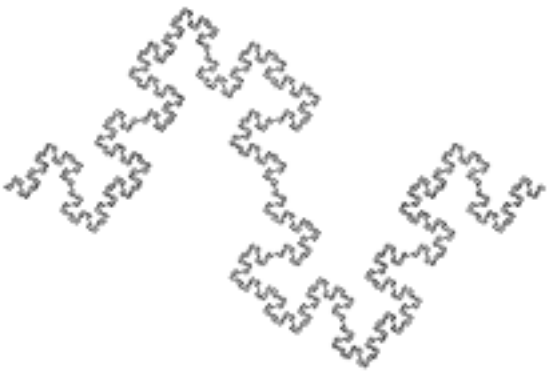
Introduction to programming with L-systems

The purpose of these notes is to introduce, through a series of examples:

- Basic components of a cpfg model: L-system file and view file,
- Basic programming constructs of the cpfg modeling language: axiom, productions, decomposition rules, and interpretation rules,
- Geometric (turtle) interpretation of L-systems,
- Basic biological notions behind the models: modules, apices, internodes, and metamers,
- Structure of developmental models expressed in plastochrons and in real time,
- Relationship between developmental and structural models.

Quadratic Koch curve

- Our first model…
- Basic L-system syntax
- Basic production syntax
- Turtle symbols F, +, -
- View file
- Color map and color index
- String output



Quadratic Koch curve is one of the fractals discovered or popularized by Mandelbrot. It provides a simple example of a **cpfg model** (object).

In the simplest case, the **L-system file** has the following overall syntax:

```
Lsystem: 1                      mandatory opening, arbitrary number
derivation length: 4            the number of derivation steps
Axiom: -F                       the axiom
F --> F+F-F-FF+F+F-F            one ( or more) productions
endlsystem                      mandatory closing
```

Parameters, such as the magnitude of turns, are read from the *view file*. In this case, it has been defined as:

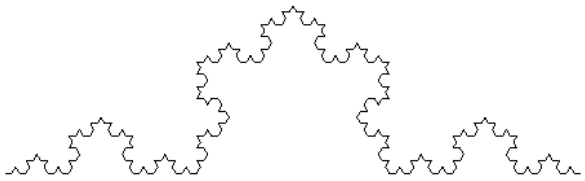
```
angle increment: 90             the turning angle, in degrees
initial line width: 1 pixels    … self-explanatory
initial color: 1                index to the color map
scale factor: 0.9000            controls the size on the screen
```

The images produced by cpfg result from the **turtle interpretation** of the generated **strings**. Sometimes (for example, for debugging purposes, when the strings are not too long) it is convenient to examine them directly. For example, the following string have been output by cpfg after 0, 1 and 2 derivation steps (spacing added for clarity):

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

Snowflake - exercise

- Your first model…
- Error messages



Fractal snowflake curve is one of the classic fractals, proposed in 1905 be Helge von Koch. You can develop its model by modifying the L-system and the view file of the quadratic Koch curve. In general, it is convenient to develop new models by modifying the existing ones.

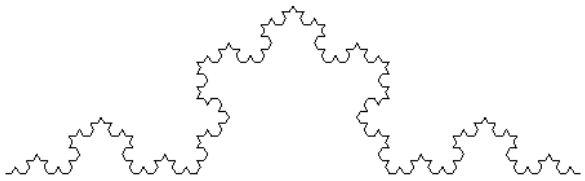
```
Lsystem: 1
derivation length: 4
Axiom: F
? --> ???                          specify new production
endlsystem
```

The turtle should no longer turn by 90 degrees. What is the new value?

```
angle increment: ?                 specify new turning angle
initial line width: 1 pixels
initial color: 1
scale factor: 0.9000
```

Snowflake - solution

- Modules (symbols with parameters)



In the previous example you have probably obtained a vertically oriented snowflake curve. To turn is around by 90 degrees , it is convenient to associate numerical **parameters** with the symbols. A symbol with parameters is called a **module**. A module may have one or more parameters. The turtle interpretation is (usually) affected by the first parameter.

```
Lsystem: 1
derivation length: 4
Axiom: -(90)F                    turn by 90 degrees
F --> F+F--F+F
endlsystem
```

