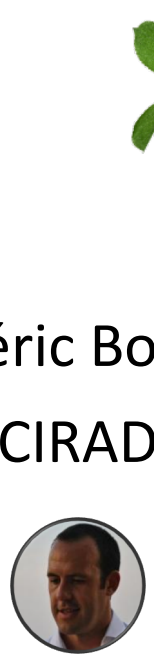
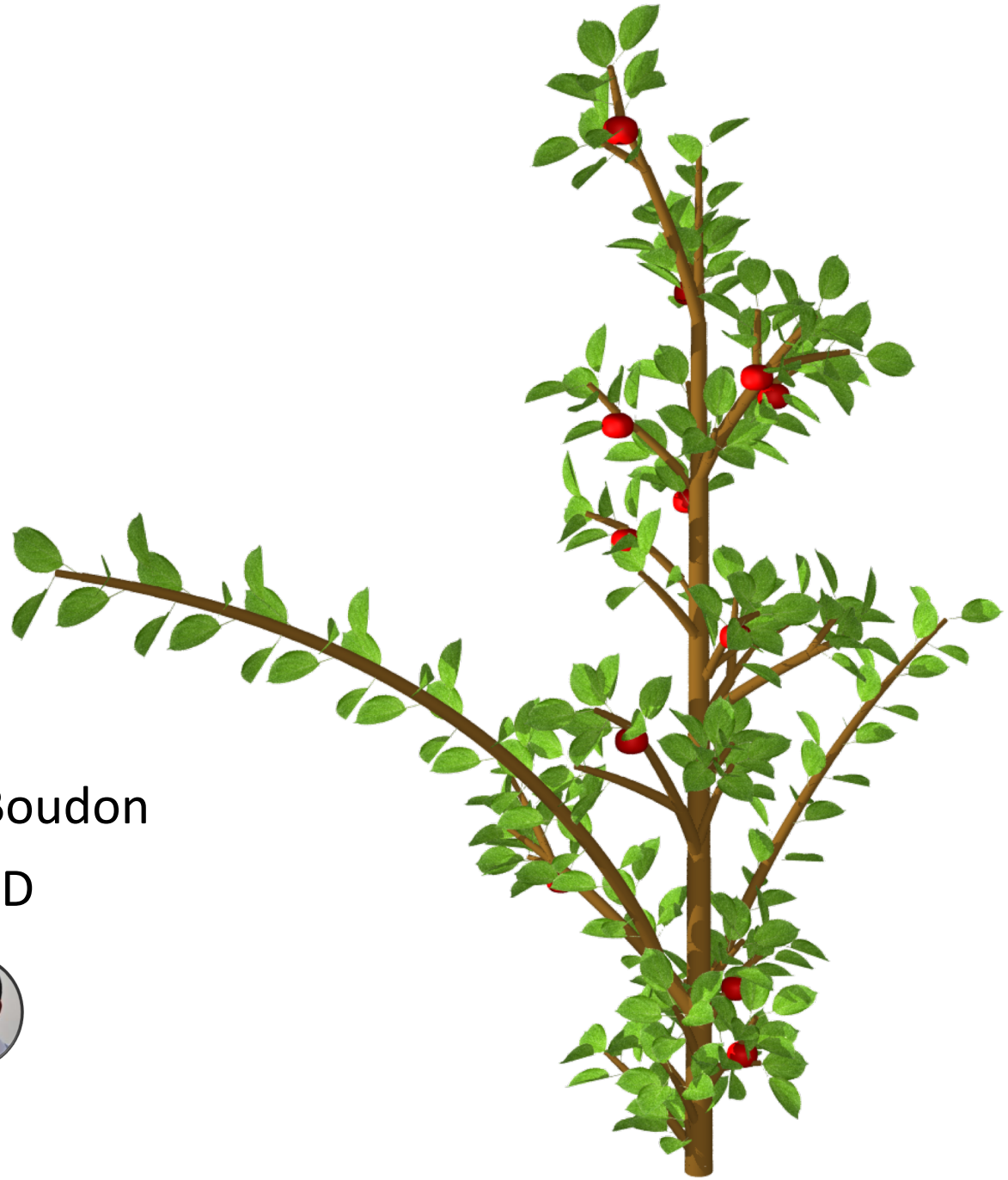
L-systems

Application to plant modelling

Frédéric Boudon

CIRAD





Some tutorials or slides were inspired from lectures of C. Godin and R. Smith

Prerequisite

- Use of the L-Py modelling framework

  - Installation instructions :
    - https://github.com/openalea-training/hbma312_training
    - Training material in folder `lsystem`

  - Launching lpy in the conda console

```
conda activate training
lpy
```

  - Exercises in the notebook: L-systems.ipynb

https://github.com/openalea-training/hbma312_training/blob/master/lsystem/L-systems.ipynb

```
conda activate training
jupyter notebook L-systems.ipynb
```

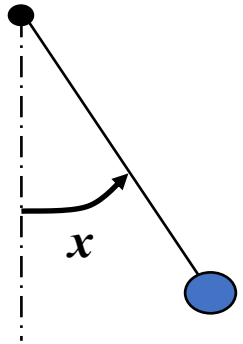
Modeling dynamical systems

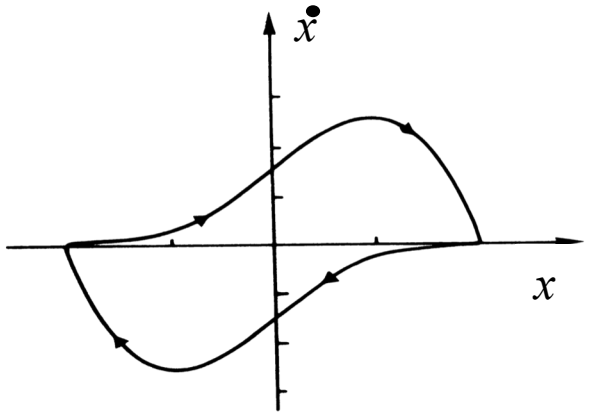
- Classical approach

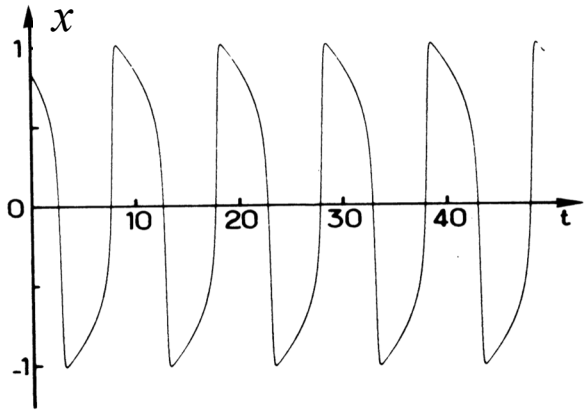
$$f(x, \dot{x}, t) = 0 \qquad x \in \mathbf{R}^n$$

n fixed

Example:
trajectories

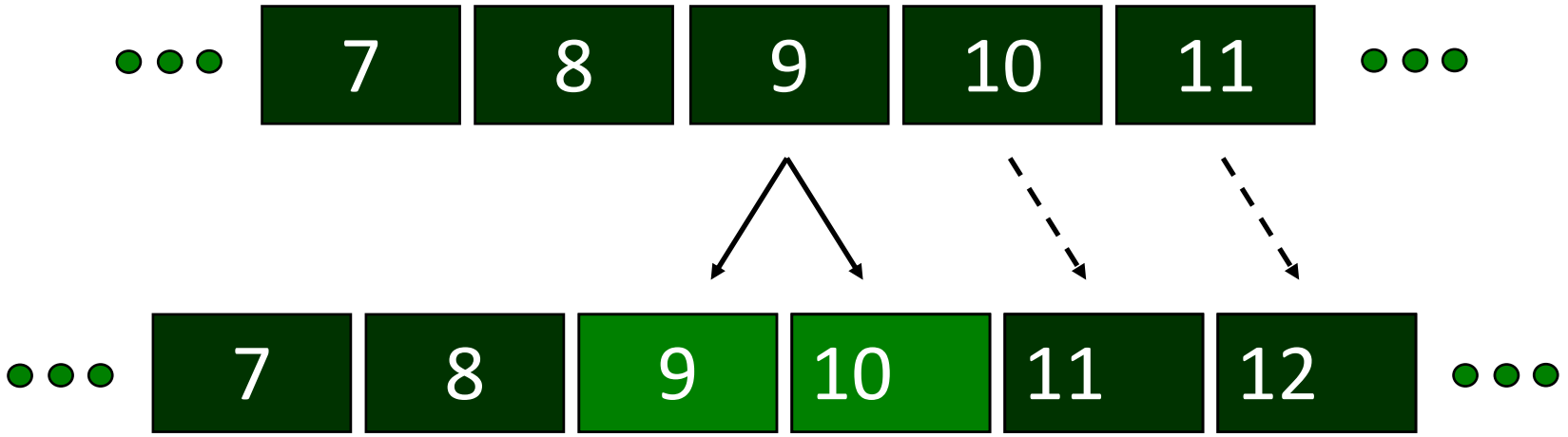






Problem of dynamical systems with dynamical structures

- Number of variables changing over simulation
- Existing formalism are not adapted to manage dynamical structures
- Problem of indexing

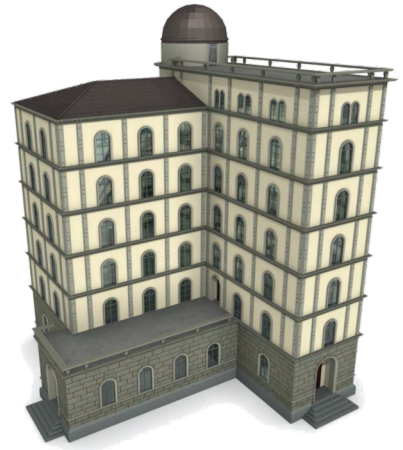


Managing indices quickly becomes a headache

➢ Use of a language to avoid indices

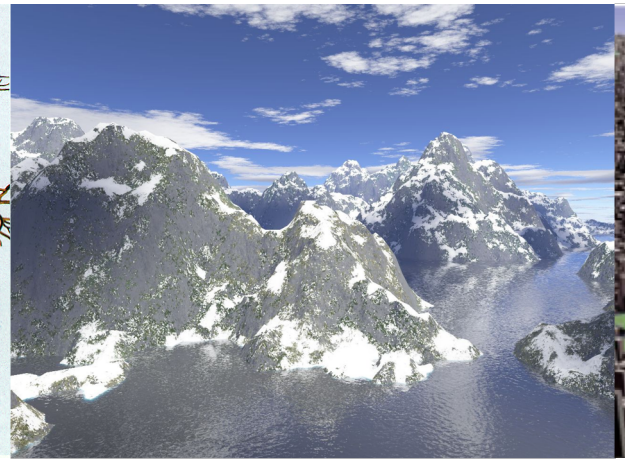
Procedural modelling

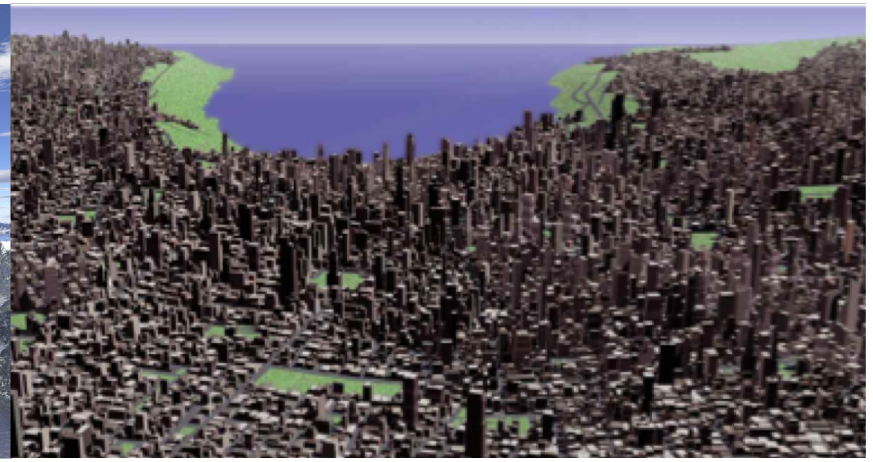
- A set of generative rules to build 3D models or textures.
- Modelling of Textures, Plants, Lands, City, Building



[Muller et al., 2006]

[Oppenheimer, 1986]

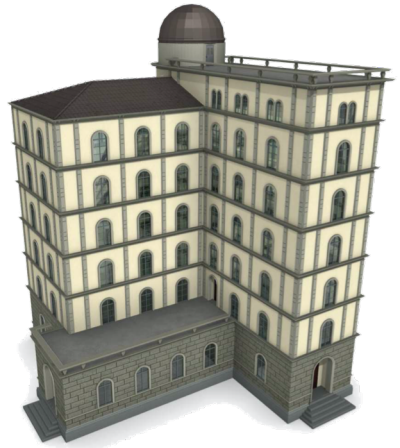




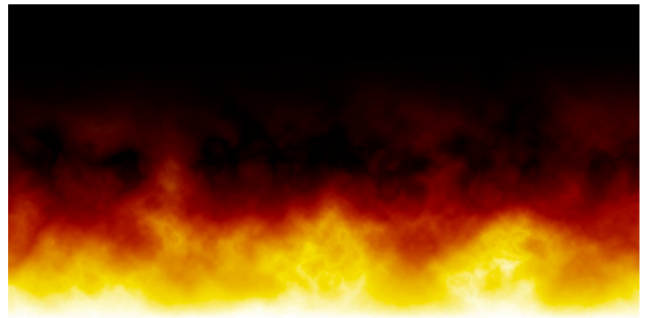
[Parish et al., 2001]

Procedural modelling

- Require development of specific formalism
- A classification [Hendrikx et al., 2011]
  - Pseudo-Random Number Generator (Perlin noise)
  - Generative Grammar (L-systems, Shape Grammar)
  - Image Filtering (Morphology, Convolution, …)
  - Spatial Algorithm (Fractals, …)
  - Artificial Intelligence (Neural Network, Genetic algorithms, …)
  - Simulation of Complex Systems (Cellular Automata, …)



[Muller et al., 2006]



Applications

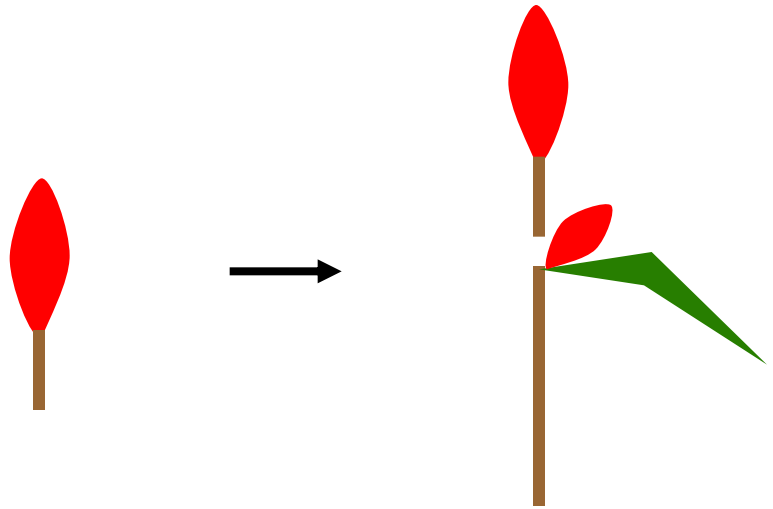
- Scientific modelling : integrative models of growing structures

[Boudon et al., 2020]

L-systems

- Introduced by A. Lindenmayer in 1968
  - Simulation of multi-cellular organisms (Anabaena)
- **Dynamic Systems with Dynamic Structures** (Giavitto, 01)
- Well adapted for modeling **plant growth**. Widely used for FSPM.
- Important contribution from computer graphics (Prusinkiewicz et al.).
- Different implementations : VLab (Prusinkiewicz, Lindenmayer, 90), GroImp/XL (Kurth, 2008), L-Py (Boudon et al., 2008)

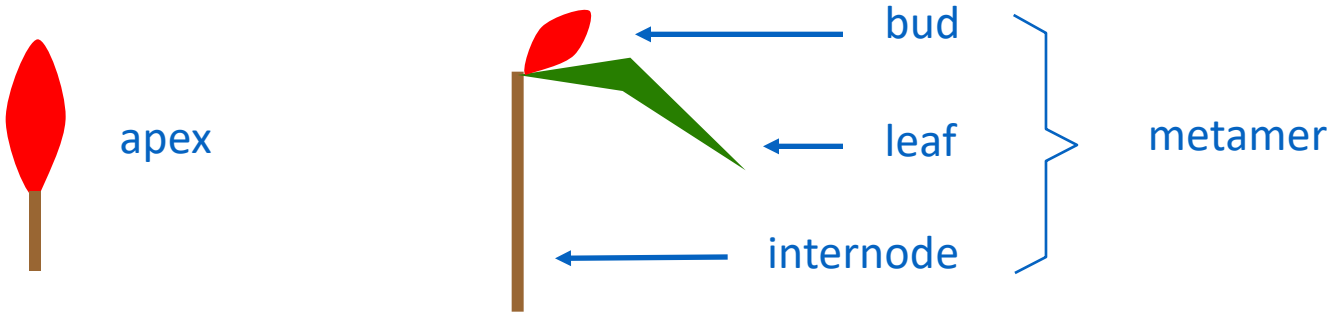
Development as a rewritting process



production

Development can be thought of as a re-writing process
Apex is replaced by an internode, leaf, bud, and another apex

Development as a rewritting process



L-System

Alphabet: (A, M)

Axiom: A

Production: A → MA

Development as a rewritting process

L-System

Alphabet: (A, M)

Axiom: A

Production: A → MA

A

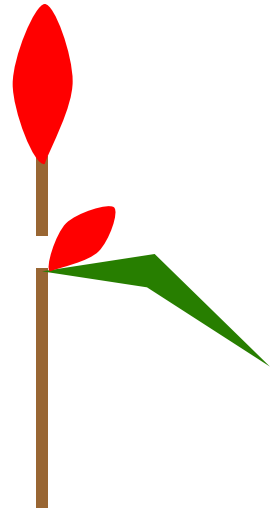
Development as a rewritting process

L-System

Alphabet: (A, M)

Axiom: A

Production: A → MA



MA

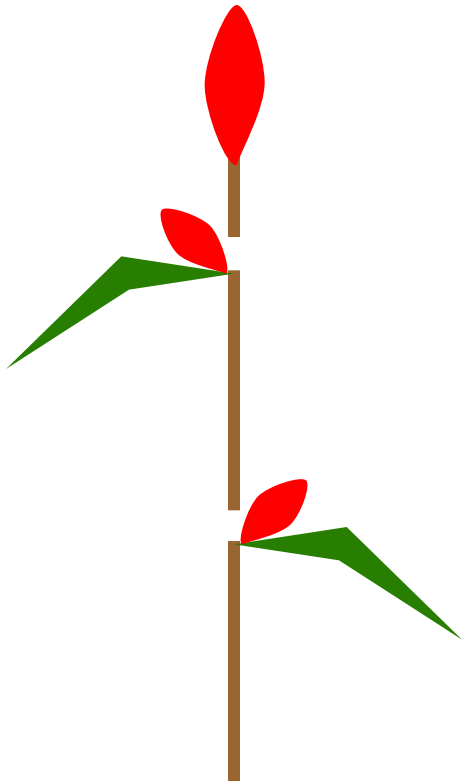
Development as a rewritting process

L-System

Alphabet: (A, M)

Axiom: A

Production: A → MA



MMA

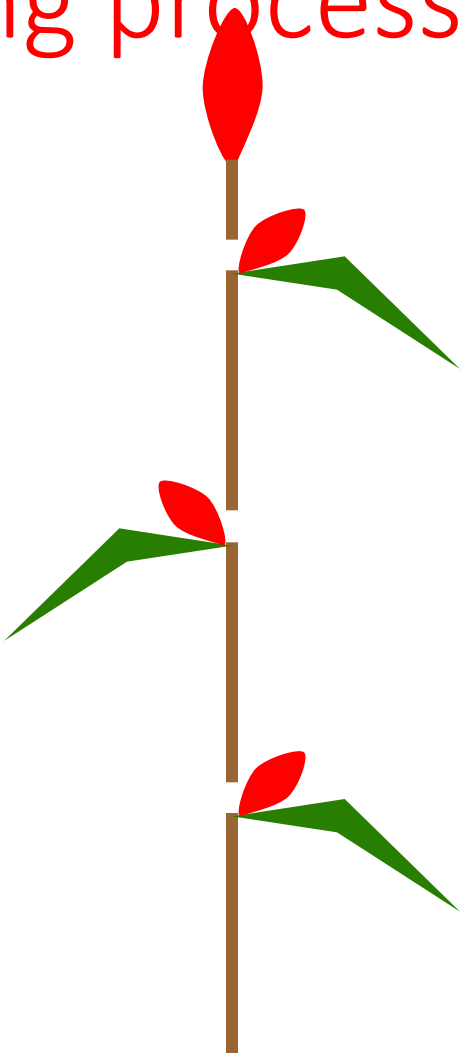
Development as a rewritting process

L-System

Alphabet: (A, M)

Axiom: A

Production: A → MA



MMMA

HFS 52.00

L-systems

- L-systems consist of an alphabet V, an axiom w and a set of productions P.

$$G = < V, w, P>$$

  - **Alphabet:** Finite set of symbols

$$V = \{A,E,+, \#, m\}$$

  - **String**: any word made of letters from *V*

AmEAA#A++#mmmm

  - **Production rule**: rule that specifies how to replace a symbol by a string of symbols in a given string

⟶ A#AE

- **L-System** : grammar for which rules are applied in parallel to all the symbols of an input string (axiom)

Example : AmEAA#A++#mmmm => AmEAAA#AEA++A#AEmmmm

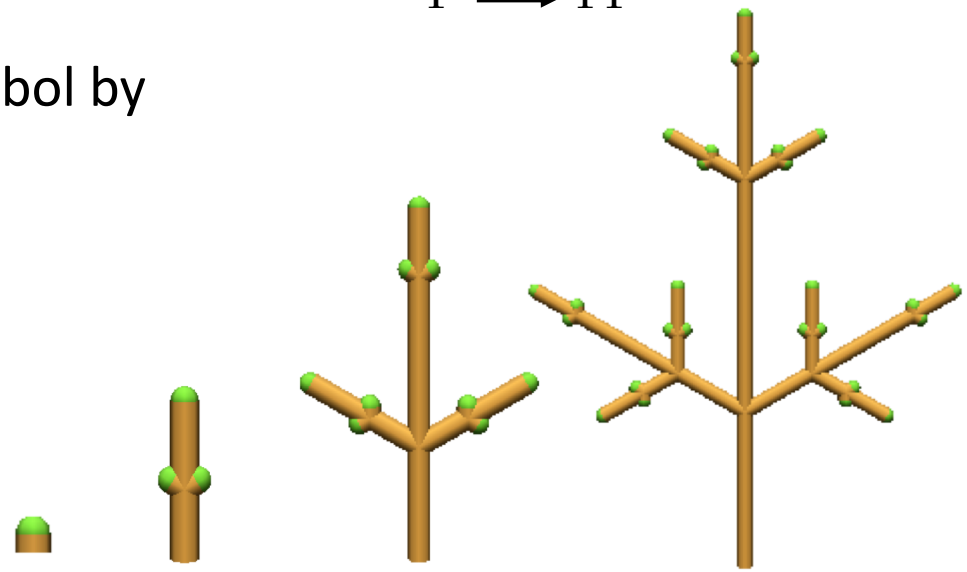
Exemple :

