A(x) :  
  if x < 3:  
    produce B(2*x)
```

```
C(z) < B(y) << A(x) > D(w) :
```

```
A(x) :  
  if random() < prob1 :  
    produce B(2*x)  
  else : produce C(0)
```

```
module ?P(pos), ?H(heading),  
       ?U(up), ?L(left)
```

```
B(t) :  
  if t >= MAX_AGE :  
    produce %  
  else : produce B(t+1)
```

```
I(ri) << [ I(r1) ] I(rj)  
          --> I(r1+rj+1)  
I(ri) << I(rj) --> I(rj+1)
```

# L-systems stochastiques

- Insertion d'une production stochastique

```
from random import *
```

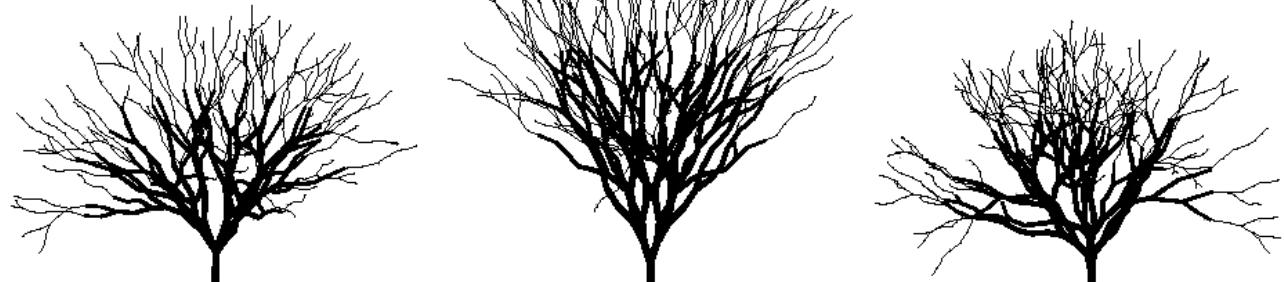
Axiom: A

production:

A :

```
    if random() < 0.5: produce B
```

```
    else: produce C
```

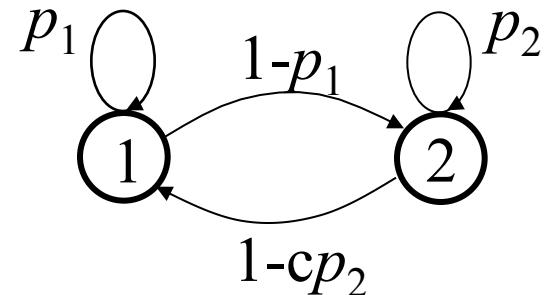


# L-systems stochastiques

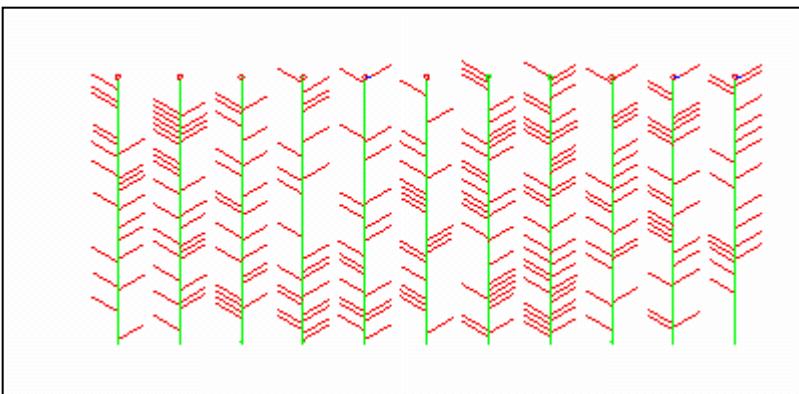
- Simulation d'une chaîne de Markov

Axiom: A

```
A : if random() < p1: produce I[M]A  
else produce IB  
B : if random() < p2: produce IB  
else produce I[M]A
```

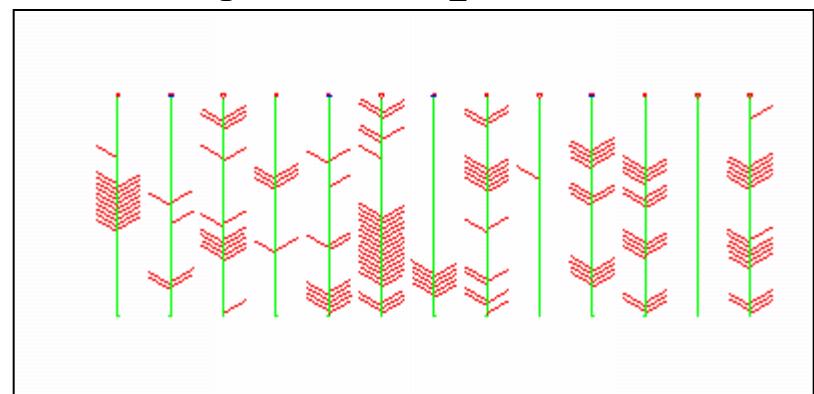


$$p_1=0.25, p_2=0.75$$



$$P(\text{branching}) = 0.25$$

$$p_1=0.85, p_2=0.95$$



$$P(\text{branching}) = 0.25$$

# Architecture racinaire

- Définir les types racinaires (modules)
  - Root, Apex, Segment
  - Crown, CrownApex, CrownSegment
  - ...
- A chaque module, associer des variables d'états
  - order
  - rayon
  - autres? (length, delay, ...)

# Architecture racinaire

- Définir les paramètres d'entrée de votre modèle architectural
  - max\_time: nombre de jours
  - dt : pas de temps
  - number\_crown, delay\_crown
  - radius\_apex
  - growth\_rate
  - crown\_angle
  - branching\_interval