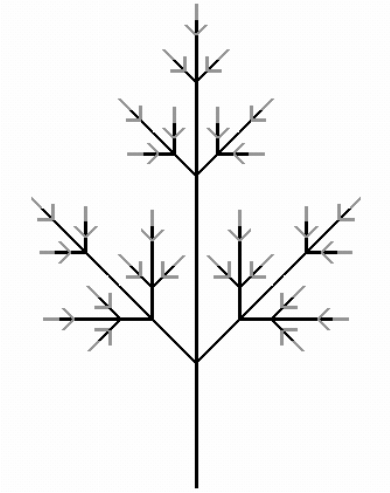
The angle increment in the view file remains in effect for symbols + and – without parameters.

```
angle increment: 60              the default turning angle
initial line width: 1 pixels
initial color: 1
scale factor: 0.9000

We could also specify other angles using parameters

Lsystem: 1
derivation length: 4
Axiom: -(90)F                    turn by 90 degrees
F --> F+(60)F-(120)F+(60)F       represent turning angles explicitly
endlsystem
```

Since all symbols + and – have a parameter associated with them, the value of the angle increment in the view file is now irrelevant.

Compound leaf + homomorphism

- Branching structure
- Apices and internodes
- Bracketed string notation
- Non-interpreted symbols
- Homomorphism
- Interpretation rules
- Changing colors
- Turtle symbols [ ] ; ,



Example of a very simple L-system generating a compound branching structure.

```
Lsystem: 1
derivation length: 4
Axiom: A
A --> F[+A][-A]FA          apex generates a branching structure
F --> FF                   internodes elongate
endlsystem
```

In the above L-system apices are not visualized, because symbols A do not have a predefined turtle interpretation. We can assign an interpretation to a symbol using the **interpretation rules**.

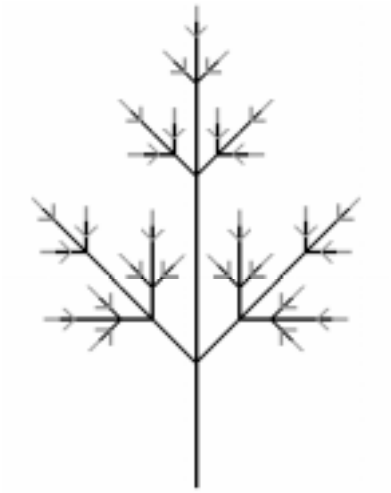
```
Lsystem: 1
derivation length: 4
Axiom: A
A --> F[+A][-A]FA
F --> FF
homomorphism               interpretation rules start here
A --> ;F                   apex is interpreted as a line of a different color
endlsystem
```

The interpretation rules make it possible to write more clear L-systems using mnemonic symbols for plant modules. For example:

```
Lsystem: 1
derivation length: 4
Axiom: A
A --> I[+A][-A]IA          mnemonic symbol I denotes an internode
I --> II
Homomorphism
I --> F                    interpretatation of (a segment of) an internode
A --> ;F                   interpretation of an apex
endlsystem
```

Compound leaf - parametric

- Parametric productions
- Actual and formal parameters
- String output by cpfg



In the previous L-system the elongation of an internode was modeled by duplicating symbols I (production `I --> II`). A better approach is to use parameters. Formal parmeters, such as x below, make it possible to assign a new parameter value as a function of the old one:

```
Lsystem: 1
derivation length: 4
Axiom: A
A --> I(1)[+A][-A]I(1)A
I(x) --> I(2*x)
homomorphism
A --> ;F
I(x) --> F(x)
endlsystem
```

Formal parameters are variables that occur in productions. Productions set **actual parameter** values of the individual modules in the string. For example, the above L-system generates the following strings after n=1, 2, and 3 derivation steps:

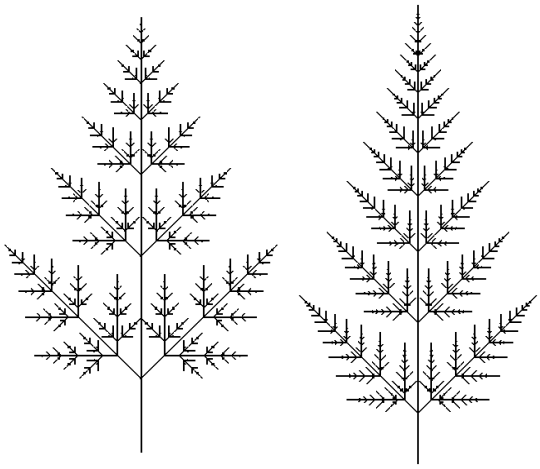
n=1
```
I(1)[+A][-A]I(1)A
```

n=2
```
I(2)
     [+I(1)[+A][-A]I(1)A]
     [-I(1)[+A][-A]I(1)A]
I(2)
     I(1)[+A][-A]I(1)A
```

n=3
```
I(4)
     [+I(2)
          [+I(1)[+A][-A]I(1)A]
          [-I(1)[+A][-A]I(1)A]
     I(2)
          I(1)[+A][-A]I(1)A]
     ...
```