$V = \{F,A,+,-\}$

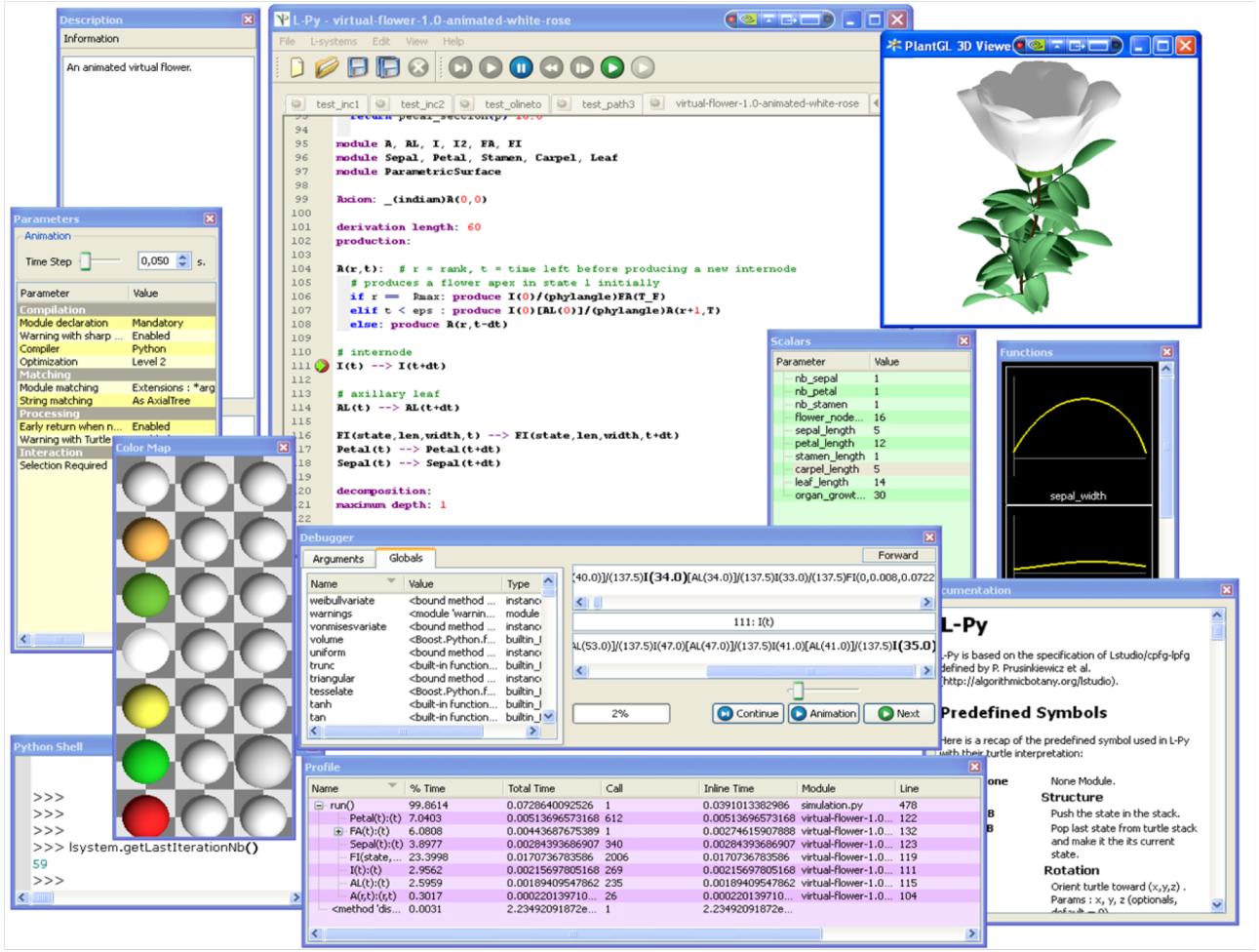
$w = A$

P: A ⟶ F[+A][-A]FA

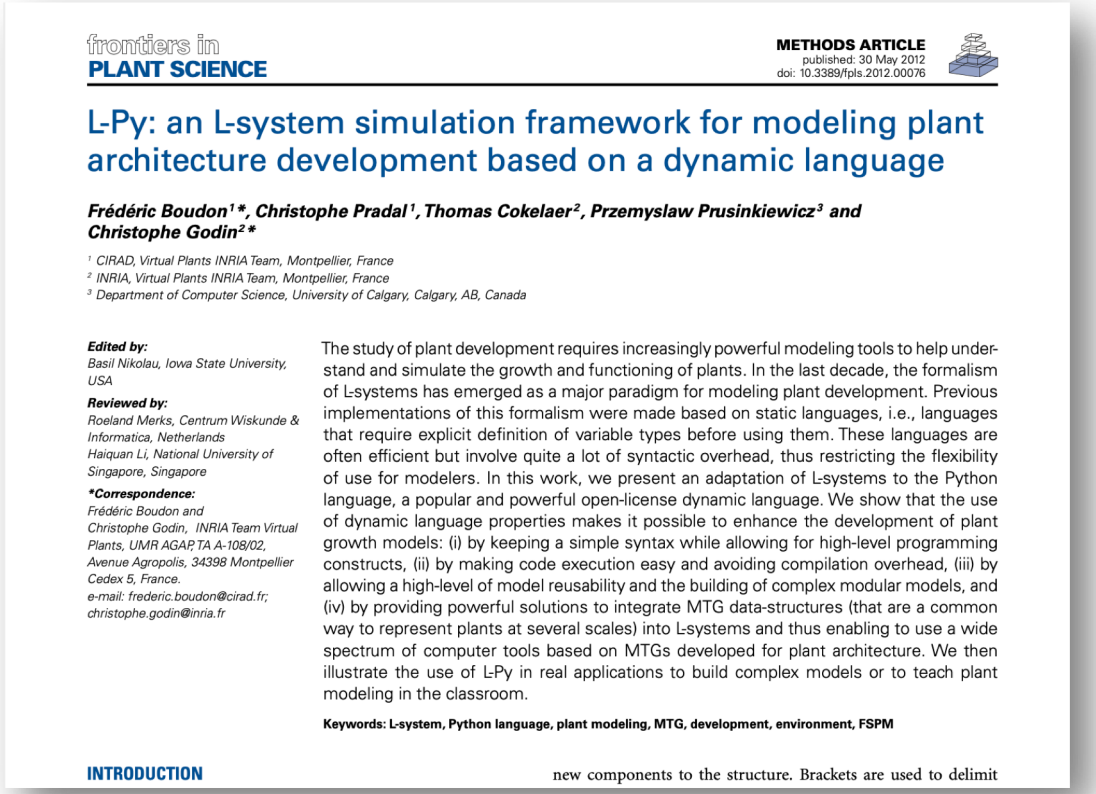
F ⟶ FF



L-Py



https://github.com/fredboudon/lpy



frontiers in
PLANT SCIENCE

METHODS ARTICLE
published: 30 May 2012
doi: 10.3389/fpls.2012.00076

L-Py: an L-system simulation framework for modeling plant architecture development based on a dynamic language

Frédéric Boudon[1]*, Christophe Pradal[1], Thomas Cokelaer[2], Przemyslaw Prusinkiewicz[3] and Christophe Godin[2]*

[1] *CIRAD, Virtual Plants INRIA Team, Montpellier, France*
[2] *INRIA, Virtual Plants INRIA Team, Montpellier, France*
[3] *Department of Computer Science, University of Calgary, Calgary, AB, Canada*

Edited by:
Basil Nikolau, Iowa State University, USA

Reviewed by:
Roeland Merks, Centrum Wiskunde & Informatica, Netherlands
Haiquan Li, National University of Singapore, Singapore

****Correspondence:***
Frédéric Boudon and Christophe Godin, INRIA Team Virtual Plants, UMR AGAP,TA A-108/02, Avenue Agropolis, 34398 Montpellier Cedex 5, France.
e-mail: frederic.boudon@cirad.fr; christophe.godin@inria.fr

The study of plant development requires increasingly powerful modeling tools to help understand and simulate the growth and functioning of plants. In the last decade, the formalism of L-systems has emerged as a major paradigm for modeling plant development. Previous implementations of this formalism were made based on static languages, i.e., languages that require explicit definition of variable types before using them. These languages are often efficient but involve quite a lot of syntactic overhead, thus restricting the flexibility of use for modelers. In this work, we present an adaptation of L-systems to the Python language, a popular and powerful open-license dynamic language. We show that the use of dynamic language properties makes it possible to enhance the development of plant growth models: (i) by keeping a simple syntax while allowing for high-level programming constructs, (ii) by making code execution easy and avoiding compilation overhead, (iii) by allowing a high-level of model reusability and the building of complex modular models, and (iv) by providing powerful solutions to integrate MTG data-structures (that are a common way to represent plants at several scales) into L-systems and thus enabling to use a wide spectrum of computer tools based on MTGs developed for plant architecture. We then illustrate the use of L-Py in real applications to build complex models or to teach plant modeling in the classroom.

Keywords: L-system, Python language, plant modeling, MTG, development, environment, FSPM

INTRODUCTION

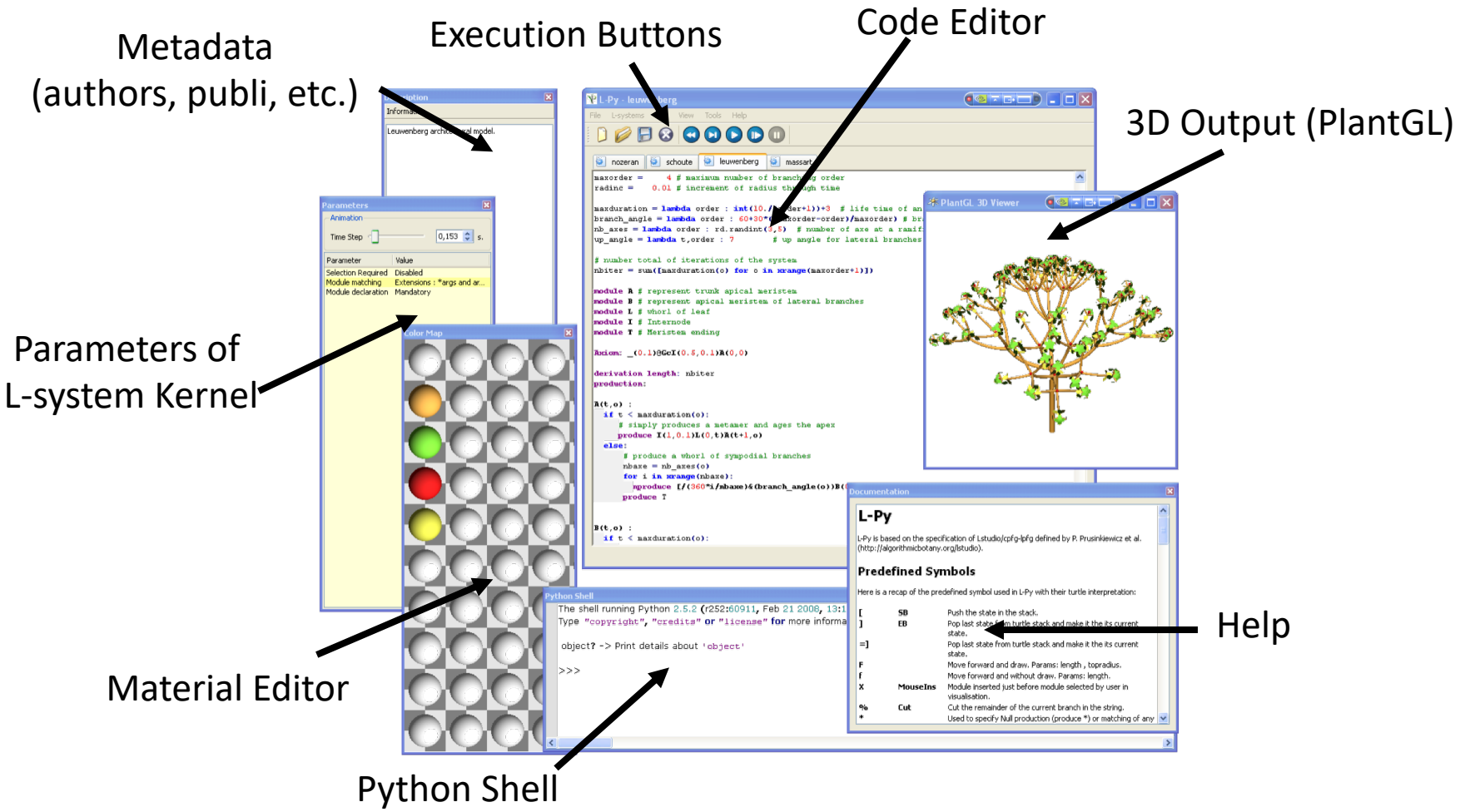
new components to the structure. Brackets are used to delimit

L-Py Syntax

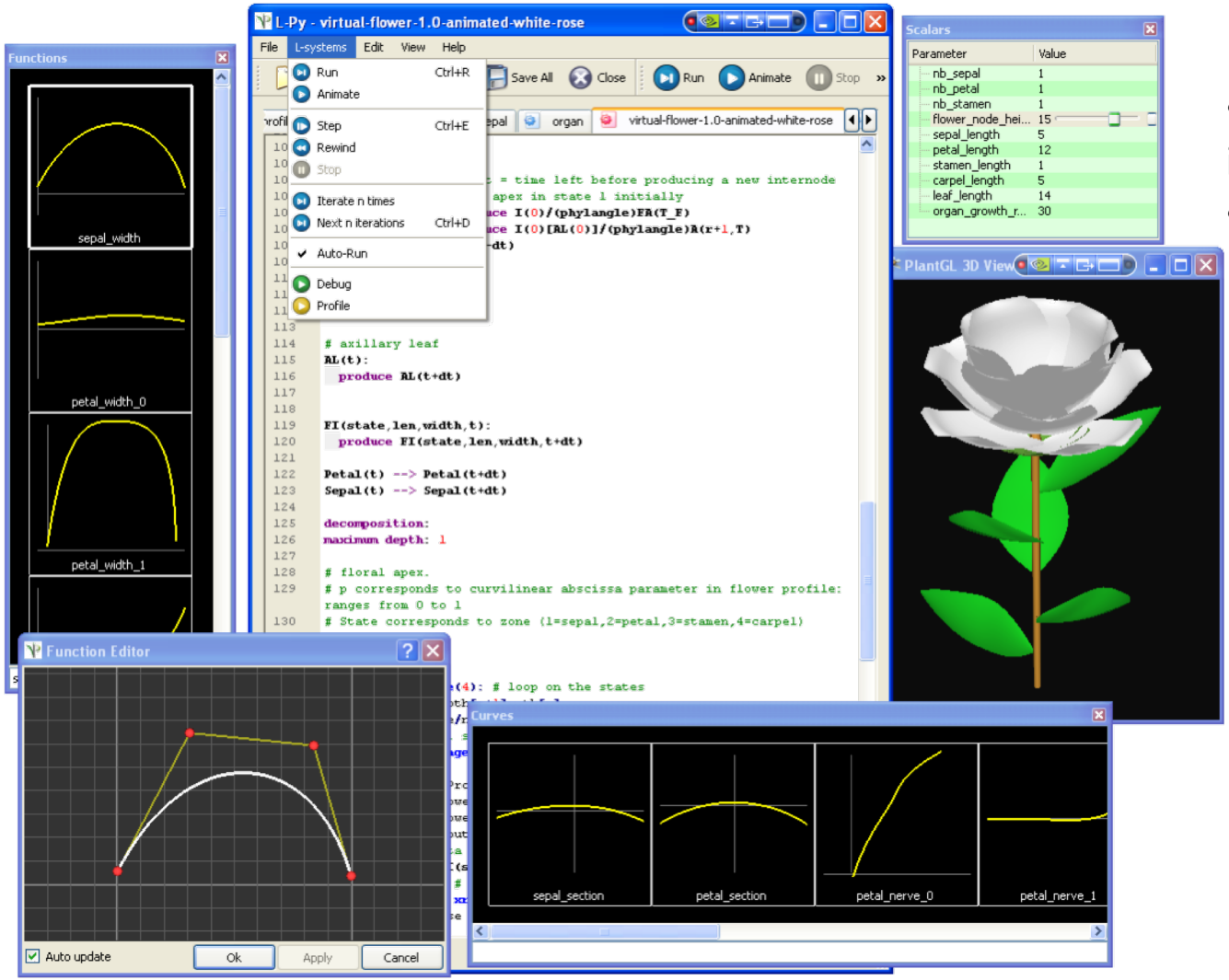
- Combining L-systems and python

```
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
```

Graphical editors



Scalars, Functions and Curves



- Direct use of objects into the code
- Continuous modeling

The notebook environment

Cell of lsystem code

```
%%lpy -w size -a True
# lsystem code
Axiom: A
```

render nbviewer

launch binder

jupyter L-systems Last Checkpoint: 11/28/2019 (unsaved changes)

Se déconnecter

Fichier Édition Affichage Insérer Cellule Noyau Widgets Aide

Trusted | Python 3

Run Code

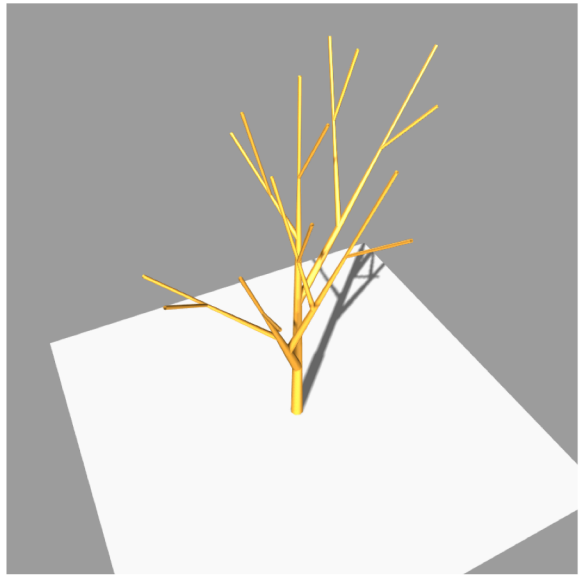
L-systems

In [2]:
```
from pgljupyter import *
```

L-Py syntax

Here comes a simple example of L-systems

In [10]:
```
%%lpy -w 10 -a True
from random import random
MAX_AGE, dr = 10, 0.03 # constants
module Apex(age), Internode(length,radius)
Axiom: @Gc Apex(0)
derivation length: 5
production:
Apex(age) :
  if age < MAX_AGE:
    produce Internode(1+ random(),0.03) /(137)[+(30)Apex(age+1)]Apex(age+1)

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



First draft of graphical parameter edition

Fundamental concepts

- Axial tree; string of characters

Structure

- 3D Representation; LOGO Turtle

Geometry

- Rewriting rules

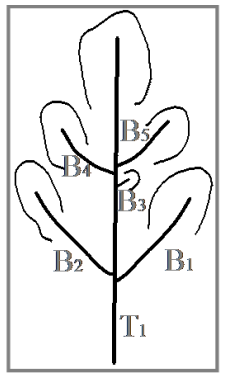
Dynamic

Tree structure representation

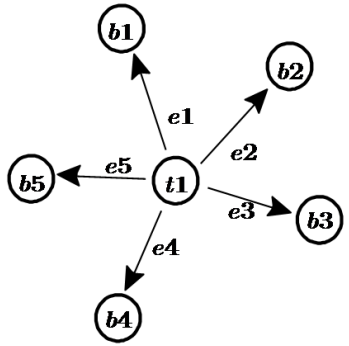
Describing trees with strings

Axial tree

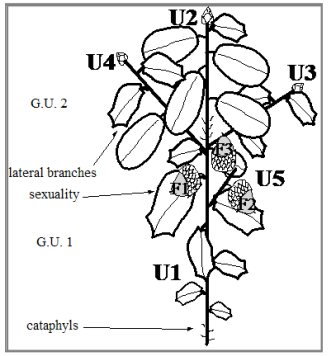
- Representation of plant structure



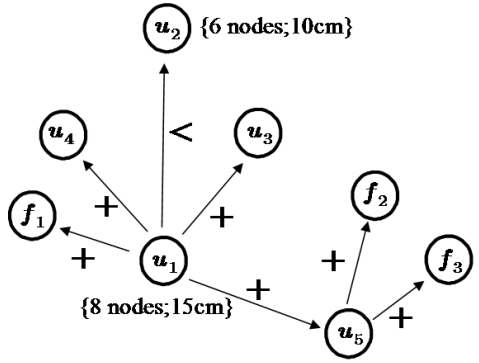
a



b



c



d

Axial tree

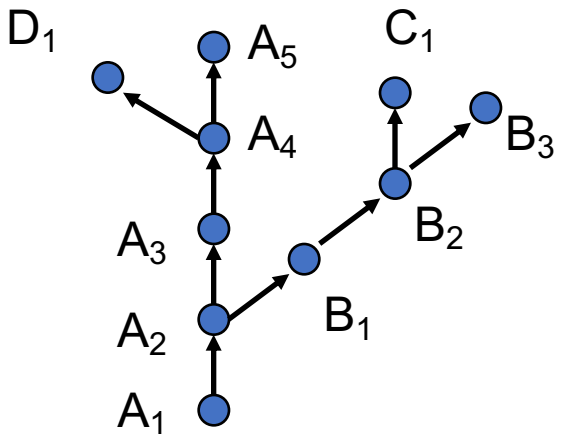
- Bracketed string notation

  Alphabet $V \qquad V_E = V \cup \{[,]\}$

  String $m = x_1[\alpha_1]x_2[\alpha_2]...x_n[\alpha_n]x_{n+1}$

  $x_1, x_2, ..., x_{n+1} \in V^* \qquad \alpha_1, \alpha_2, ..., \alpha_n \in V_E^*$

- Example



$A_1A_2[B_1B_2[C_1]B_3]A_3A_4[D_1]A_5$

Parameters

- Use of parameters to characterise modules of plants
  - Parametric letters (or modules)
    - ➢ Parametric strings

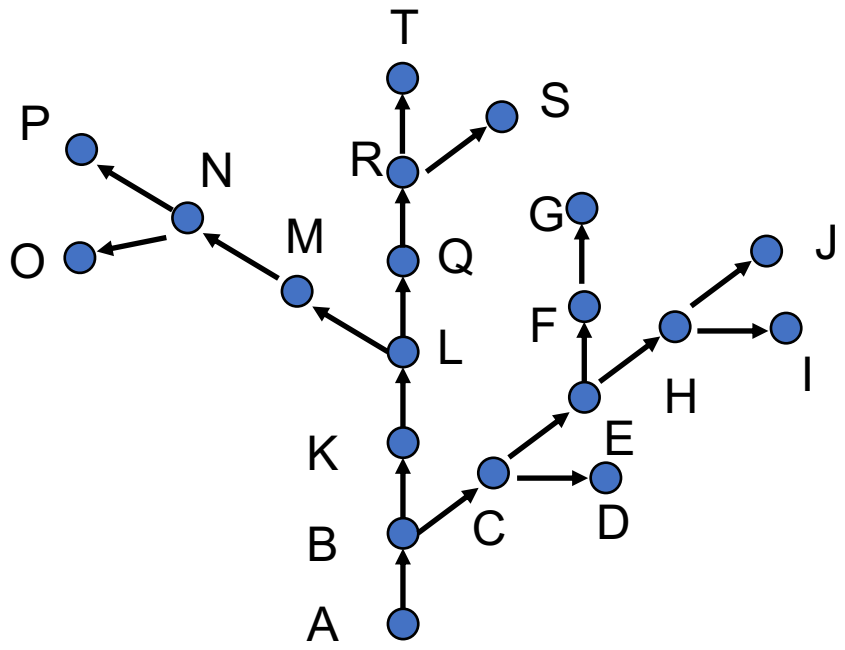
- Example:

```
I(5,6)I(4,5.8)I(3,5.8)I(1,3)B('F')
```

- Useful for geometric interpretation

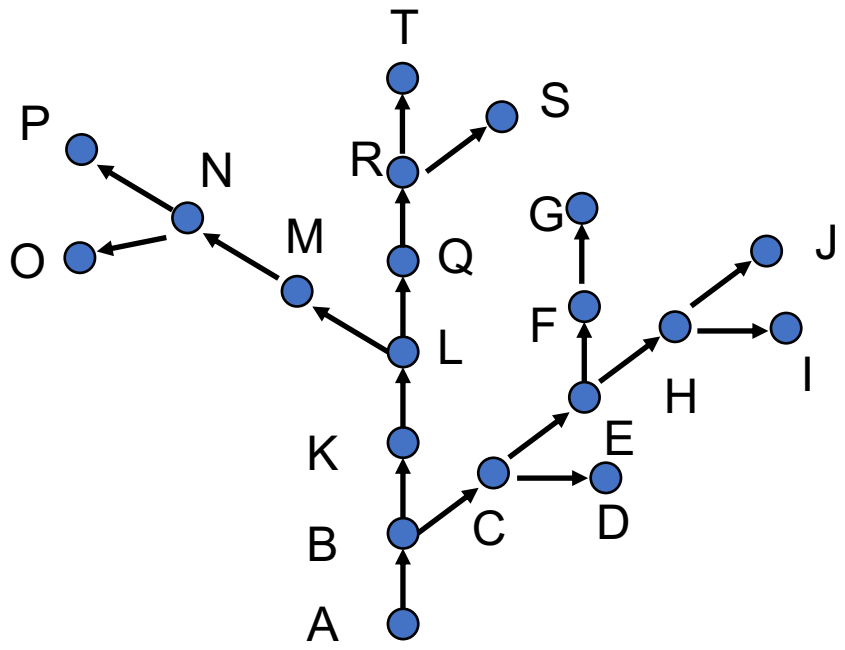
Exercice

- Give the string representing the following structure



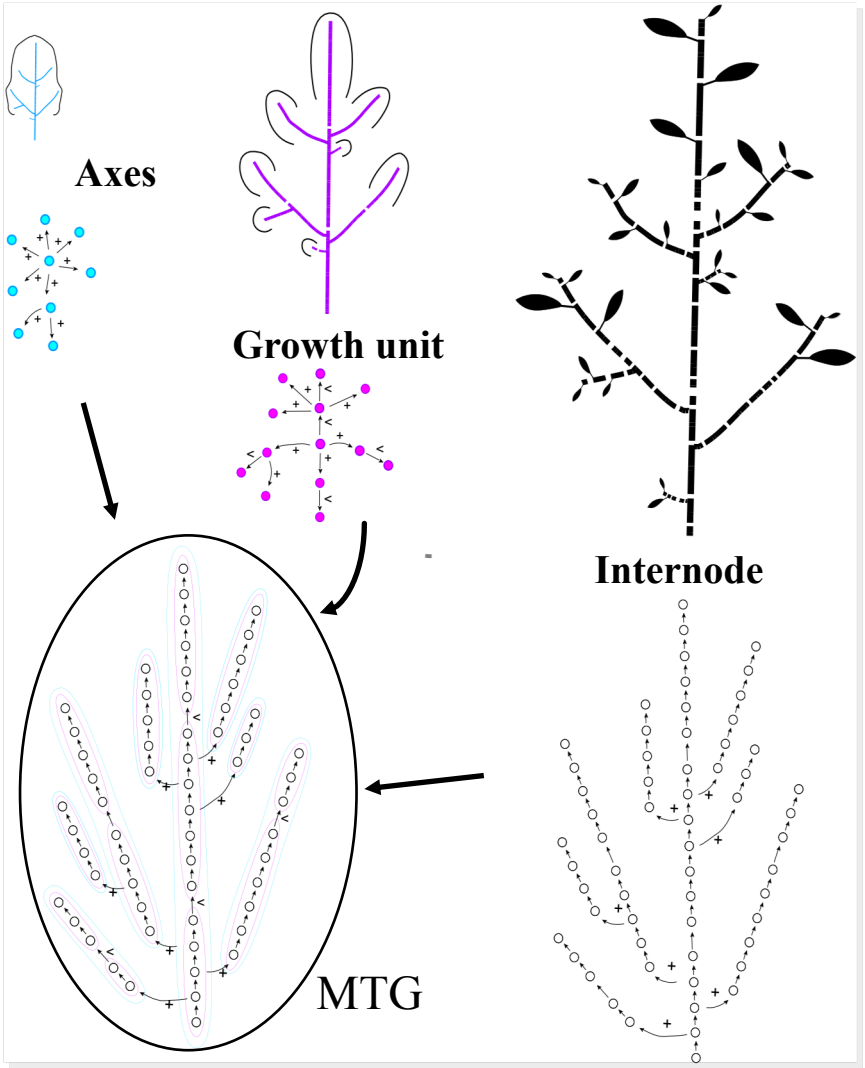
Exercice

- Give the string representing the following structure

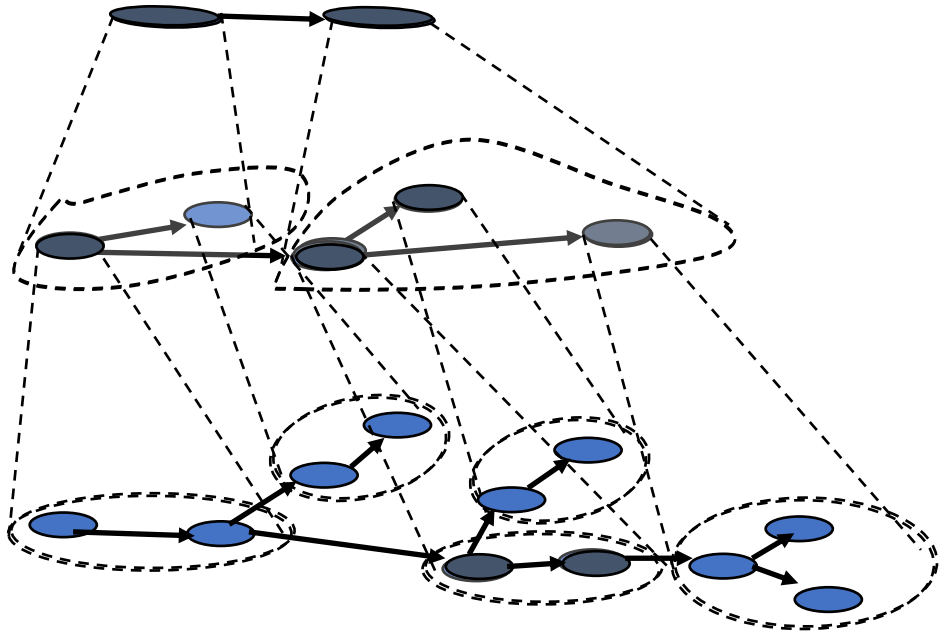


AB[C[D]E[FG]H[I]J]KL[MN[O]P]QR[S]T

Multiscale topological representation of plants

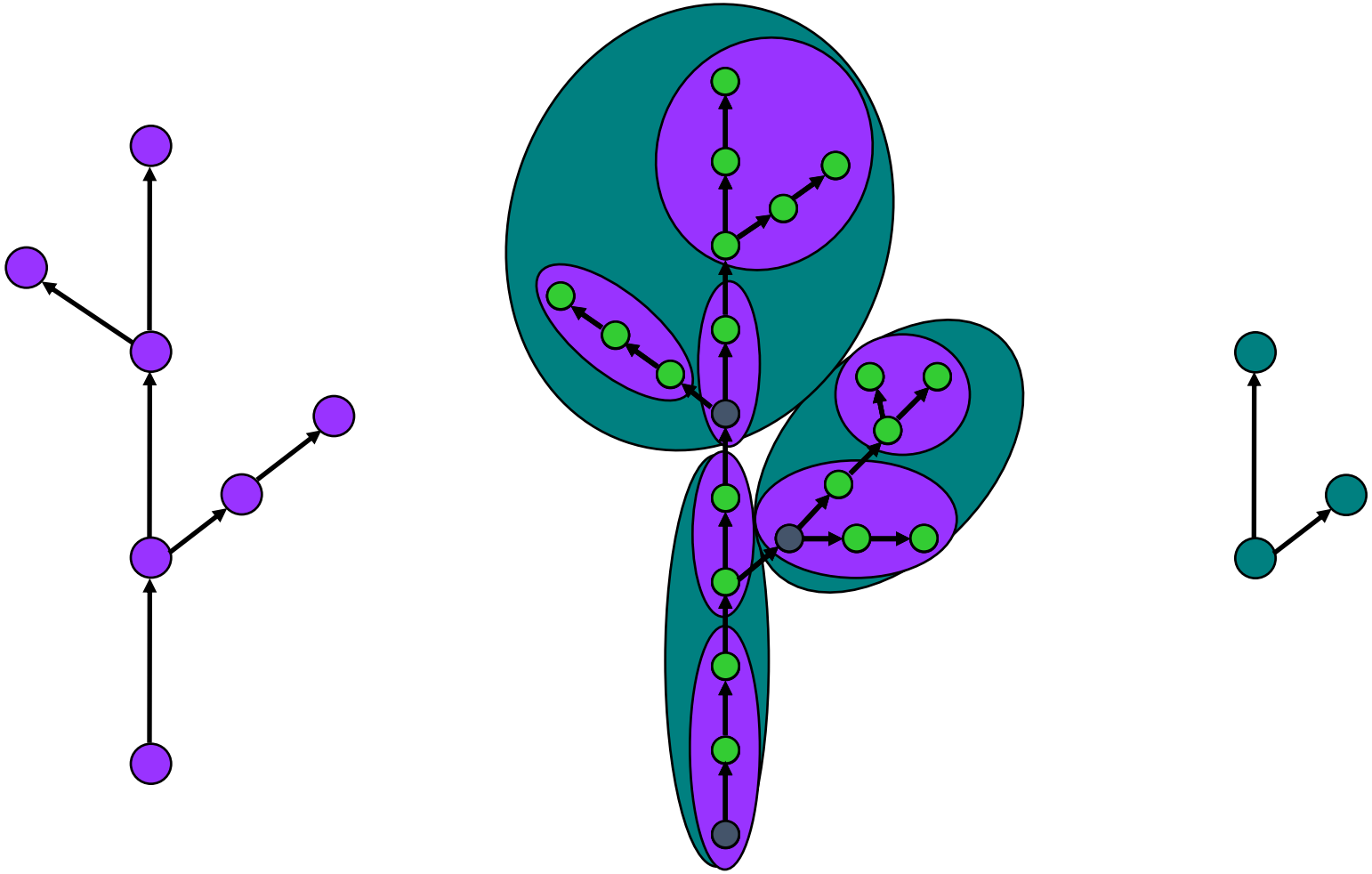


Multiscale Tree Graph (MTG)



(Godin, Caraglio 98)

Specification of macroscopic scale



N N N N [N [N N] N N [N] N] N N [N N N] N N [N N] N N

B N N N **B** N [**B** N [N N] N **B** N [N] N] N **B** N [**B** N N N] N **B** N [N N] N N

S **B** N N N **B** N [**S** **B** N [N N] N **B** N [N] N] N **S** **B** N [**B** N N N] N **B** N [N N] N N

Multi-scale extension

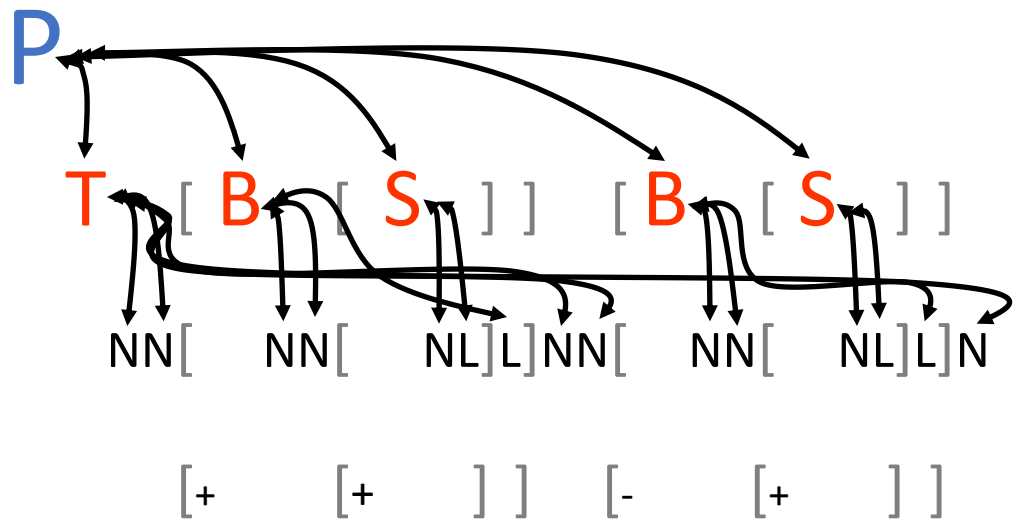
- Extend modules with scale tags

module P : scale 1
module T,B, S : scale 2
module N,L : scale 3

PTNN[+BNN[+SNL]L]NN[-BNN[+SNL]L]N

PTNN[+BNN[+SNL]L]NN[-BNN[+SNL]L]N

- Add some between scale relations



Multi-scale extension

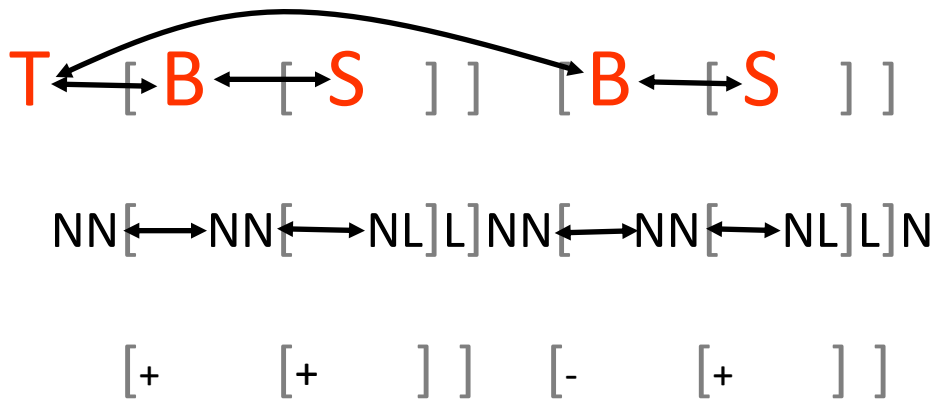
- Extend modules with scale tags

module P : scale 1
module T,B, S : scale 2
module N,L : scale 3

PTNN[+BNN[+SNL]L]NN[-BNN[+SNL]L]N

PTNN[+BNN[+SNL]L]NN[-BNN[+SNL]L]N

- Add some within scale relations



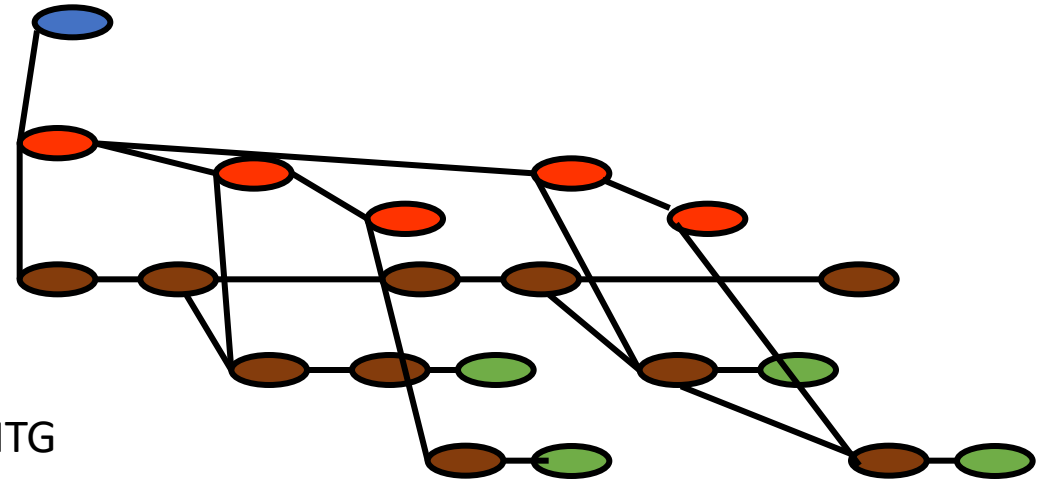
Multi-scale extension

- Extend modules with scale tags

module P : scale 1
module T,B, S : scale 2
module N,L : scale 3

PTNN[+BNN[+SNL]L]NN[-BNN[+SNL]L]N

PTNN[+BNN[+SNL]L]NN[-BNN[+SNL]L]N



MTG

Exercise

- Let the following multiscale structure

module S : scale = 1
module B : scale = 2
module N : scale = 3

$S_1B_1N_0N_1\,[S_2B_2N_2\,[N_3]\,B_3\,N_4\,\,[N_5N_6]\,N_7[N_8]N_9]N_{10}B_4N_{11}\,[S_3B_5N_{12}N_{13}[N_{14}]N_{15}]N_{16}N_{17}\,[N_{18}]N_{19}$

- Draw a geometrical interpretation

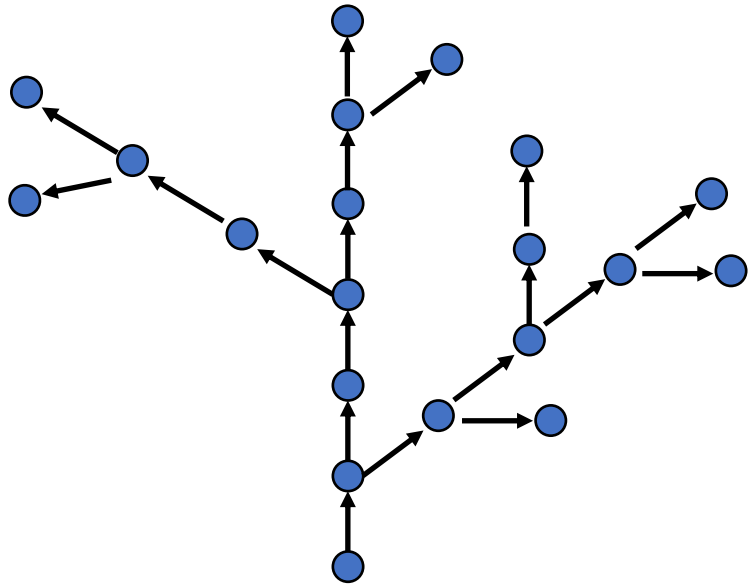
Exercise

- Let the following multiscale structure

module S : scale = 1
module B : scale = 2
module N : scale = 3

$S_1B_1N_0N_1\,[S_2B_2N_2\,[N_3]\,B_3\,N_4\,\,[N_5N_6]\,N_7[N_8]N_9]N_{10}B_4N_{11}\,[S_3B_5N_{12}N_{13}[N_{14}]N_{15}]N_{16}N_{17}\,[N_{18}]N_{19}$

- Draw a geometrical interpretation



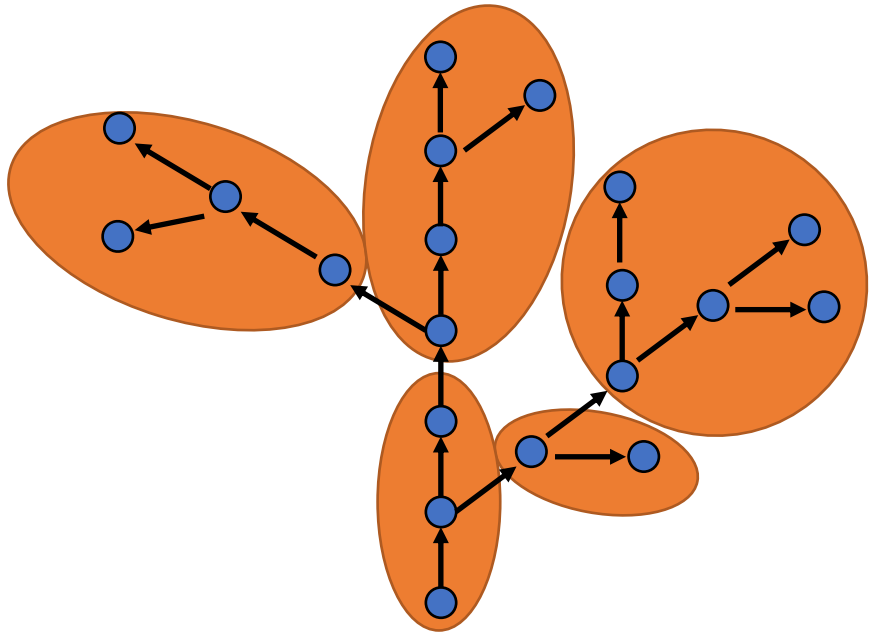
Exercise

- Let the following multiscale structure

module S : scale = 1
module B : scale = 2
module N : scale = 3

$S_1B_1N_0N_1\,[S_2B_2N_2\,[N_3]\,B_3\,N_4\,\,[N_5N_6]\,N_7[N_8]N_9]N_{10}B_4N_{11}\,[S_3B_5N_{12}N_{13}[N_{14}]N_{15}]N_{16}N_{17}\,[N_{18}]N_{19}$

- Draw a geometrical interpretation



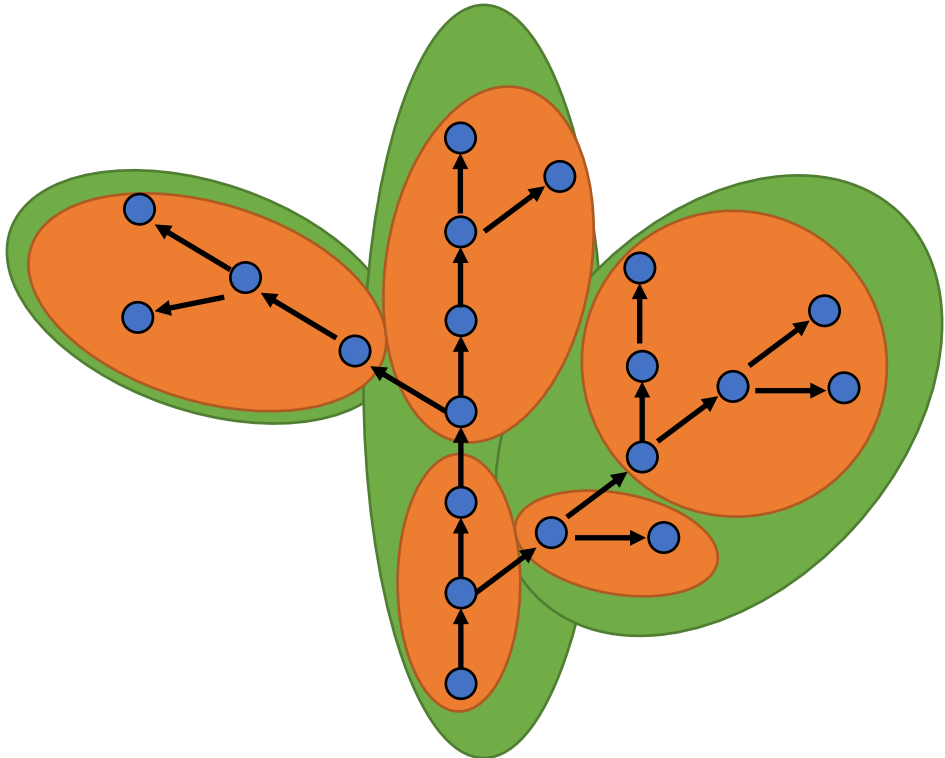
Exercise

- Let the following multiscale structure

module S : scale = 1
module B : scale = 2
module N : scale = 3

$S_1B_1N_0N_1\,[S_2B_2N_2\,[N_3]\,B_3\,N_4\,\,[N_5N_6]\,N_7[N_8]N_9]N_{10}B_4N_{11}\,[S_3B_5N_{12}N_{13}[N_{14}]N_{15}]N_{16}N_{17}\,[N_{18}]N_{19}$

- Draw a geometrical interpretation



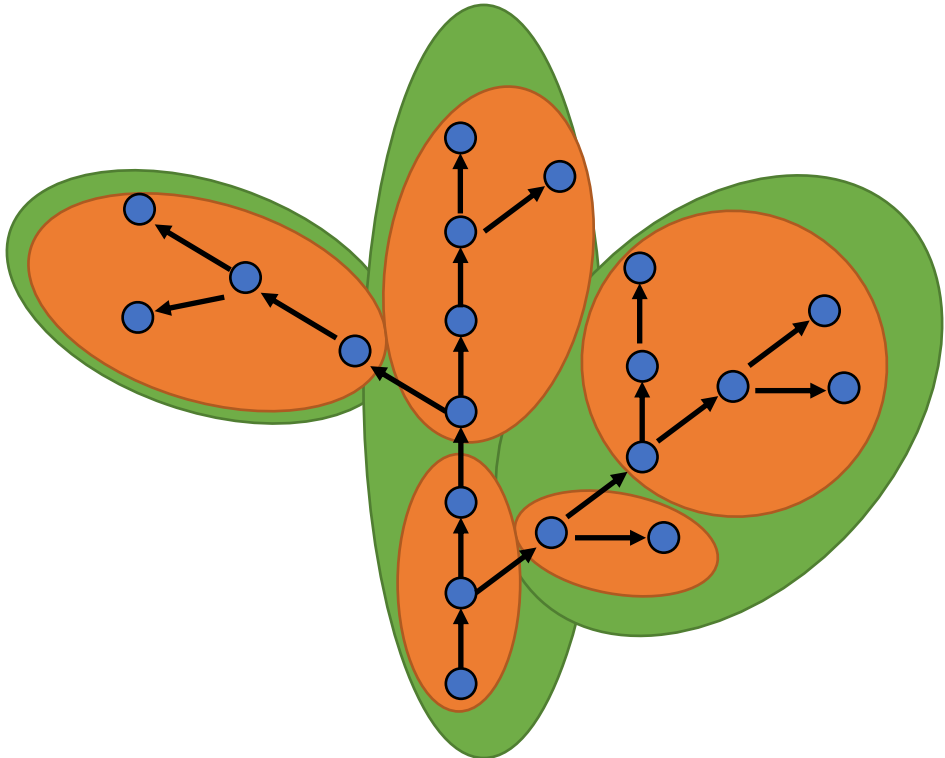
Exercise

- Let the following multiscale structure

module S : scale = 1
module B : scale = 2
module N : scale = 3

$S_1B_1N_0N_1\,[S_2B_2N_2\,[N_3]\,B_3\,N_4\,\,[N_5N_6]\,N_7[N_8]N_9]N_{10}B_4N_{11}\,[S_3B_5N_{12}N_{13}[N_{14}]N_{15}]N_{16}N_{17}\,[N_{18}]N_{19}$

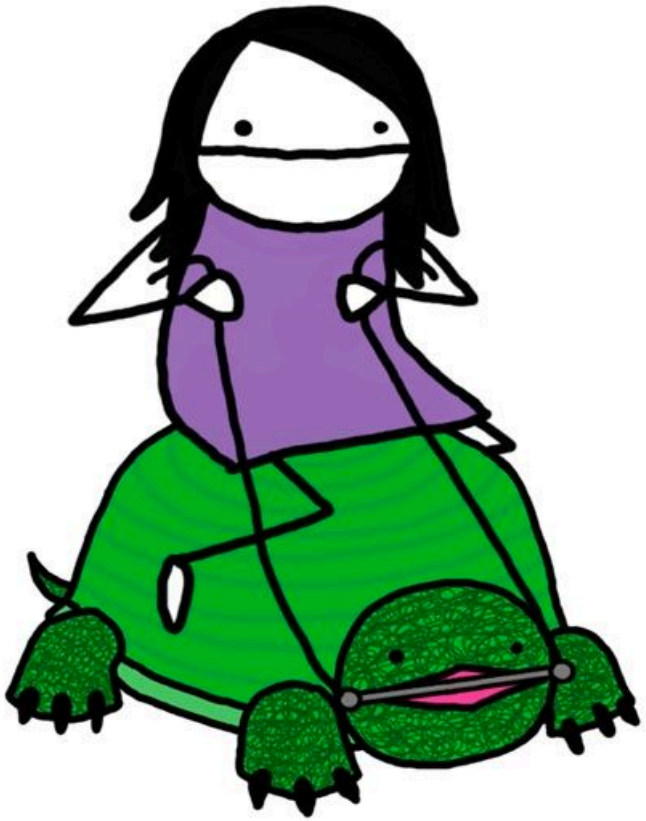
- Draw a geometrical interpretation
- Complexes at scale 2 of N_3 ? B_2
- Complexes at scale 1 of B_3 ? S_2
- Components at scale 3 of B_3 ? N_4 N_5N_6 N_7 N_8 N_9 N_{10}
- Parents at scale 2 of B_3? B_2
- Parents at scale 3 of N_{12}? N_{11}
- What are the children at scale 3 of N_4 ? N_5 and N_7



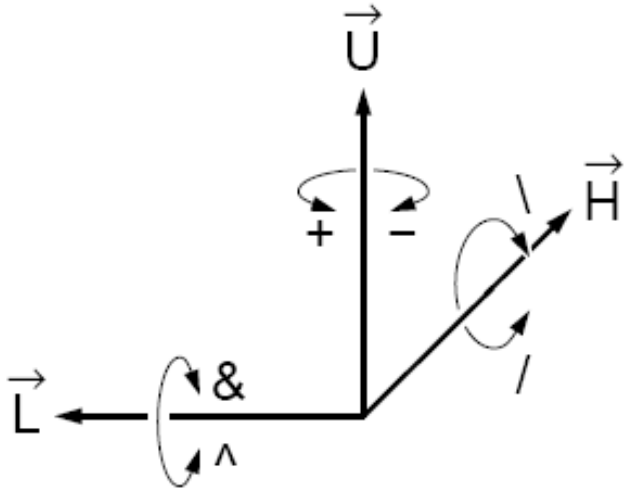
Geometrical Interpretation

Riding the turtle

Geometric Interpretation

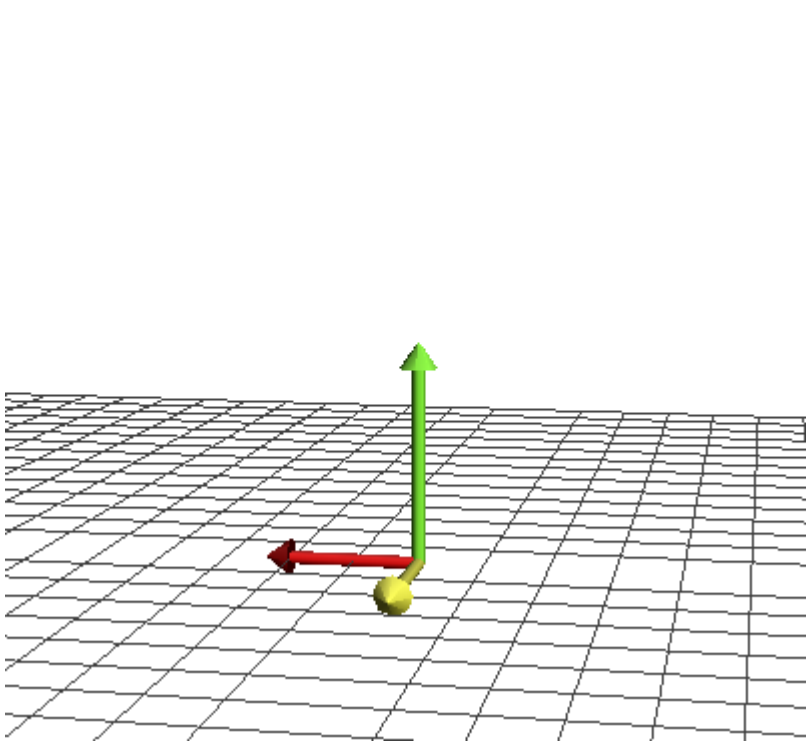


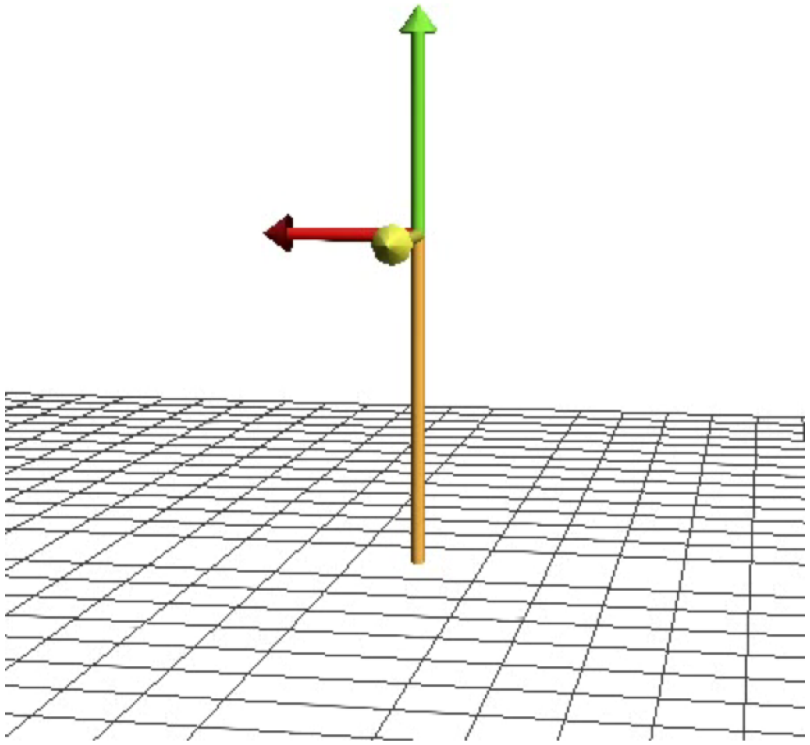
- LOGO Turtle :



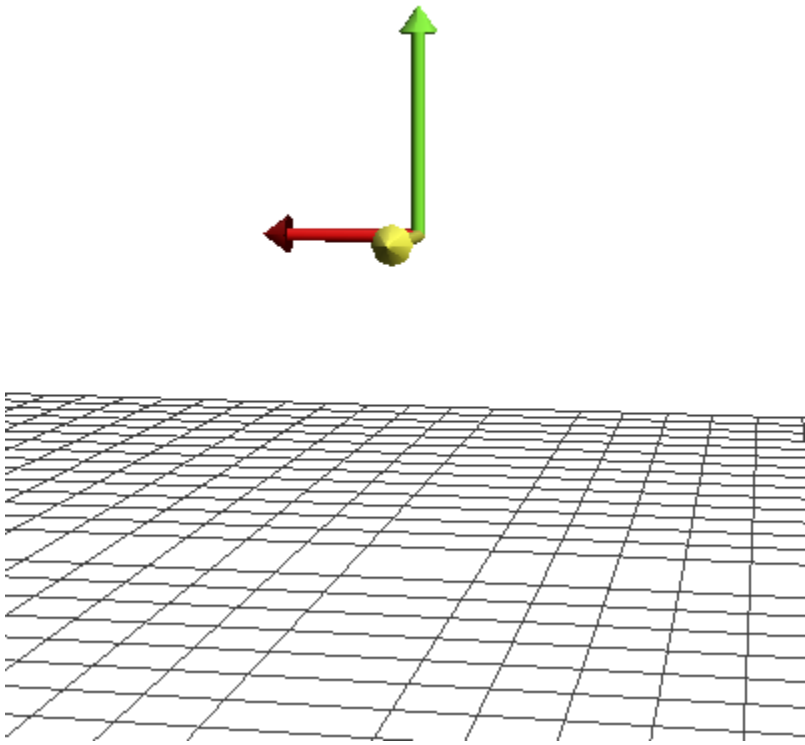
- Symbols interpreted as commands for the turtle

Geometric Interpretation



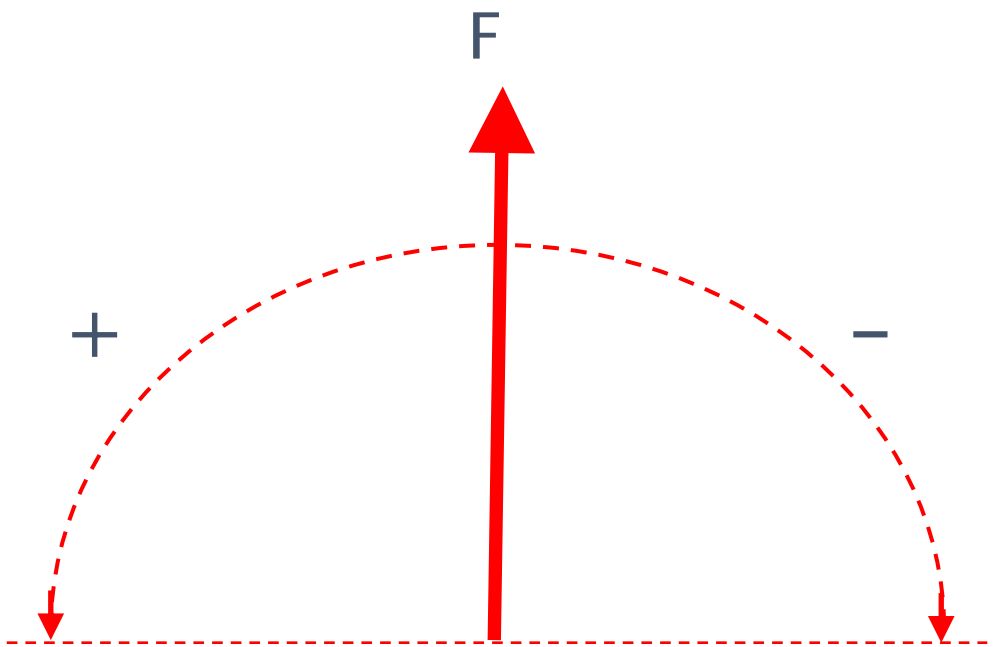


F(x)



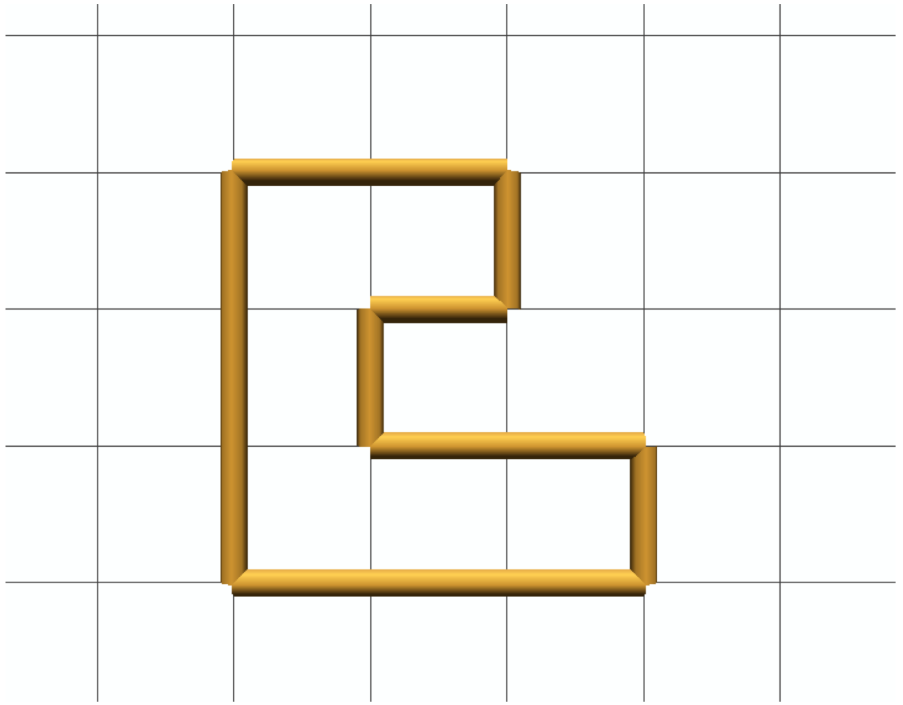
f(x)

Geometric Interpretation



Logo style turtle interpretation

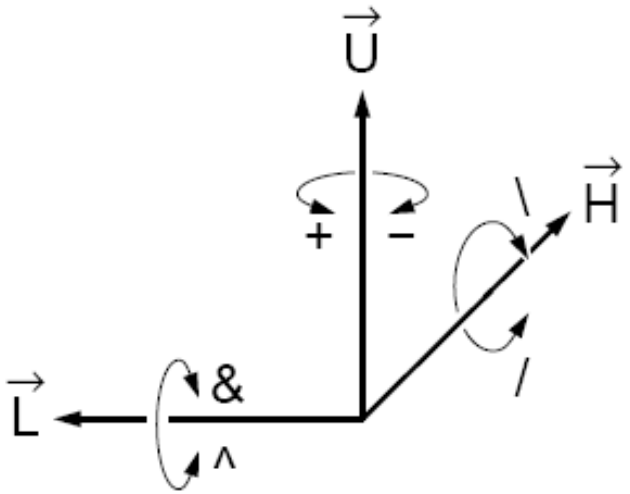
F Move forward, drawing a line
\+ Turn left 90 degrees
\- Turn right 90 degrees



Axiom: FFF-FF-F-F+F+FF-F-FFF

Geometric Interpretation

- Left: – (a) Right: + (a)
- Down: & (a) Up: ^ (a)
- RollL: / (a) RollR: \ (a)
- TurnAround: |



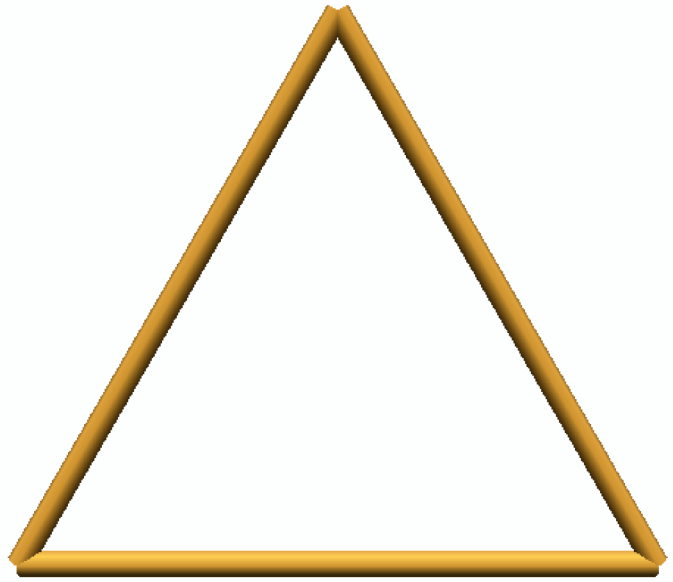
$$R_U(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$R_L(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix},$$

$$R_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

First geometric figures with L-Py

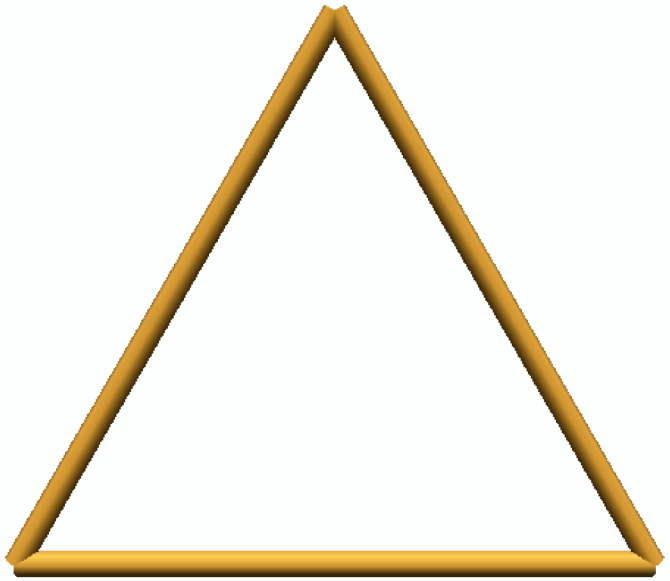
- Drawing an equilateral triangle



Axiom: -(90) F(5) …

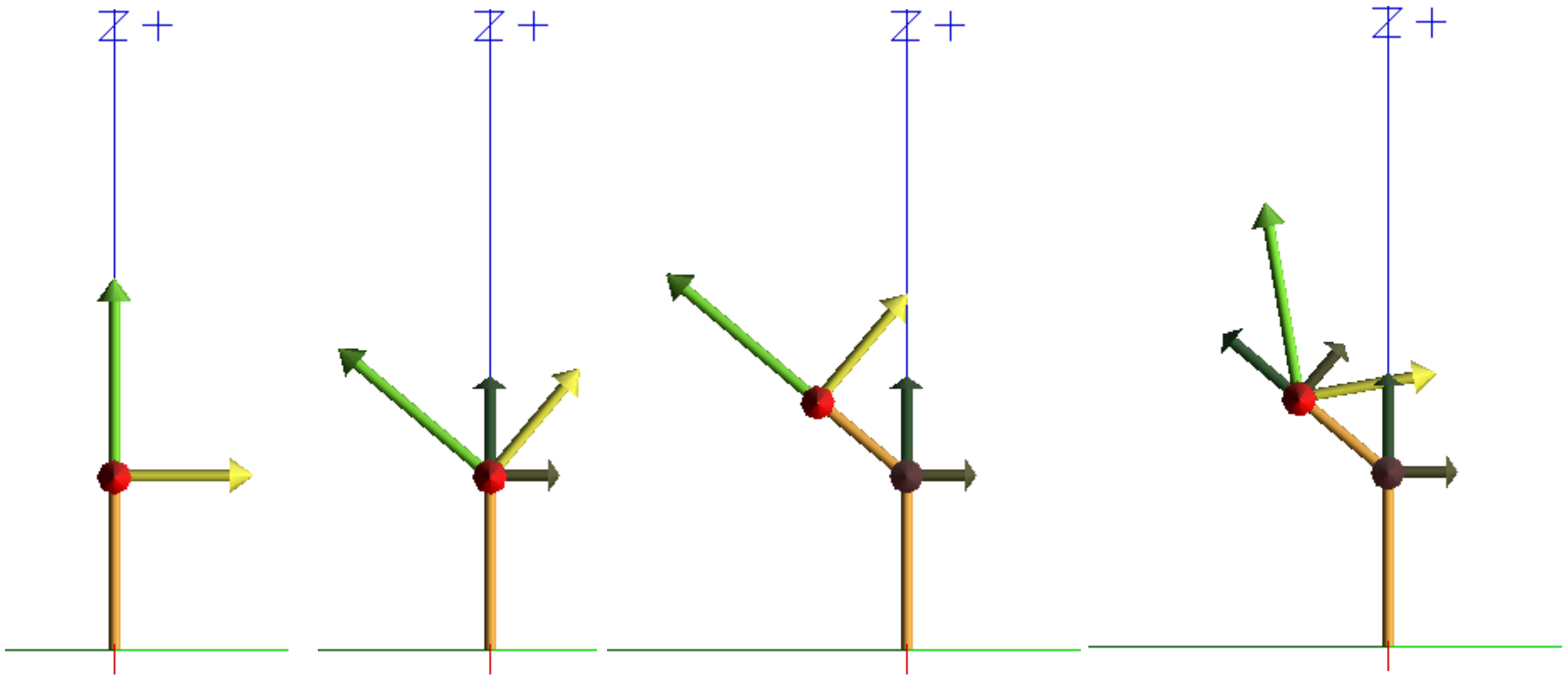
First geometric figures with L-Py

- Drawing an equilateral triangle



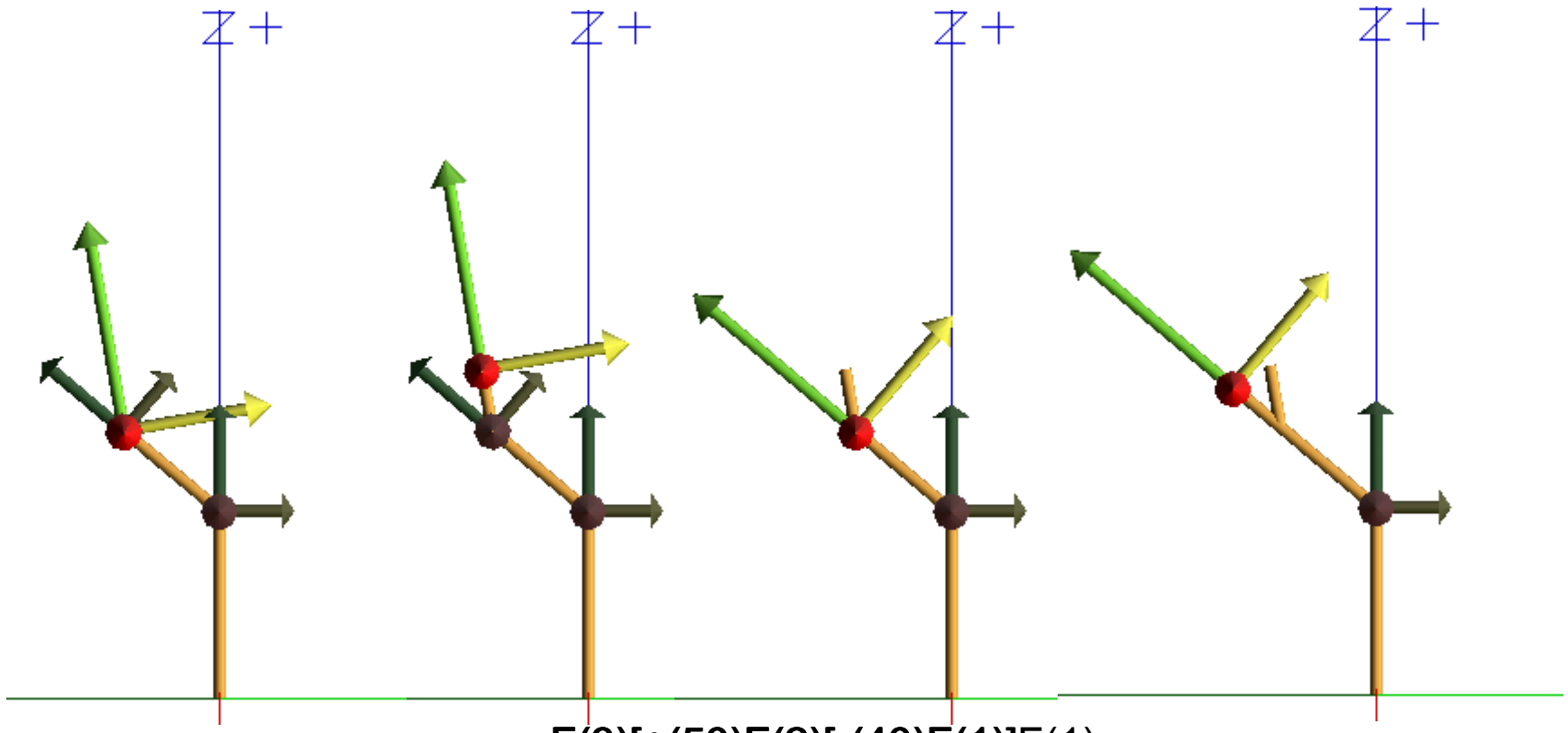
Axiom: -(90) F(5) +(120) F(5) +(120) F(5)

Geometric interpretation of bracketed strings […]



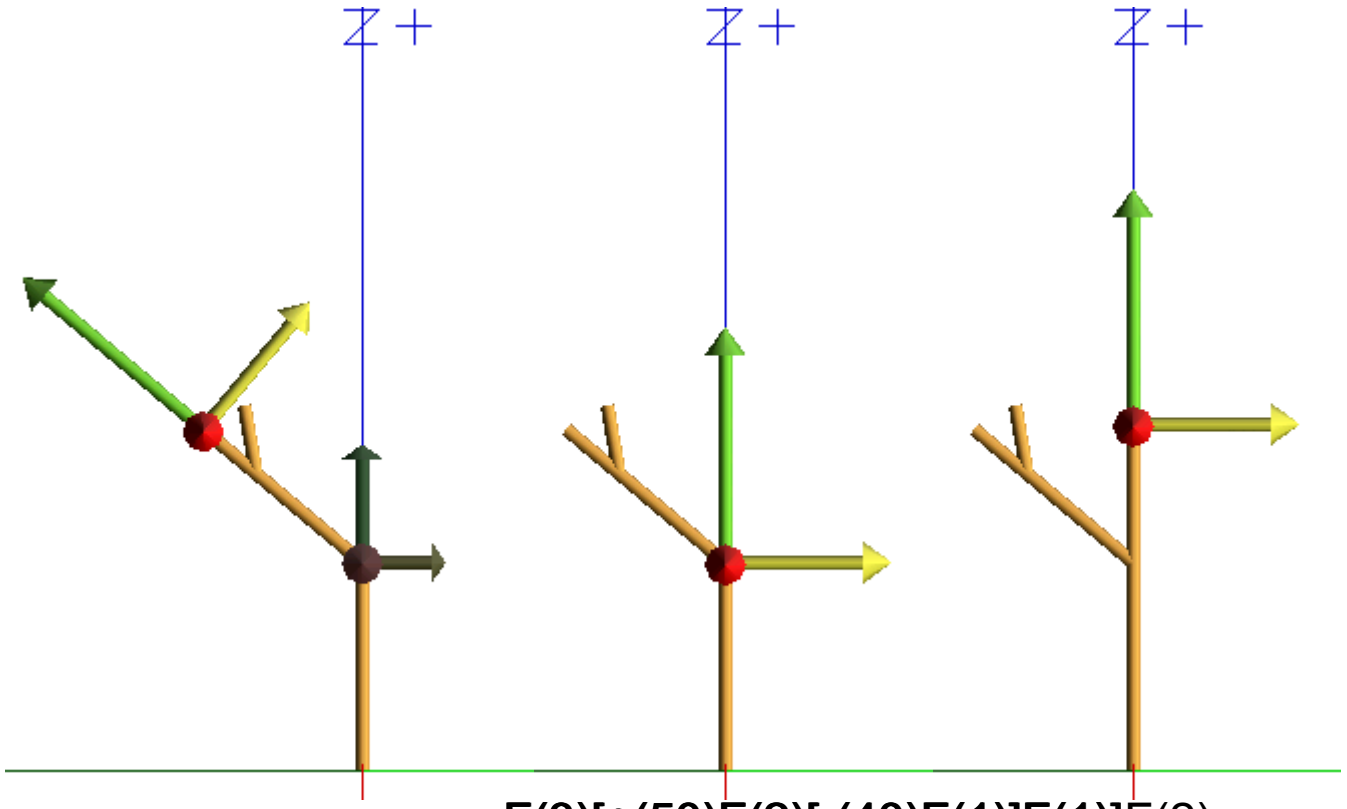
F(3)[+(50)F(2)[-(40)

Use of a stack to manage bifurcation points […]



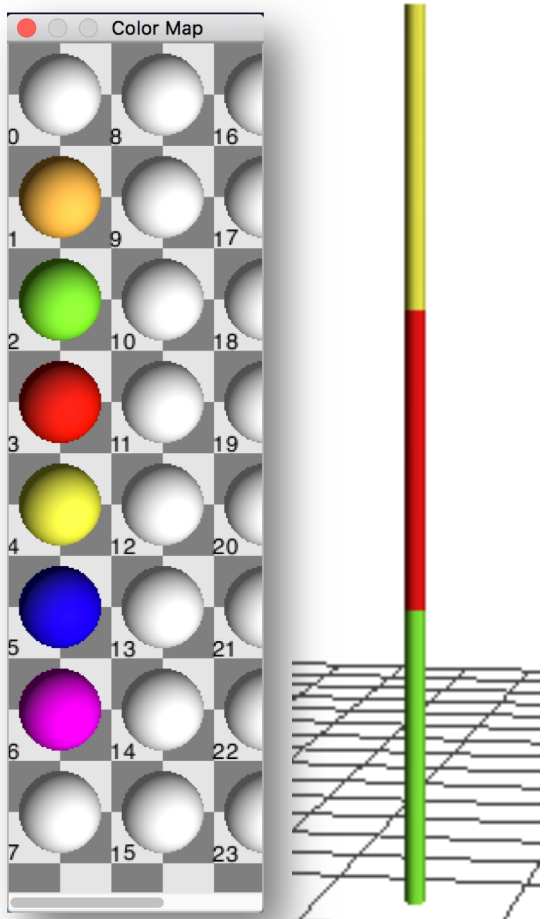
F(3)[+(50)F(2)[-(40)F(1)]F(1)

Use of a stack to manage bifurcation points […]

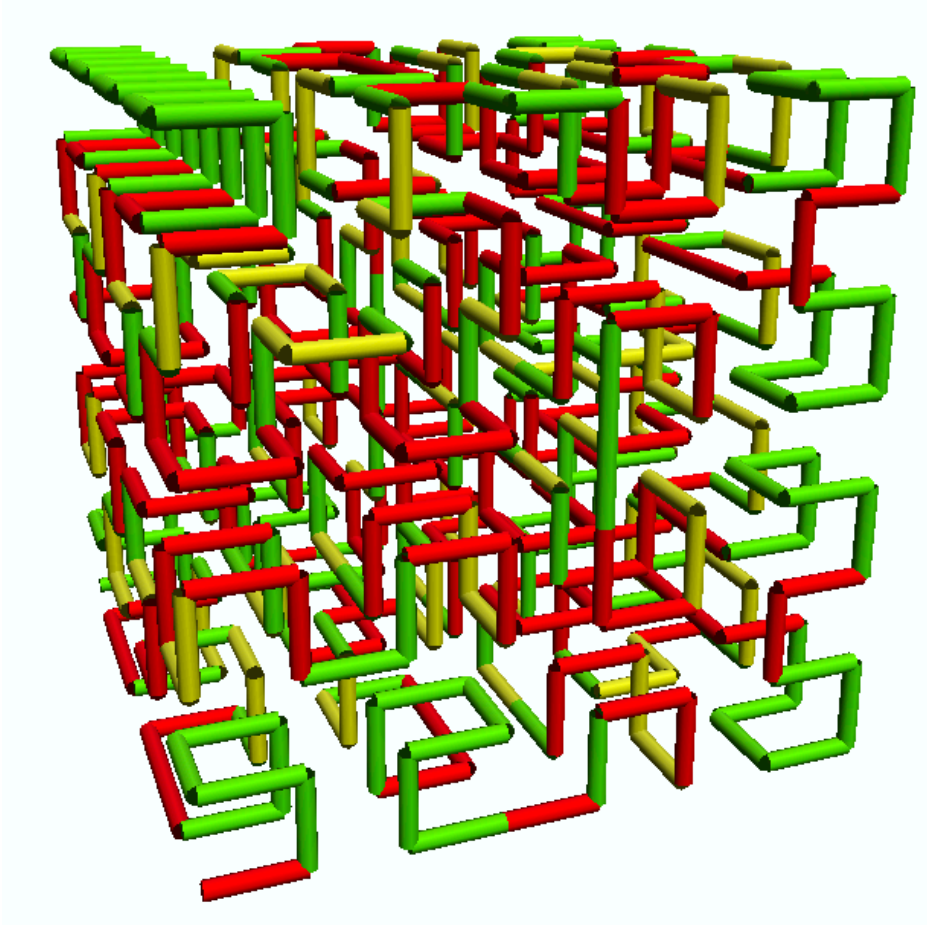


Geometric Interpretation

- Moving to a precise location :
  - `@M(x,y,z)`
- Locating Turtle
  - `Frame()`
- Parameter of the plot:
  - Width : `_(w)`
  - Color : `,(c)`
- Section of the plot
  - `SetContour`

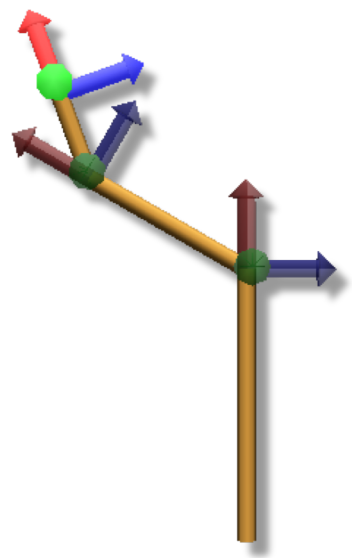


Turtle Interpretation