Fractal fern leaves

- development with delays
- growth rate
- conditional productions
- C preprocessor
- #define statements
- comments
- splitting long productions

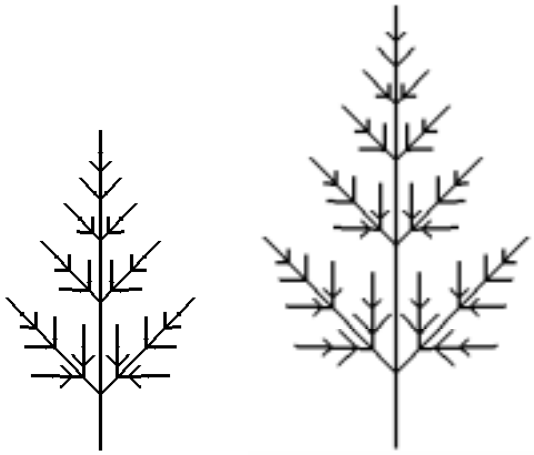


Parameters are a powerful construct, and can be used for various modeling purposes. For example, in the model below they are used to introduce **branching delay**.

```
#define STEPS 18                          definitions of constants – C syntax
#define D 3
#define R 1.28

/* Sample parameter values:               a comment
     STEPS= 4, D=0, R=2
     STEPS=11, D=1, R=1.5
     STEPS=18, D=3, R=1.28
*/

Lsystem: 1
derivation length: STEPS
Axiom: A(0)
A(t) : t<D  --> A(t+1)                    delay;  parameter used in condition
A(t) : t==D -->                           split production…
     I(1)[+A(0)][-A(0)]I(1)A(t)           …next line starts with a tab
I(x) --> I(R*x)                           exponential growth with rate R
homomorphism
A(t) --> ;F(t)
I(x) --> F(x)
endlsystem
```

Fern – continuous time

- continuous-time models
- animation of development
- statement blocks
- local variables



In the previous models, each derivation step corresponded to a **plastochron**. In contrast, the model below operates in **real time**, advanced by a (small) interval dt.
To express the model conveniently, we introduce **statement blocks** using syntax:

```
predecessor : { α } condition { β } --> successor
```

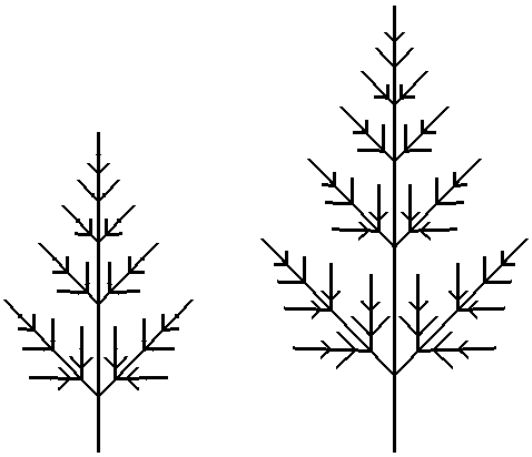
α is a block of C-like statements evaluated **before** the condition.
β is a block of C-like statements evaluated **after** the condition.
If one or both statement blocks are present, condition must be present as well (can be 1)
Statement blocks may introduce **local variables**.

```
#define STEPS 400
#define dt 0.02

#define D 2
#define R 1.5

Lsystem: 1
derivation length: STEPS
Axiom: A(0)
A(t) : {t1=t+dt;} t1<D --> A(t1)
A(t) : {t1=t+dt;}
     t1 >= D
     {t2=t1-D; t3=t1-1;} -->
     I(0.5*R^t2)[+A(t2)]
     [-A(t2)]I(0.5*R^t2)
     A(t3)
I(x) --> I(x*R^dt)
homomorphism
A(t) --> ;F(t)
I(x) --> F(x)
endlsystem
```

t1 is a local variable
statement block α
condition
statement block β

Fern - decomposition

- metamers
- decomposition
- recursion
- macro definitions



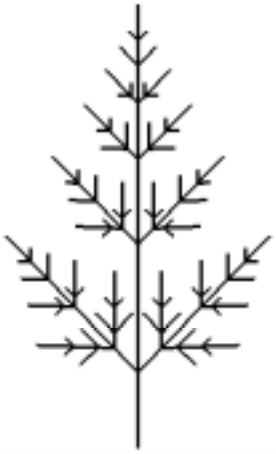
Previous model can be expressed in a more elegant manner using **decomposition rules**.

```
#define STEPS 400
#define dt 0.02

#define D 2
#define R 1.5

#define RATE (R^dt)                    macro definition of constant RATE

Lsystem: 1
derivation length: STEPS
Axiom: A(0)
A(t) --> A(t+dt)
I(x) --> I(x*RATE)
decomposition                          decomposition rules start here
maximum depth: 100                     safeguard against infinite recursion
A(t) : t >= D --> M(t-D) A(t-1)        M is a metamer
M(t) : 1 {x=0.5*R^t} -->
     I(x)[+A(t)][-A(t)]I(x)
homomorphism
A(t) --> ;F(t)
I(x) --> F(x)
endlsystem
```

Decomposition rules are evaluated recursively. Consequently, this model also works properly for dt > 1.

Recursive interpretation rules

- developmental vs. structural models
- homomorphism – maximum depth



At a limit, the entire structure at a certain time can be obtained by recursive application of decomposition rules and/or interpretation rules. This constitutes a link between developmental and **structural models**. The example below illustrates the **recursive application of interpretation rules**.

```
#define T 8

#define D 2
#define R 1.5

Lsystem: 1
derivation length: 1
Axiom: A(T)
```

no regular nor decomposition rules

```
homomorphism
maximum depth: 100
```

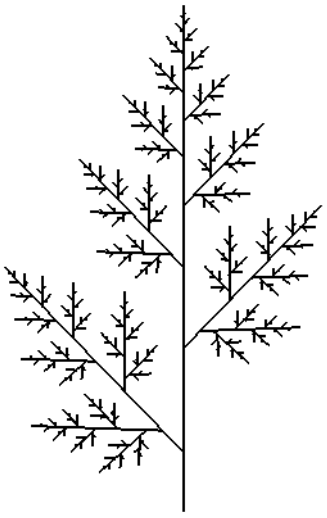
must be high enough

```
A(t) : t >= D --> M(t-D) A(t-1)

M(t) : 1 {x=0.5*R^t;} -->
     I(x)[+A(t)][-A(t)]I(x)

A(t) : t < D --> ;F(t)
I(x) --> F(x)

endlsystem
```

Models can be constructed this way only if no context-sensitive productions are used.

Alternating fern - exercise

- Alternating branching pattern
- Apical states



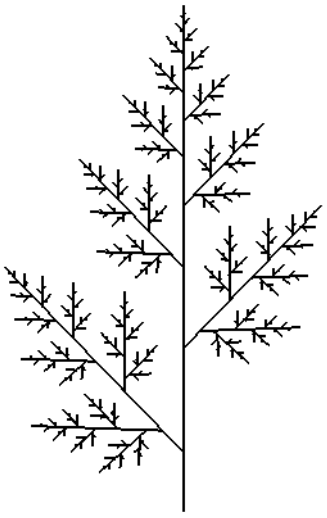
The L-system given below defines a leaf with an **opposite** branching pattern. Modify this L-system to produce a leaf with an **alternating** branching pattern.

```
#define STEPS 400
#define dt 0.02

#define D 2
#define R 1.5

#define RATE (R^dt)

Lsystem: 1
derivation length: STEPS
Axiom: A(0)
A(t) --> A(t+dt)
I(x) --> I(x*RATE)
decomposition
maximum depth: 100
A(t) : t >= D --> M(t-D) A(t-1)
M(t) : 1 {x=0.5*R^t} -->
     I(x)[+A(t)][-A(t)]I(x)
homomorphism
A(t) --> ;F(t)
I(x) --> F(x)
endlsystem
```

HINT: Introduce two different apex types A and B that produce different metamers M and N. A metamer M issues a lateral branch to the left, whereas a metamer N issues a lateral branch to the right.

Alternating fern

- Alternating branching pattern
- Apical states



A fractal leaf with an alternating branching pattern.

```
#define STEPS 800
#define dt 0.02

#define D 2.5
#define R 1.3

#define RATE (R^dt)

Lsystem: 1
derivation length: STEPS
Axiom: A(0)
A(t) --> A(t+dt)
B(t) --> B(t+dt)
I(x) --> I(x*RATE)
decomposition
maximum depth: 2
A(t) : t >= D --> M(t-D) B(t-1)      terminal apices A and B…
B(t) : t >= D --> N(t-D) A(t-1)      …alternate
M(t) : 1 {x=0.5*R^t;} -->
     I(x)[+A(t)]I(x)                 metamer M issues apex A to the left
N(t) : 1 {x=0.5*R^t;} -->
     I(x)[-B(t)]I(x)                 metamer N issues apex B to the right
homomorphism
A(t) --> ;F(t)
B(t) --> ;;F(t)
I(x) --> F(x)
endlsystem
```