```
Axiom : A
A   --> B-F+CFC+F-D&F^D-F+&&CFC+F+B//
B   --> A&F^CFB^F^D^^-F-D^|F^B|FC^F^A//
C   --> |D^|F^B-F+C^F^A&&FA&F^C+F+B^F^D//
D   --> |CFB-F+B|FA&F^A&&FB-F+B|FC//
```

Lpy help



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

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.
		Rotation
Pinpoint		Orient turtle toward (x,y,z) . Params : x, y, z (optionals, default = 0).
PinpointRel		Orient turtle toward pos+(x,y,z) . Params : x, y, z (optionals, default = 0).
@R	**SetHead**	Set the turtle Heading and Up vector. Params: hx, hy, hz, ux, uy, uz (optionals, default=0,0,1, 1,0,0).
+	**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).
iRollL		Roll left intrinsically around Heading vector. Params : angle (optional, in degrees).
iRollR		Roll right intrinsically around Heading vector. Params : angle (optional, in degrees).
|	**TurnAround**	Turn around 180deg the Up vector.
@v	**RollToVert**	Roll 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
		Position
@M	**MoveTo**	Set the turtle position. Params : x, y, z (optionals, default = 0).
MoveRel		Move relatively from current the turtle position. Params : x, y, z (optionals, default = 0).
		Scale
@Dd	**DivScale**	Divides the current turtle scale by a scale factor, Params : scale factor (optional, default = 1.0).
@Di	**MultScale**	Multiplies the current turtle scale by a scale factor, Params : scale factor (optional, default = 1.0).
@D	**SetScale**	Set the current turtle scale, Params : scale (optional, default = 1.0).
		Primitive
F		Move forward and draw. Params: length , topradius.
f		Move forward and without draw. Params: length.
@Gc	**StartGC**	Start a new generalized cylinder.
@Ge	**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.
LineTo		Trace line to (x,y,z) without changing the orientation. Params : x, y, z (optionals, default = 0).
OLineTo		Trace line toward (x,y,z) and change the orientation. Params : x, y, z (optionals, default = 0).

The rewriting system

Rule application

- Parametric rules

```
A(x) --> B(2*x)
```

apply to `A(3)` but not `B(5)` and not `A(3,5)`.

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

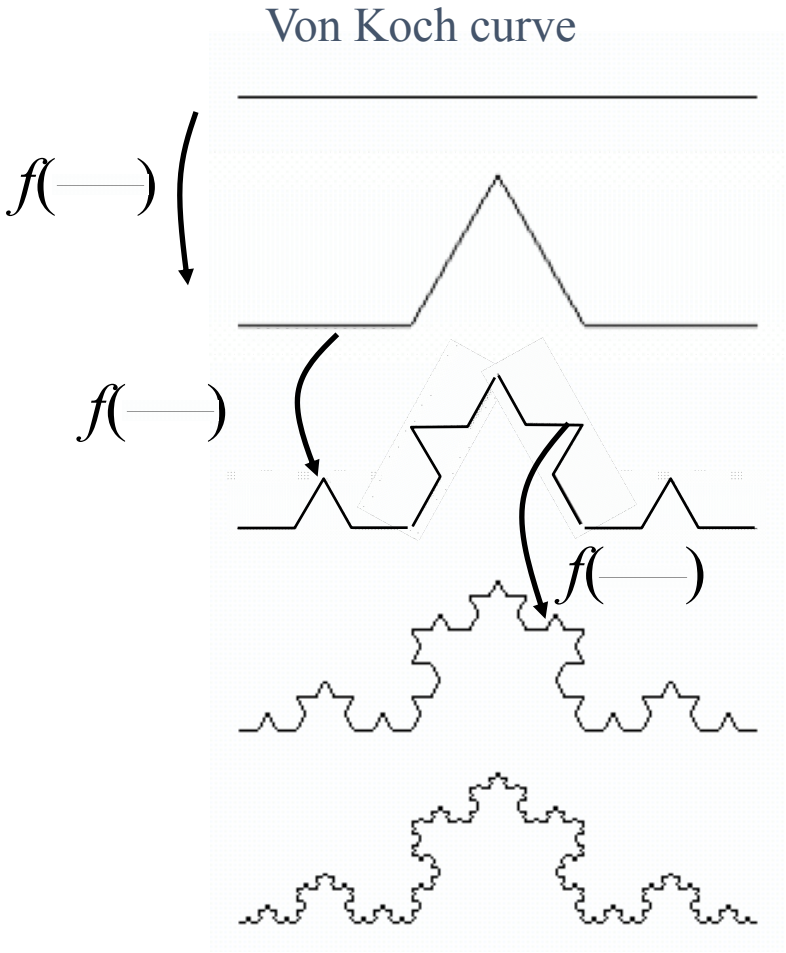
apply to `A(2)` but not `A(3)`.

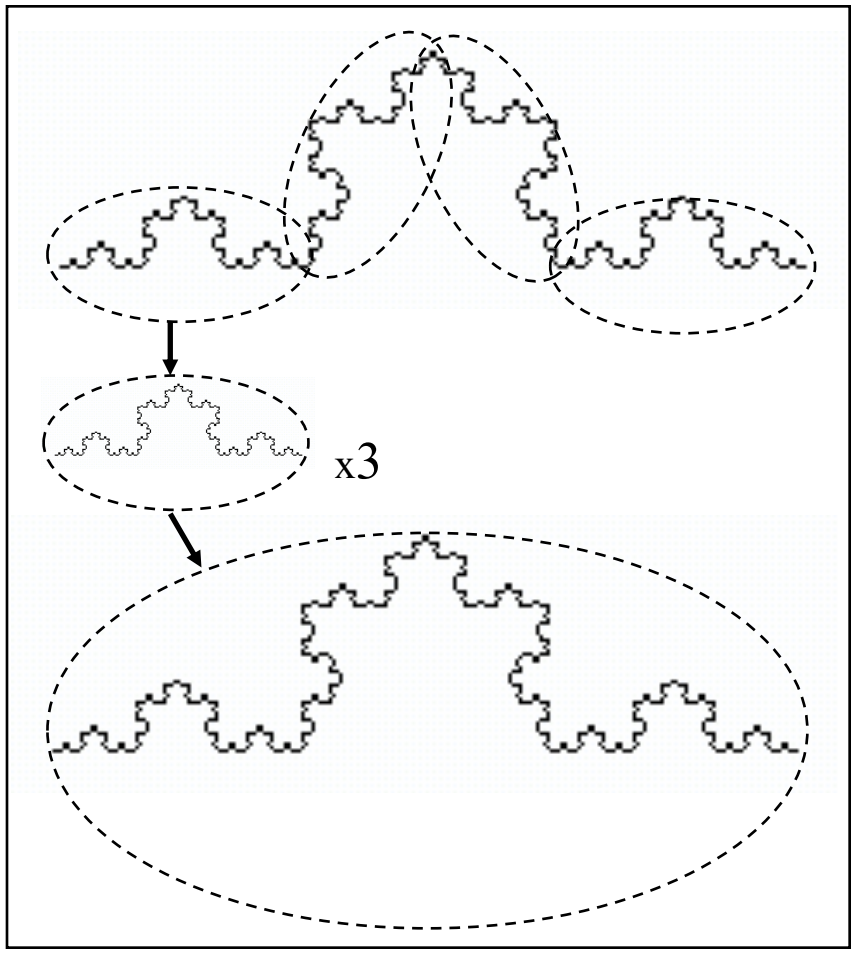
- Possibility to make production in multiple steps

```
B(x) :
  for i in range(x):
      nproduce F +(5)
```

Application to fractals

Mathematical object whose structure is invariant by scale change: Mandelbrot, B.B. (1983), W.N. Freeman

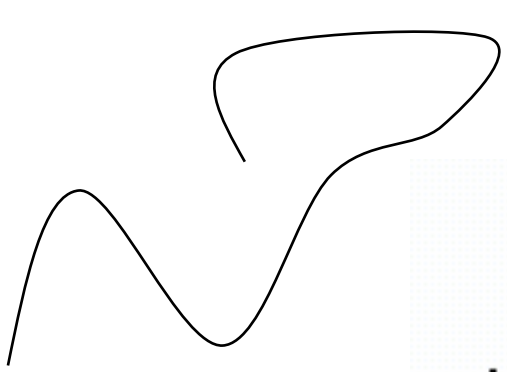


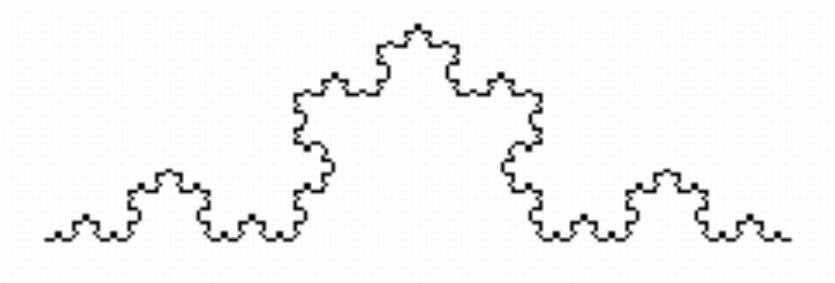


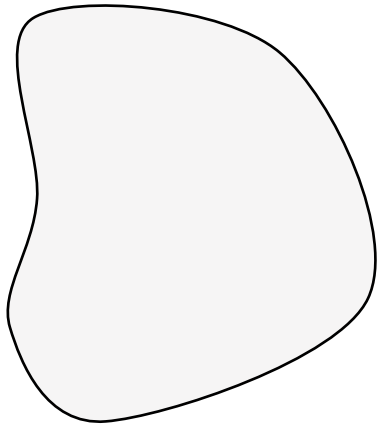
autosimilarité !

Fractal dimension

The fractal dimension of an irregular object can be not an integer. It quantifies complexity of fractals as a ratio of change in details to the change of scale.





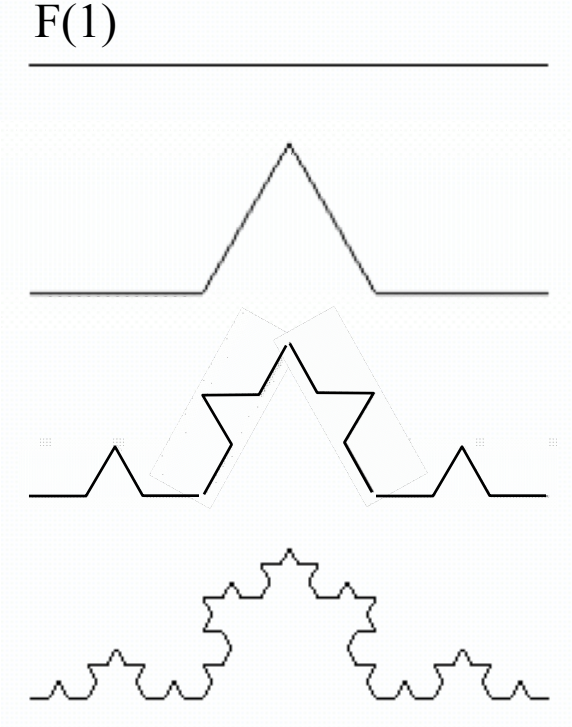


$D = 1$ $D = 1.262$ $D = 2$

When the scale of measured is divided by 3, its length is multiply by 4.

$$D = \log 4 / \log 3$$

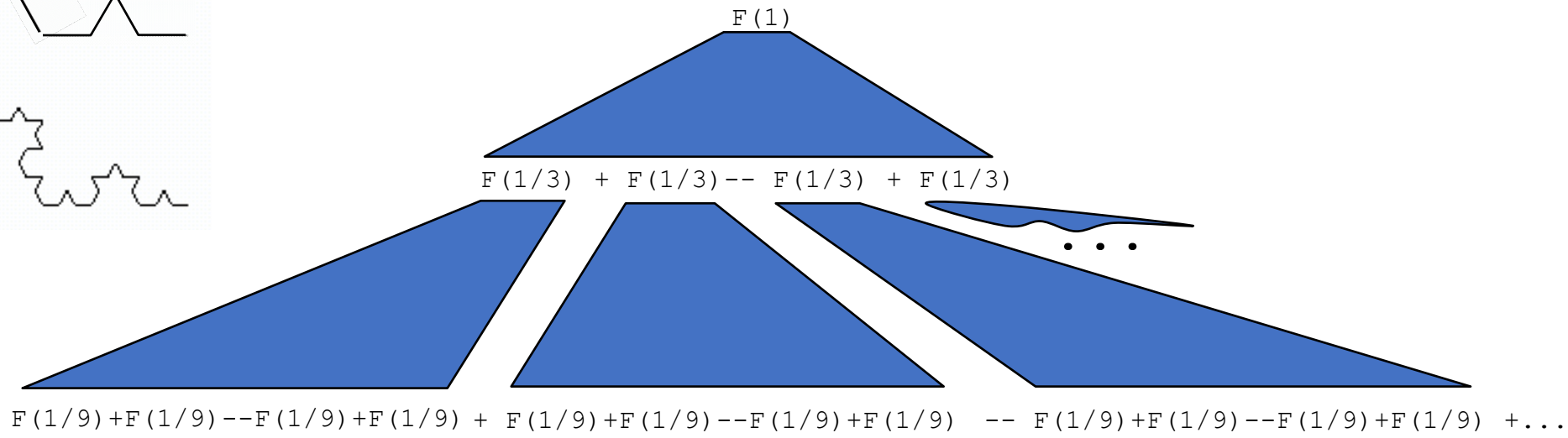
L-Systems for simple fractals



G = ({F,+,-}, { F(1) }, P)

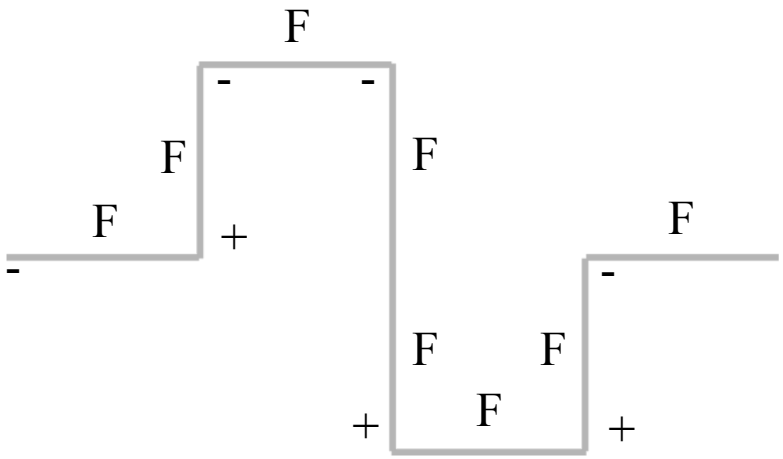
Parametric rule de P:

F(x) --> F(x/3)+F(x/3)--F(x/3)+F(x/3)



Minkowski curve

a.k.a the Minkowsi sausage

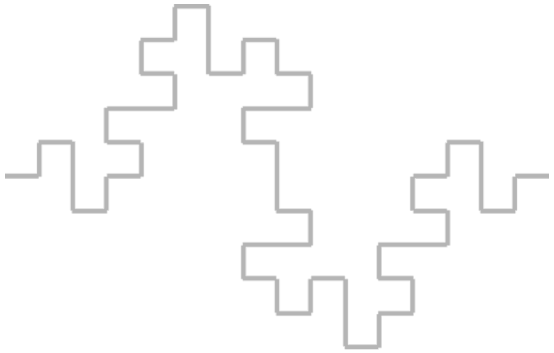


The Axiom:

-F

Production, or re-writing rule:

F --> F+F-F-FF+F+F-F



After one step:

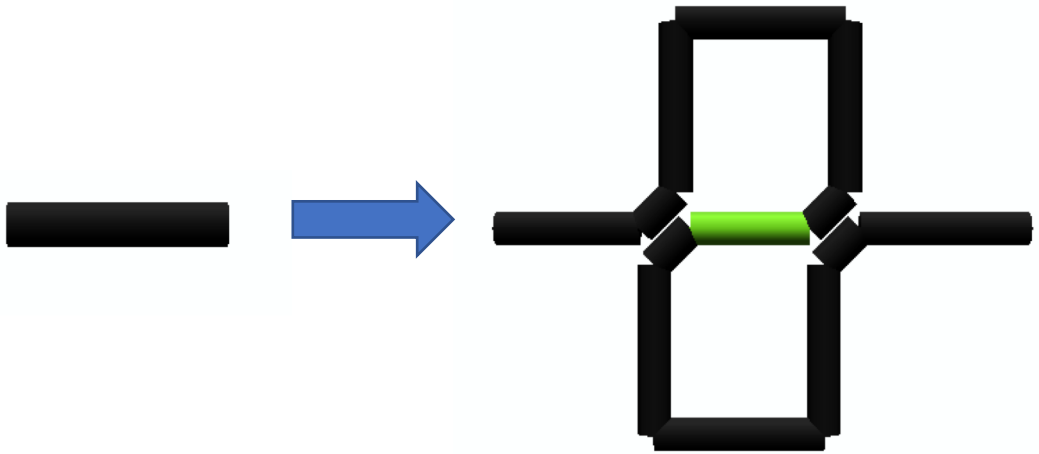
-F+F-F-FF+F+F-F

After two steps:

- F+F-F-FF+F+F-F + F+F-F-FF+F+F-F - F+F-F-FF+F+F-F
- F+F-F-FF+F+F-F F+F-F-FF+F+F-F + F+F-F-FF+F+F-F
+ F+F-F-FF+F+F-F - F+F-F-FF+F+F-F

More details on ‘Lsystem.ipynb’

Exercises



The peano curve

The cantor dust

Exercise : Barnley fern

Recursive fractal construction

Exercise : Barnley fern

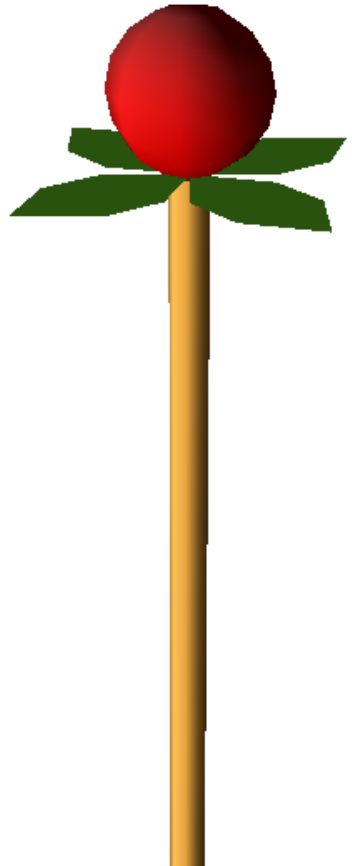
Recursive fractal construction

```
R = 1.5
Axiom: _(0.01) ,(2) A(1)
derivation length: 16
A(s) --> F(s)-(5)[+B(s/R)]F(s)-(5)[/(180)+B(s/R)]F(s)A(s/R)
B(s) --> A(s/R)
```

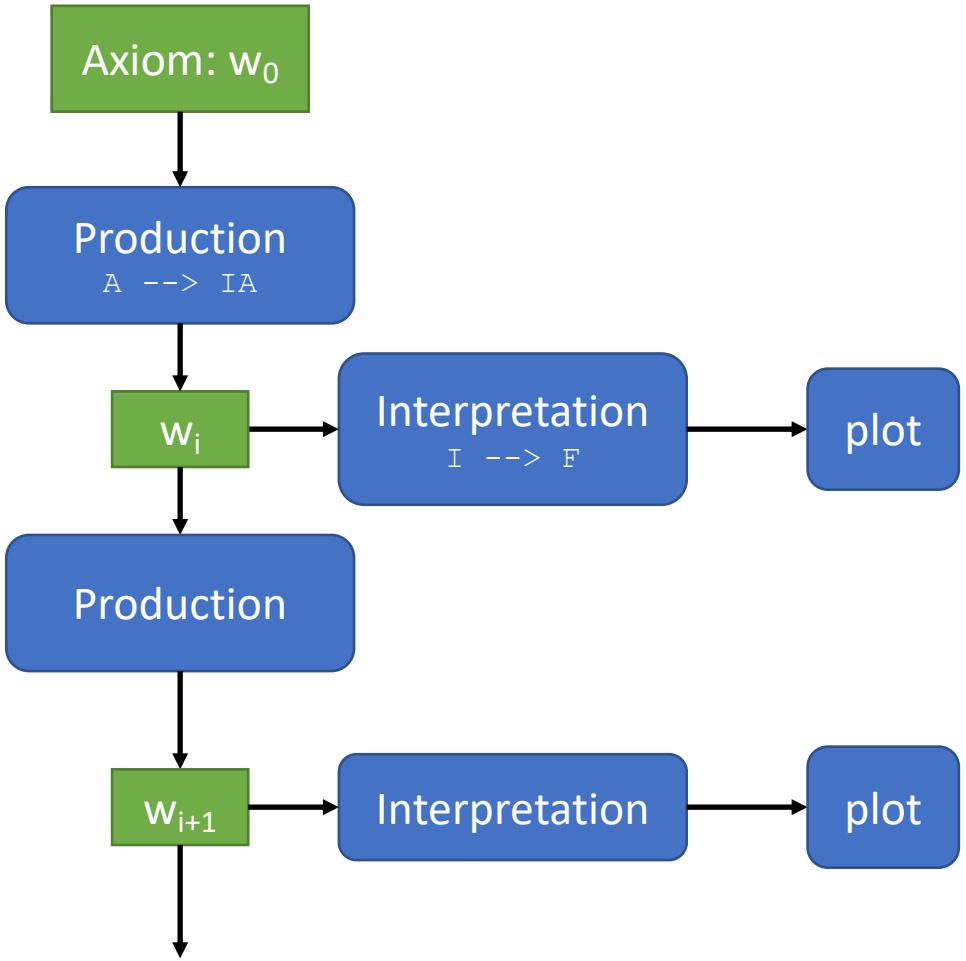
Interpretation

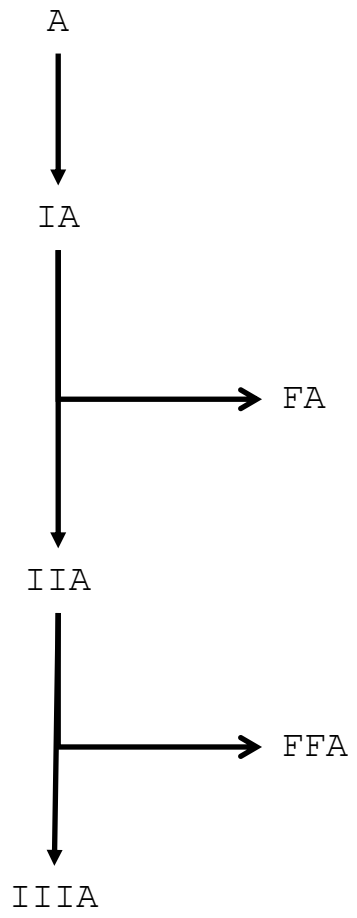
- Interpretation rule of module into geometrical symbol. Can be recursive.

```
interpretation:
maximum depth: 2
A(x) --> F(5)[,(2)/(90)L/(90)L/(90)L/(90)L,(3)f(0.4)@O(0.4)]
L --> [+(90)/(90)~l]
```

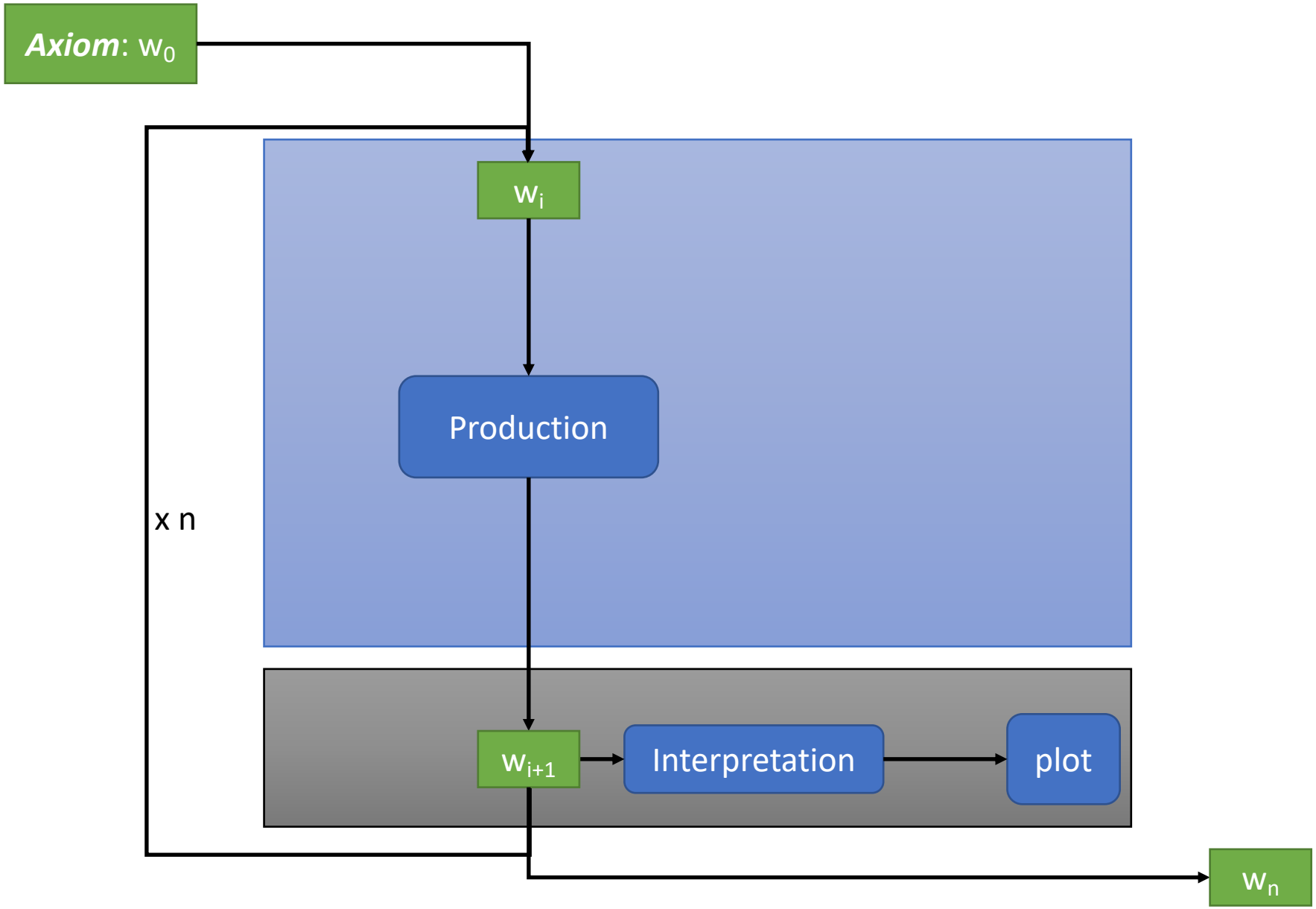


Simulation Pipeline

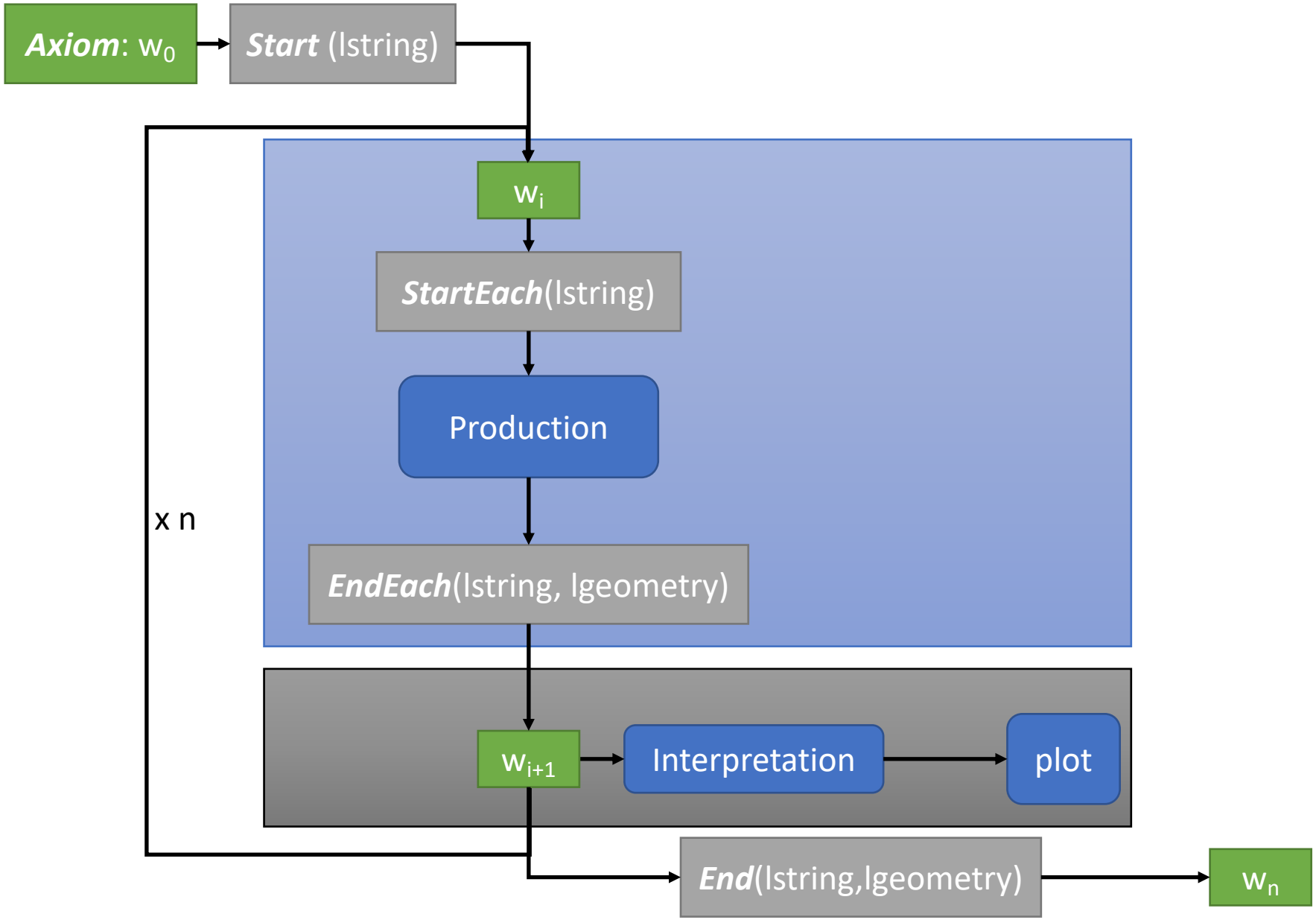




Simulation Pipeline (more details)



Simulation Pipeline (more details)



Controlling Simulation

```
def Start(lstring):
      pass
def End([lstring[,scene]]):
      pass
def StartEach(lstring):
      pass
def EndEach([lstring[,scene]]):
      pass
```

```
def StartEach():
 if isForward(): backward()
 else: forward()
```

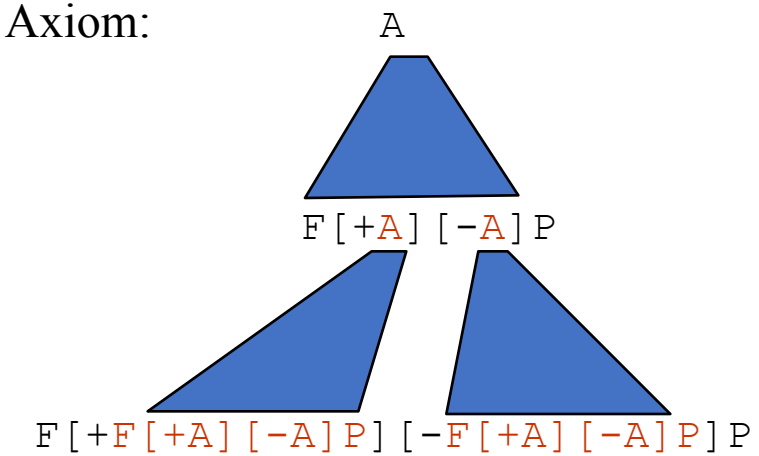
```
def StartEach():
 if getIterationNb() % 2: useGroup(3)
 else: useGroup(2)

production:
A --> BA
Group 2:
B --> C
Group 3:
B --> D
EndGroup
```

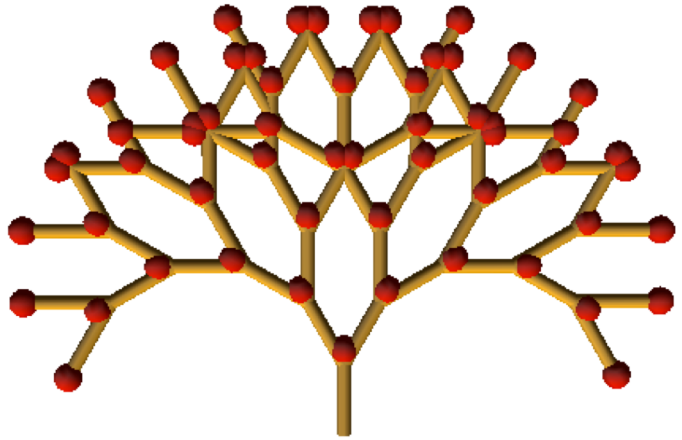
Rewritting trees

Rules:

```
A --> F[+A][-A]P
```

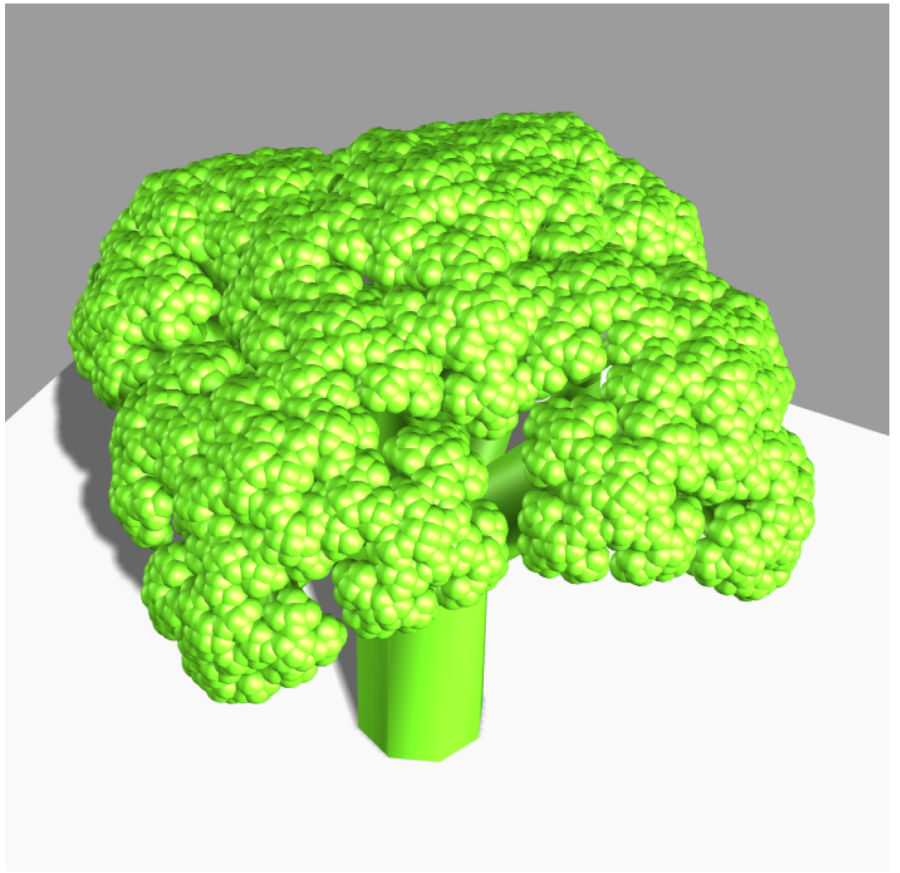
Axiom:

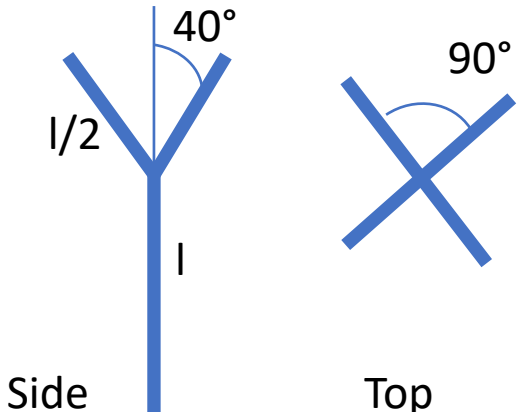




Exercise

- Generate a broccoli
  - Tree structure with 4 children internode at each node
    - Insertion angle: 40
    - 90 degree between each children
    - Width that depend on the order of ramification
      - Scaling factor of 0.5 between each order
    - Finish structure with sphere with size double from terminal internode

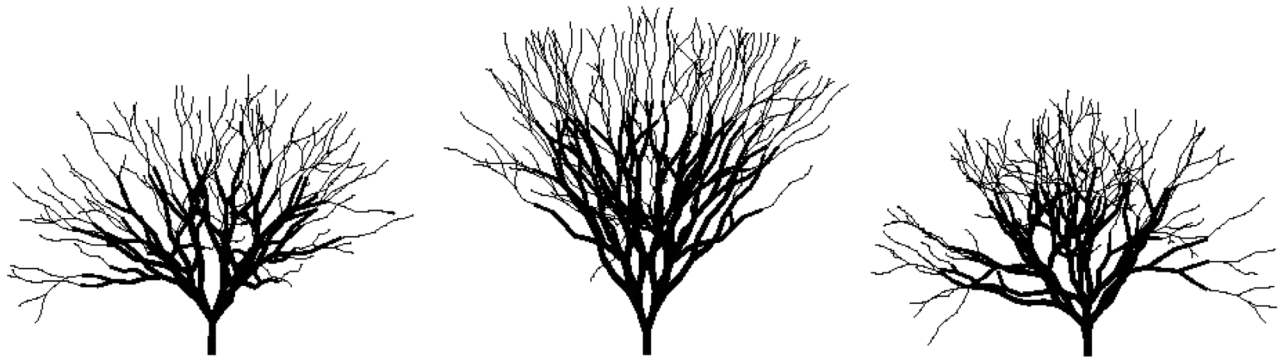




Stochastic L-systems

- Insertion of stochastic productions

```
from random import *

Axiom: A
production:
A :
  if random() < 0.5: produce B
  else produce C
```

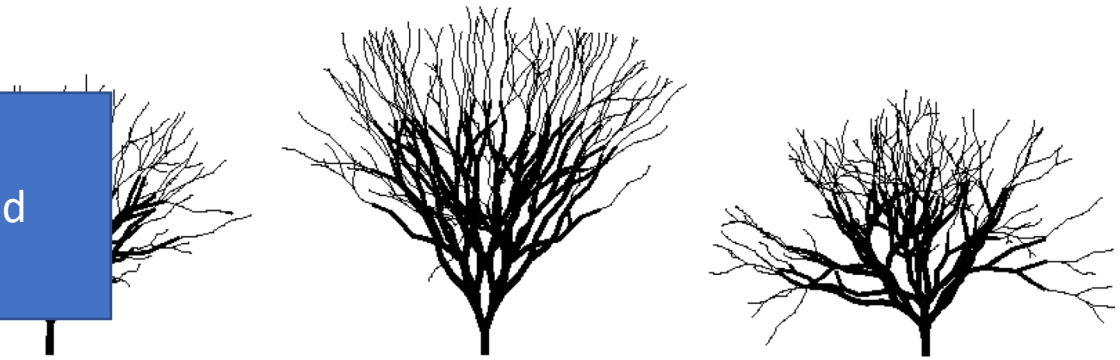


Stochastic L-systems

- Insertion of stochastic production

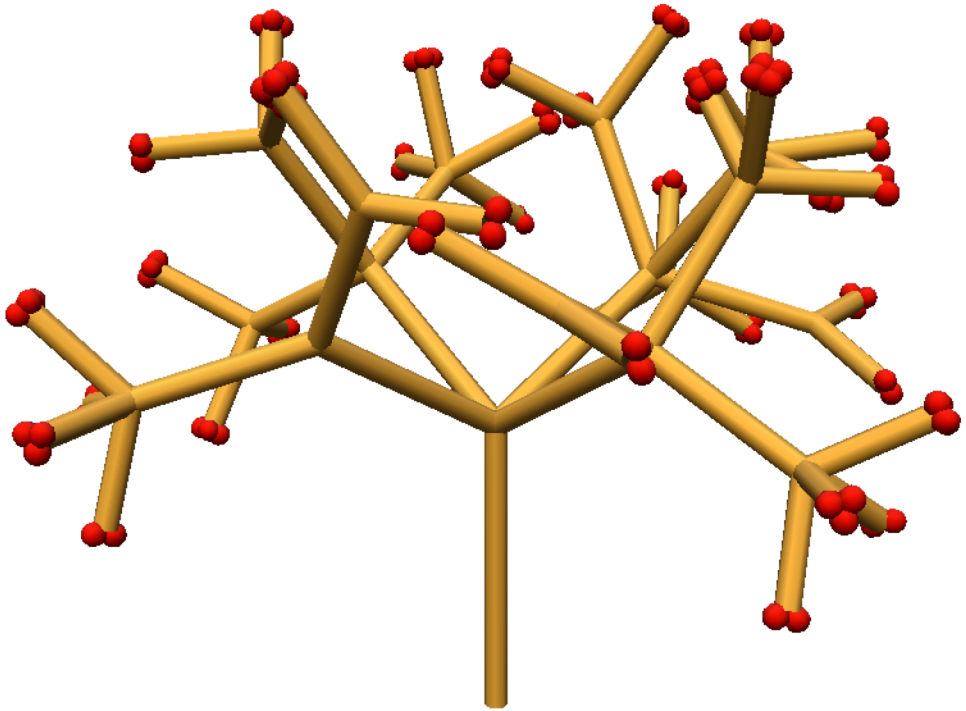
```
from random import *
seed(50)
Axiom: A
production:
A :
  if random() < 0.5: produce B
  else produce C
```

Control of the random seed



Random trees

- Generate a tree such as a the end of each segment, a random number of lateral segment children (between 2 and 4) are generated.
  - Insertion angle: 60
  - Divergence angle between segments at the same node: proportional to number of segments i.e. 360/nb



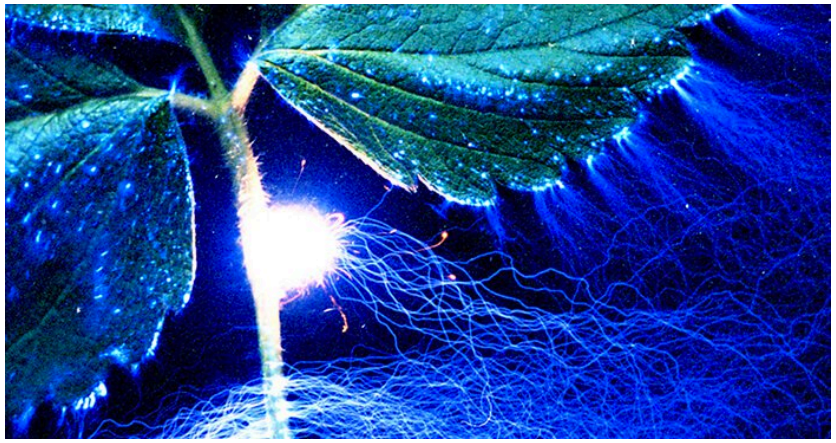
Simulations of continuous development

- Use infinitesimal increments *dt* or *dx*

```
phyllochrone = 1
maxgrowthtime =1
dt = 0.1

Axiom: A(0)

A(t):
 if t <= 0 : produce I(0)/(137.5) L(0) A(0)
 else: produce A(t+dt)

I(t) --> I(min(maxgrowthtime,t+dt))
L(t) --> L(min(maxgrowthtime,t+dt))

interpretation:
I(t) --> F(t * Li)
L(t) --> [&(90*t),(2) ~l(t*Lf)]
```

Pruning

Information transfer and signal



Contextual L-systems

- Left and Right Contexts

```
A < S > B --> c
```

apply on `ASB`

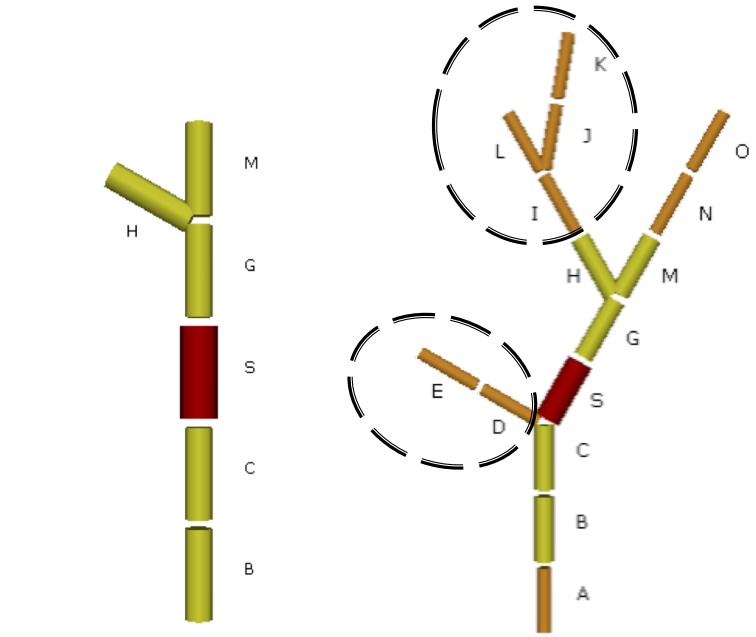
- Parametric contexts to retrieve information

```
S(x) < S(y) --> S(f(x,y))
```

- Possibility to ignore some symbols (for instance geometrical ones)

```
ignore: +-/\
```

Context application within trees



BC < S > G[H]M

in

ABC[DE]SG[HI[JK]L]MNO

(Prusinkiewicz et al., 90, 94, 96, 07)

Information transfer

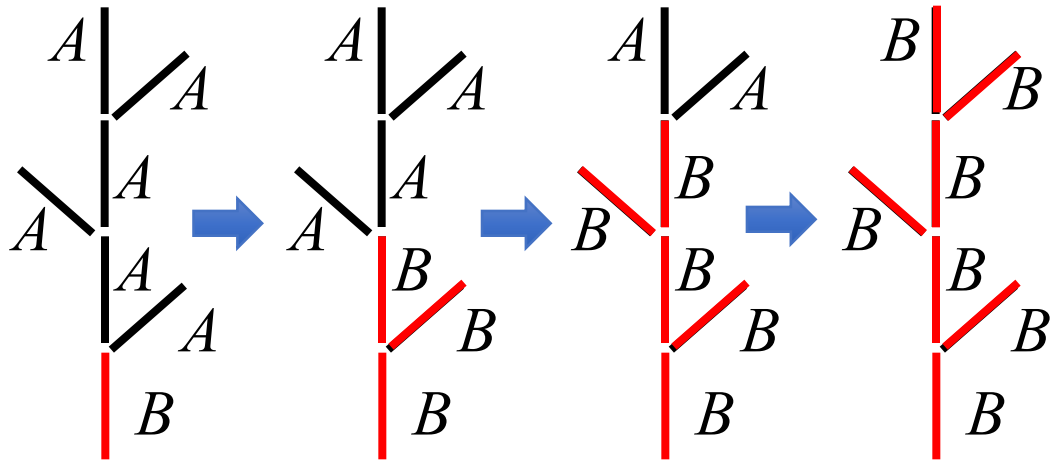
Contextual rules:

```
B < A  --> B
```

Axiom:

```
B[A]A[A]A[A]A
```

```
B[A]A[A]A[A]A
B[B]B[A]A[A]A
B[B]B[B]B[A]A
B[B]B[B]B[B]B
```



Requires n iterations

Information transfer

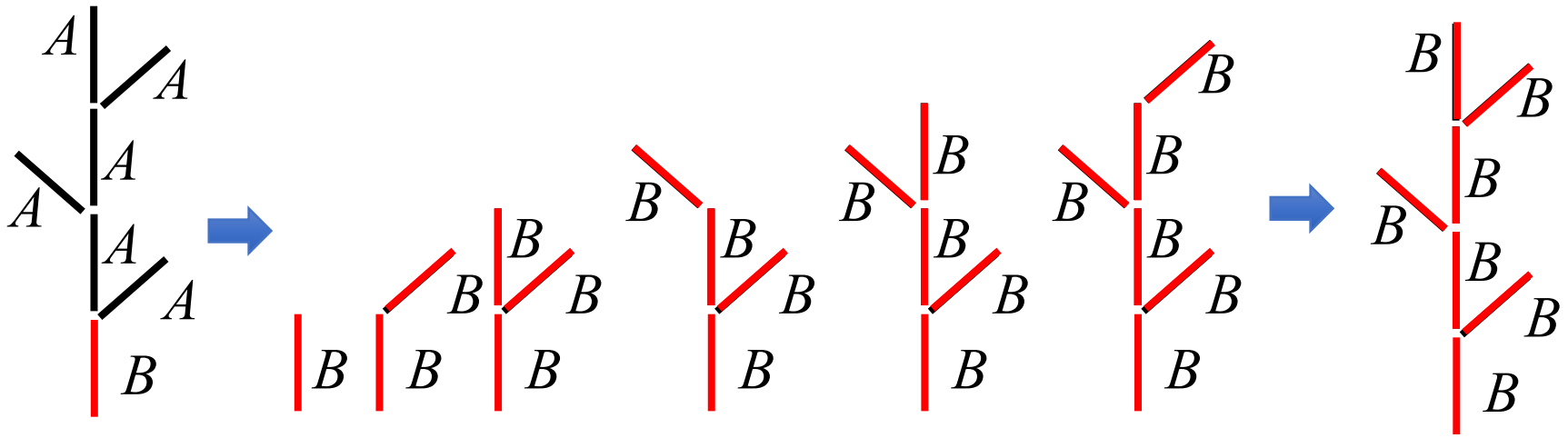
Contextual rules:

```
B << A  --> B
```

Axiom:

```
B[A]A[A]A[A]A
```

```
B
B[
B[B]
B[B]B
…
B[B]B[B]B[B]B
```



Requires 1 iterations

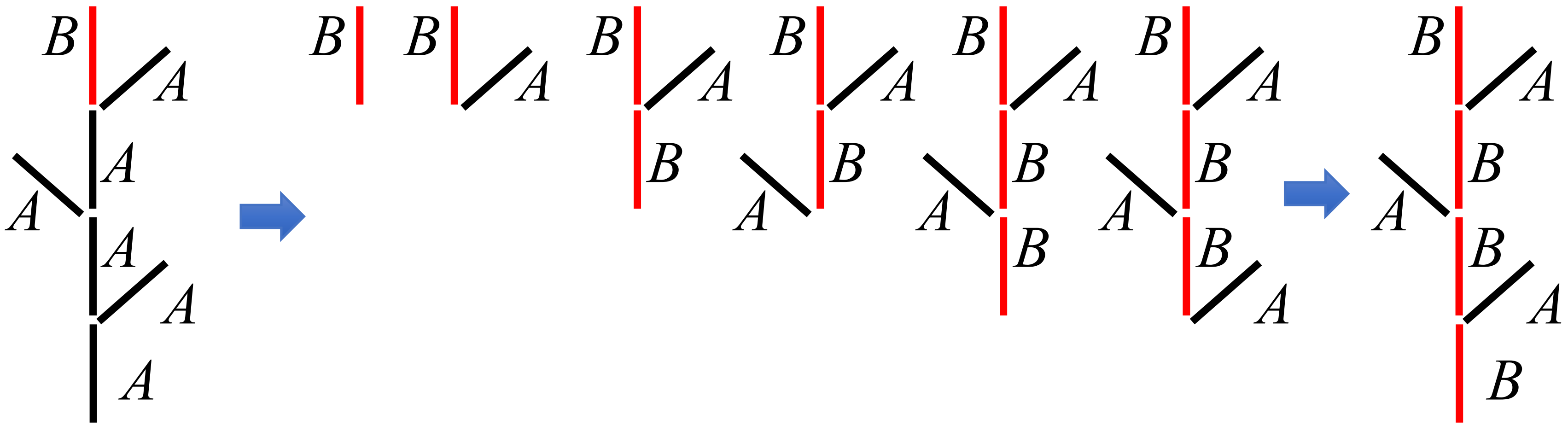
Information transfer

Contextual rules:

```
A >> B --> B
```

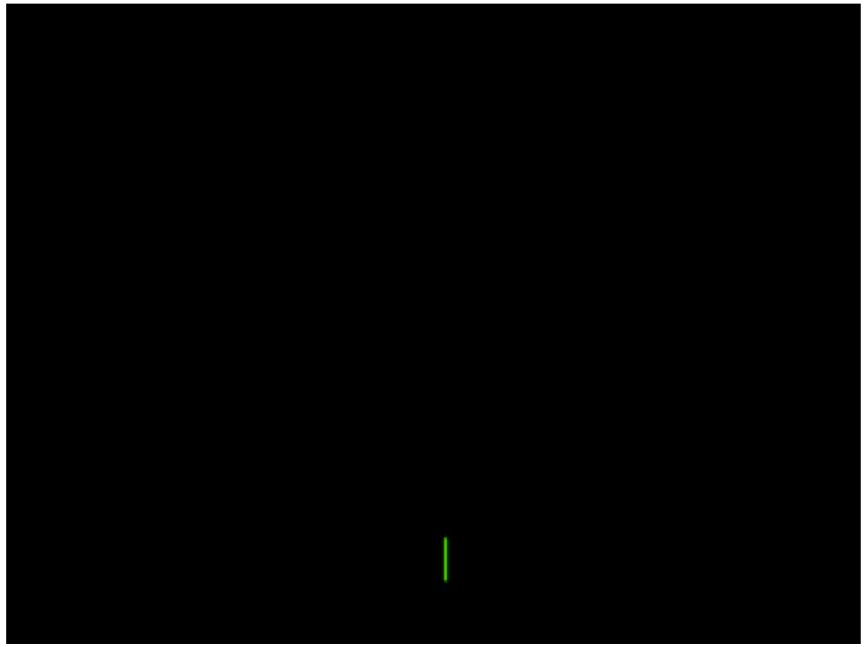
Axiom: `A[A]A[A]A[A]B`

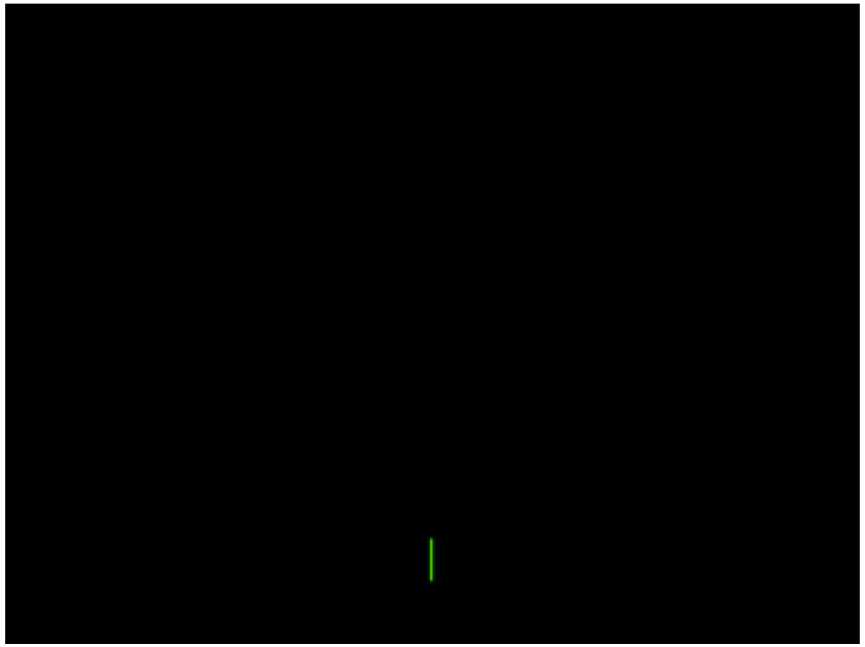
```
           B
          ]B
        [B]B
       B[B]B
           …
B[B]B[B]B[B]B
```



- Backward rewriting mode : from leaf to root
- Not symetrical

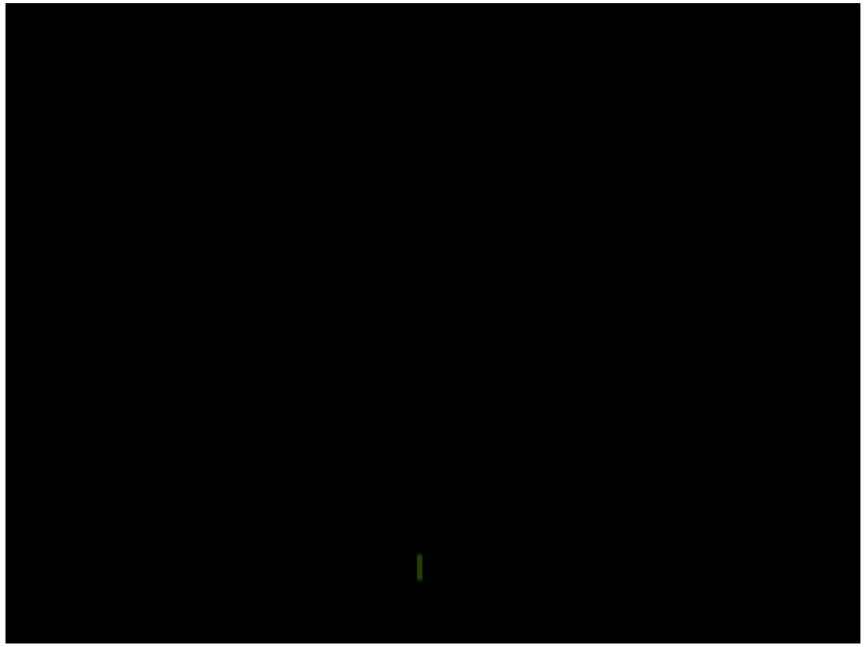
Signal et Flowering





From http://algorithmicbotany.org/vmm-deluxe/Section-10.html

Signal et Flowering

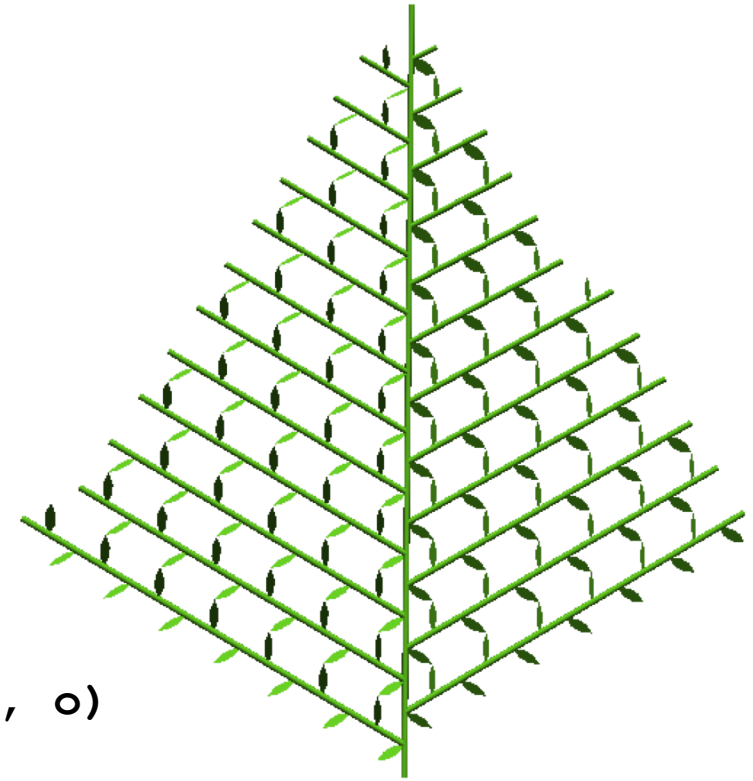


Mycelis muralis

- Complete the following L-systems to simulate signal propagation

Exercise

```
Delay, LDelay = 5, 10
T = 120
Axiom: R(T) I(0) A(Delay, 0)
derivation length: 150
production:
consider: R I A B

R(t) --> R(t-1)

I(x) < A(d,o) :
  # produce flower
  if x == 1 : produce W
  # continue to growth
  elif d > 0: produce A(d-1, o)
  else:
    # produce lateral apex or leaf
    if o == 0: nproduce [ +(60) A(0,o+1) ]
    else : nproduce [ +(60) /(60) ,(2) ~l
    produce I(0) /(180) A(Delay if o == 0 else LDelay, o)
```



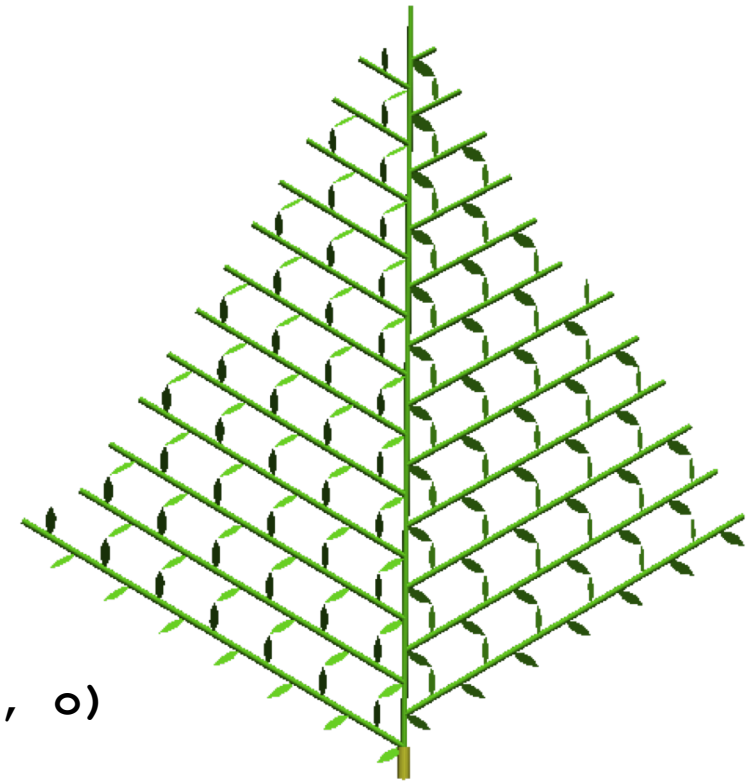
- Complete the following Lsystems to simulate signal propagation

Exercise

```
Delay, LDelay = 5, 10
T = 120
Axiom: R(T) I(0) A(Delay, 0)
derivation length: 150
production:
consider: R I A B
```

```
R(t) --> R(t-1)
R(t) < I(x):
    # Wait for the root to propagate
    if t < 0: produce I(1)
```

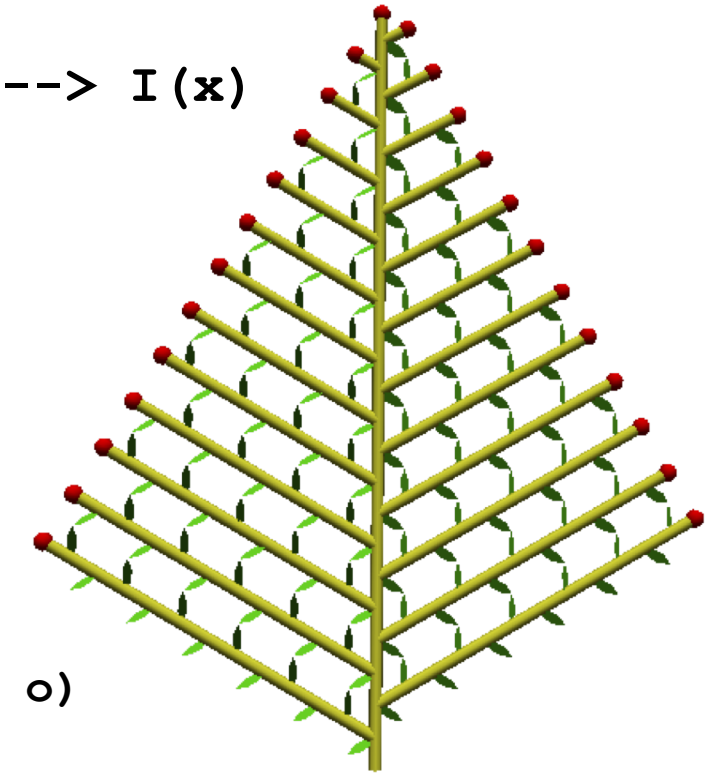
```
I(x) < A(d,o) :
  # produce flower
  if x == 1 : produce W
  # continue to growth
  elif d > 0: produce A(d-1, o)
  else:
    # produce lateral apex or leaf
    if o == 0: nproduce [ +(60) A(0,o+1) ]
    else : nproduce [ +(60) /(60) ,(2) ~l
    produce I(0) /(180) A(Delay if o == 0 else LDelay, o)
```



- Complete the following Lsystems to simulate signal propagation

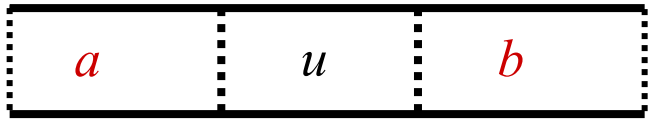
Exercise

```
Delay, LDelay = 5, 10
T = 120
Axiom: R(T) I(0) A(Delay, 0)
derivation length: 150
production:
consider: R I A B

I(x) < A(d,o) :
  # produce flower
  if x == 1 : produce W
  # continue to growth
  elif d > 0: produce A(d-1, o)
  else:
    # produce lateral apex or leaf
    if o == 0: nproduce [ +(60) A(0,o+1) ]
    else : nproduce [ +(60) /(60) ,(2) ~l
    produce I(0) /(180) A(Delay if o == 0 else LDelay, o)

R(t) --> R(t-1)
R(t) < I(x):
      # Wait for the root to propagate
      if t < 0: produce I(1)

I(x) < I(s) --> I(x)
```

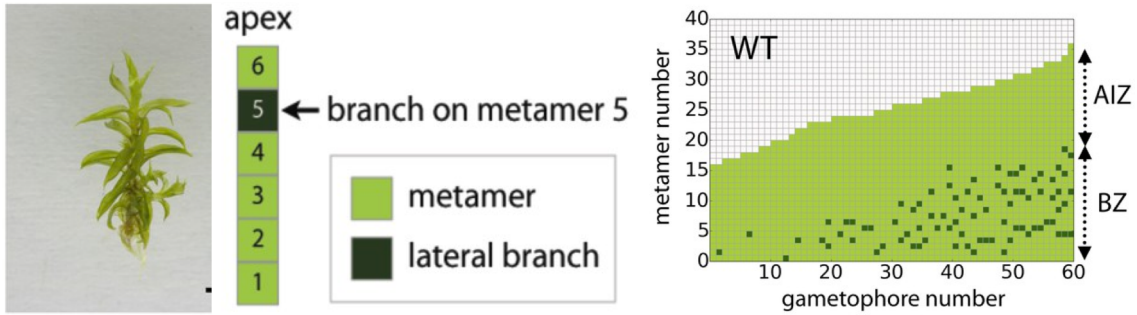


Hormone diffusion

Diffusion of hormones using conservation law: $\frac{\partial u}{\partial t} = \alpha \Delta u + \beta - \gamma u$



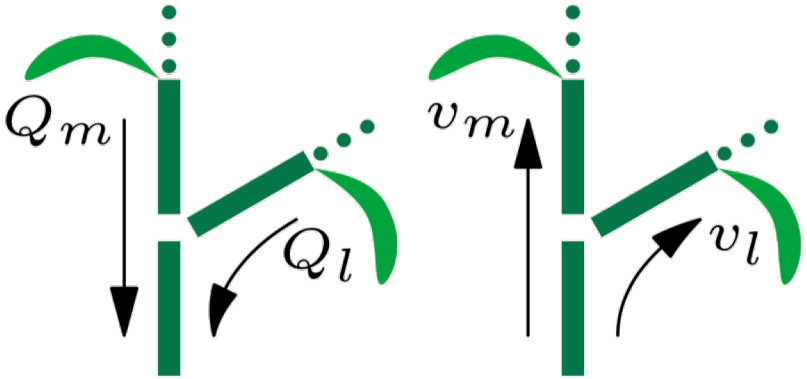

```
I(a) < I(u) > I(b) :
 produce I(u +(a+b-2*u)*alpha +beta - u*gamma)
```



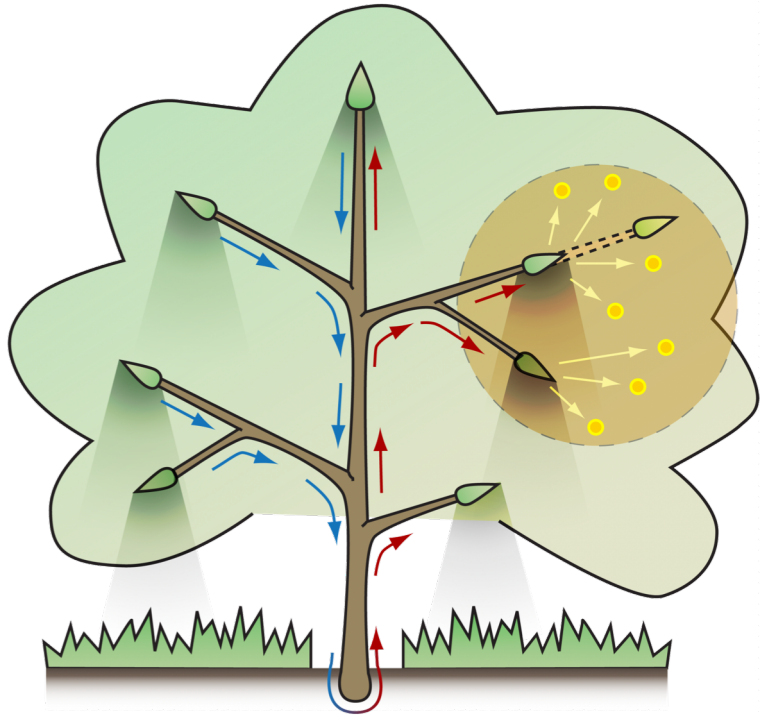
(Coudert et al., eLife, 2015)

Transport of nutrients

- Model of Borchert-Honda, 84:
  - Simulating flows of resources (photosynthates or water and mineral)
  - Apical control is determined with a parameter in resources redistribution.



$$v_m = v\frac{\lambda Q_m}{\lambda Q_m + (1-\lambda)Q_l}$$



```
I(t, v, Q) >> [I(tl, vl, Ql)]I(tm, vm, Qm)   :
  produce I(t, v, Qm+Ql)
```

```
I(tp, vp, Qs) << I(t, v, Q)   :
   Qs = Qp - Q
   l = lambda if t == Apical else (1-lambda)
   vm = vp * (l*Q) / (l*Q + (1-l)*Qs)
   produce I(t, vm, Q)
```

(Palubicki et al., 2009)

Transport of water and carbon

- Coupling of models of aerial and root part of a tree structure
  - Flow of photosynthate control root dev and mineral uptake control aerial dev.

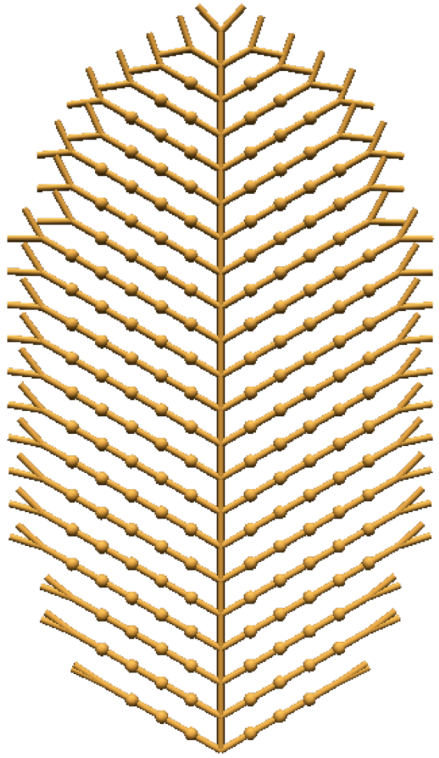
Environment

Environmentally sensitive L-systems

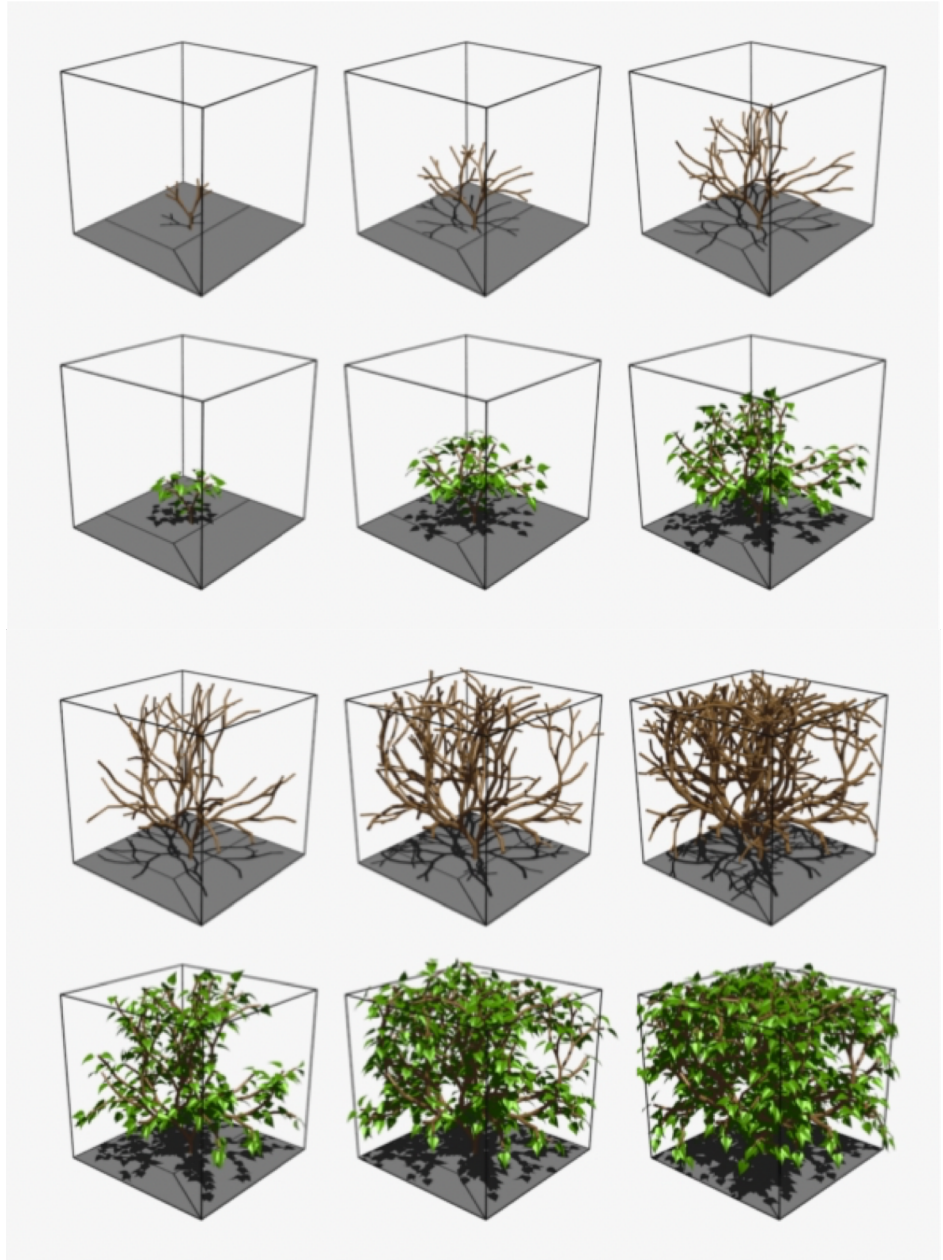
- Request symbols

```
?P(v)  ?H(v)
?U(v)  ?L(v)
?F(p, h, u, l)
```

- Example



```
Axiom: A
production:
A --> [+(60)B][-(60)B]F ?P A
B --> F ?P @O(0.2)B
?P(p) :
  if 4*p.y*p.y +(p.z-10)**2 > 100:
     produce [+(2* p.z)F][-(2* p.z)F]%
  else: produce *
```



(Prusinkiewicz et al., 1994)

Local interaction with environment

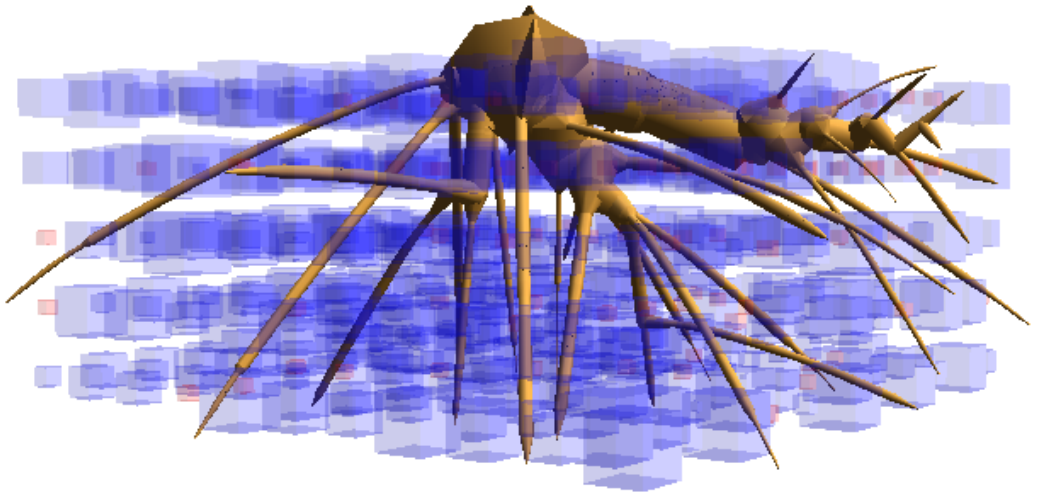
- Use of a structure to represent space

```
class Grid:
  def __init__(self, size = (10,10,10))
    self.values = numpy.random.random(size))
  def getvalue(self, pos3d): …
  def setvalue(self, pos3d, value):…

grid = Grid()

Axiom: A(0) ?P

A(x) ?P(v):
   water = grid.getValue(v)
   water -= dw
   x += dw
   grid.setValue(v, water)
```

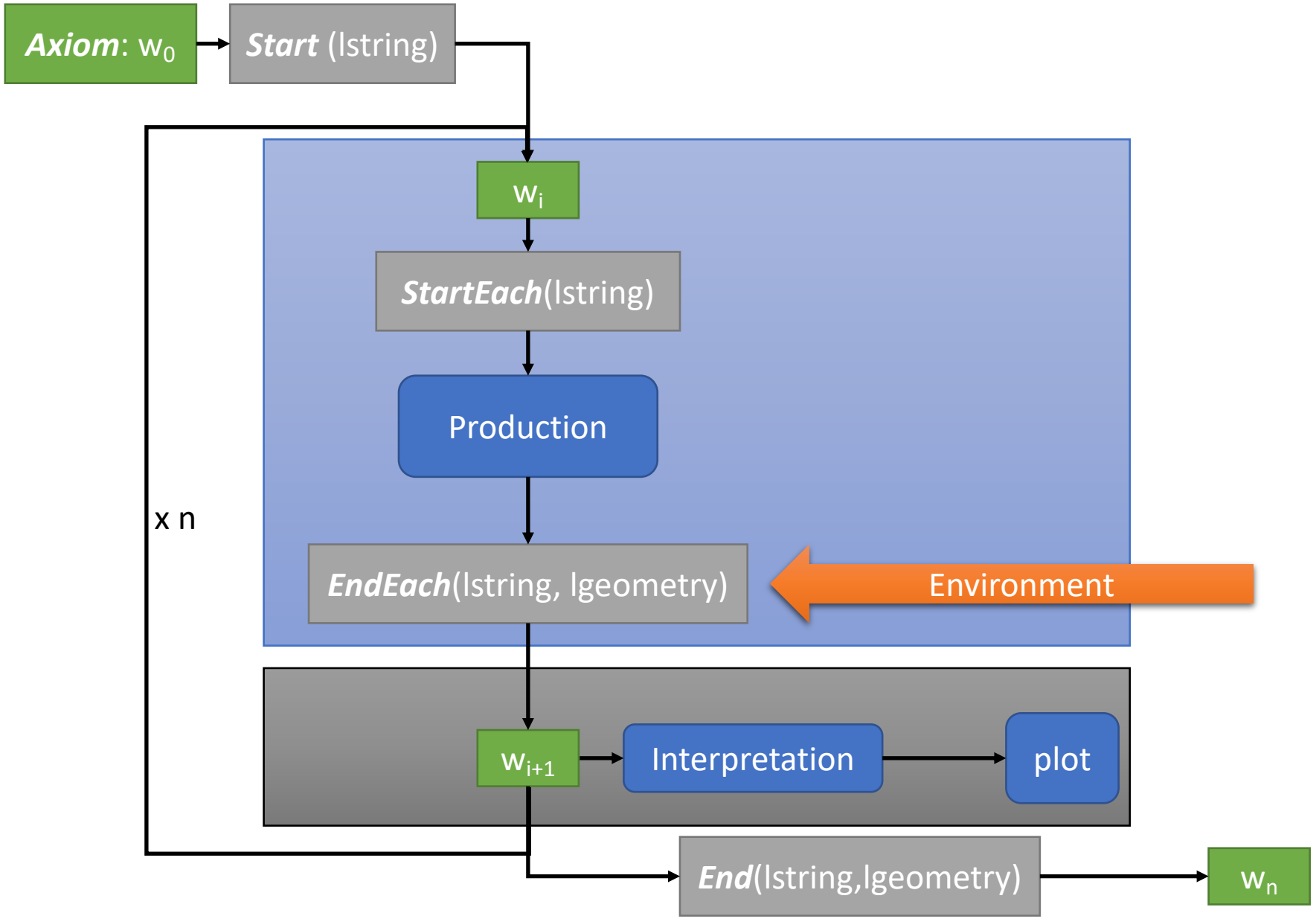
Global structure to represent environment

Request position in environment

Change value in environment



Simulation Pipeline



Global interaction with environment

- Light Interaction. Depend on the entire geometry of the plant

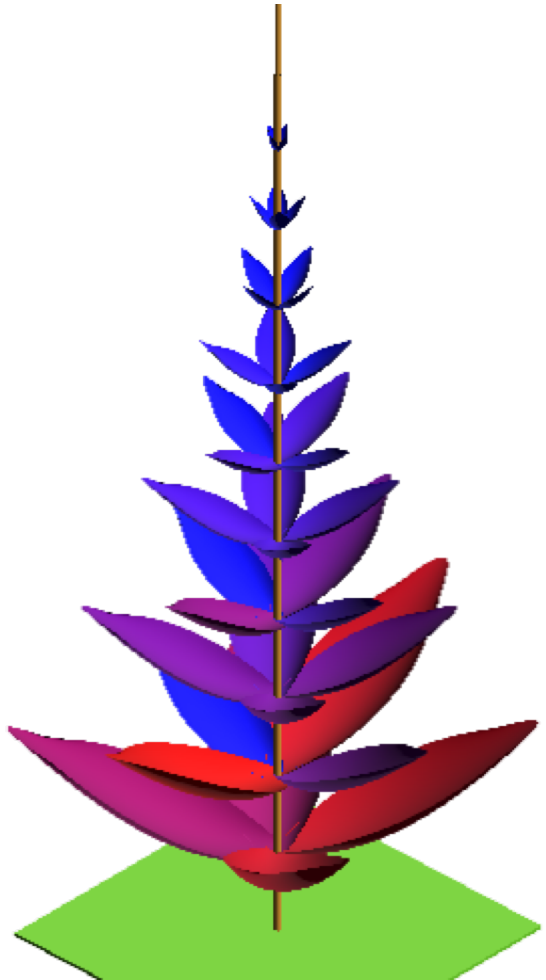
```
from openalea.fractalysis.light.directLight import diffuseInterception
module Leaf(t, carbon, light)
def EndEach(lstring,lscene):
  lightvalues = diffuseInterception(lscene)
  for modid, light in lightvalues.items():
     if lstring[modid].name == 'Leaf':
         lstring[modid].light = light
  return lstring
```

Compute intercepted light

Report light value in the string

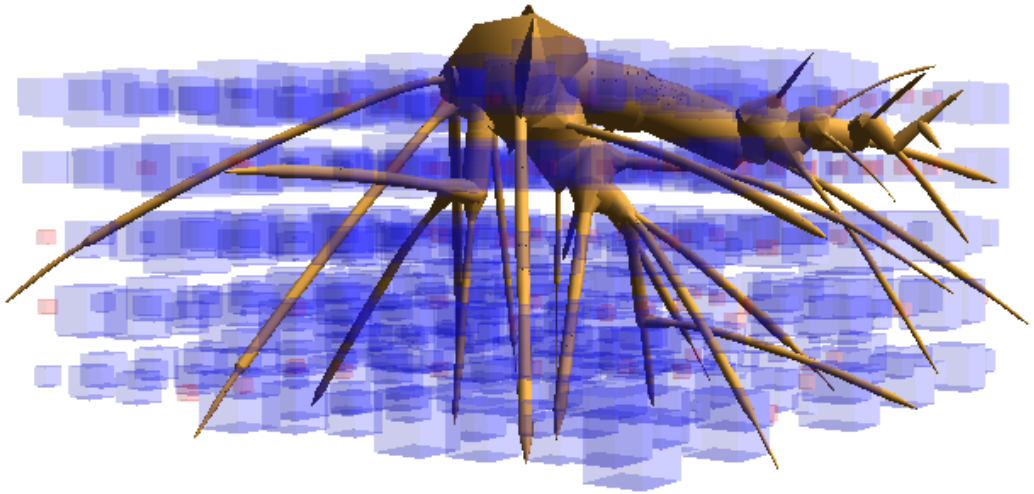
For module `modid`, set its `light` parameter to the `light` value.

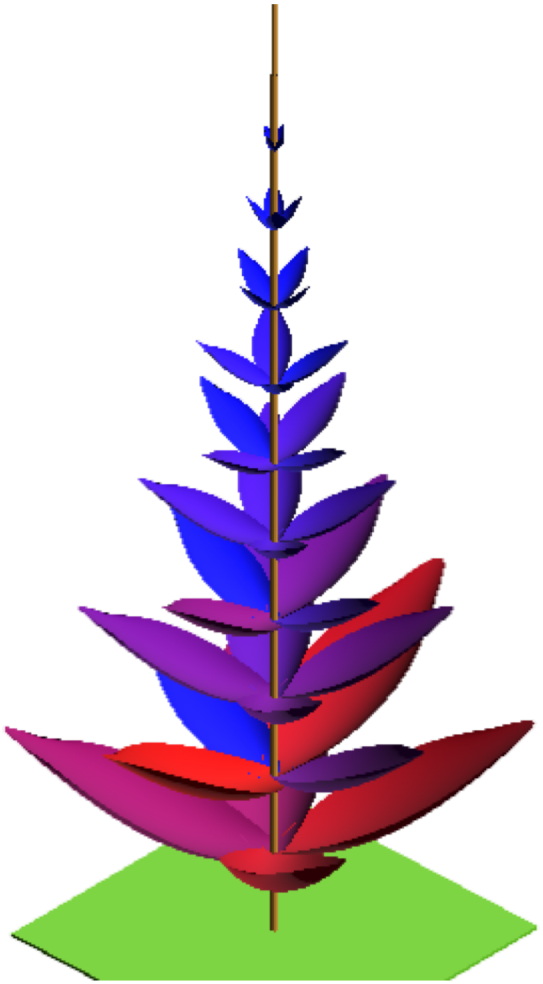
- *diffuseInterception* return of dictionary of id and cumulated value of intercepted light.
- *directInterception* is doing the same with a given set of light directions.



Exercise

- Test the two environmental models





Interactive edition based on space colonization

Thanks for you attention