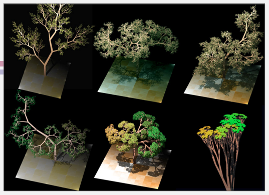
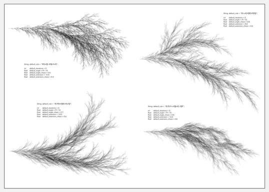
L-Systems

- a rule-rewriting system developed by botanist Aristid Lindenmeyer in 1968 to model plant growth processes
- can be used to describe fractal shapes
- components
  - an *alphabet* of symbols
  - a set of *production rules*
  - an initial *generator* string
  - a *graphical interpretation*
    - e.g. turtle interpretation





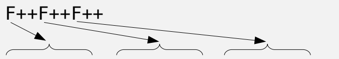
https://upload.wikimedia.org/wikipedia/commons/7/74/Dragon_trees.jpg
http://blog.kenperlin.com/?p=1453
http://www.erase.net/projects/l-systems/images/page.12.1024.jpg

L-Systems Example

- generator – F++F++F++
- production rule – F → F-F++F-F

start with the generator

apply production rules – replace each F with F-F++F-F



F++F++F++

F-F++F-F++F-F++F-F++F-F++F-F++

F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F+
+F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-
F-F-F++F-F++F-F++F-F-F-F++F-F++

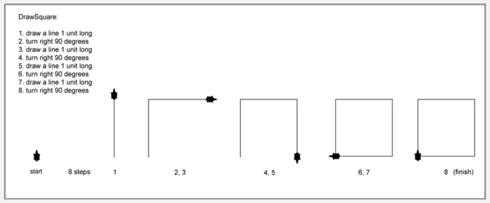
F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F+
+F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-
F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-
F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F+
+F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-F+
+F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-
F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-
F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-

this is an infinite process – typically stop after some number of levels of expansion

Turtle Interpretation of L-Systems

a turtle has a **position** and an **orientation**

it can **move forward**, **draw a line** (move forward with pen down), **turn left**, **turn right**



DrawSquare:

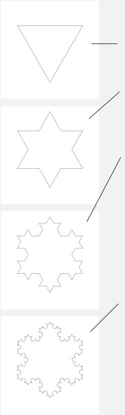
1. draw a line 1 unit long
2. turn right 90 degrees
3. draw a line 1 unit long
4. turn right 90 degrees
5. draw a line 1 unit long
6. turn right 90 degrees
7. draw a line 1 unit long
8. turn right 90 degrees

- F – move forward, drawing a line
- f – move forward (without drawing)
- + – turn right
- - – turn left
- [– save current position/orientation (push matrix)
-] – restore last-saved position/orientation (pop matrix)
- other symbols – do nothing

Turtle Interpretation

- F – move forward, drawing a line
- f – move forward (without drawing)
- + – turn right
- - – turn left
- [– save current state
-] – restore last-saved state
- other symbols – do nothing

angle: 60 degrees



F++F++F++

F-F++F-F++F-F++F-F++F-F++F-F++

F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-
F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-
F++F-F-F-F++F-F++

F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-
F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-
F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-F-F-F+
+F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F+
+F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F+
+F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F+
+F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F+
+F-F-F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F+
+F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-
F++F-F++F-F-F-F++F-F++

Implementing L-Systems

Steps –

- create a drawing function for the F production rule
- create a drawing function for each of the other production rules, if any
- create a drawing function for the whole fractal
- call the whole-fractal drawing function to actually draw the fractal

Implementing L-Systems – Pattern

- the F production rule becomes a drawing function

```
// production rule: F -> …

void drawF ( int depth, float len ) {
  if ( depth == 0 ) {
    // 'F' means draw line, move turtle
    line(0, 0, len, 0);
    translate(len, 0);
  } else {
    do what the right side of the rule states: …
  }
}
```

coordinates are interpreted relative to the turtle so this will draw a line in front of the turtle rather than in the upper left corner of the window

- reference to a production symbol → call to that function (depth decreases by 1, len may be adjusted by fractal's scale factor)
- +, - → rotate(*a*) or rotate(-*a*) (angle *a* depends on the fractal)
- [,] → pushMatrix(), popMatrix()

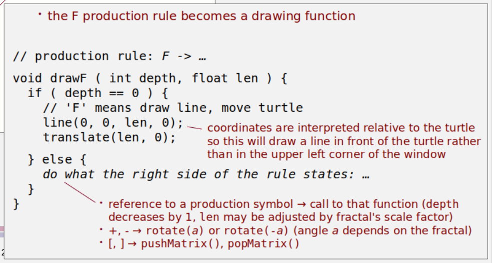
Example

```
// production rule: F -> F-F++F-F
// angle: 60 degrees, scale factor 1/3
// depth is the number of levels of recursion remaining
// len is the length of the line to draw
void drawF ( int depth, float len ) {
  if ( depth == 0 ) {
    // do the 'F' action - draw line, move turtle
    line(0, 0, len, 0);
    translate(len, 0);
  } else {
    // otherwise, do what the rule states: F-F++F-F
    drawF(depth-1, len/3);
    rotate(radians(-60));
    drawF(depth-1, len/3);
    rotate(radians(60));
    rotate(radians(60));
    drawF(depth-1, len/3);
    rotate(radians(-60));
    drawF(depth-1, len/3);
  }
}
```

DIVISION WARNING! Writing (1/3)*len, while mathematically correct, will result in 0! If you want division to include decimal points, use floats – (1.0/3.0)*len

scale factor 1/3 means len/3
rotation angle 60 degrees

depth decreases by 1


- the F production rule becomes a drawing function

```
// production rule: F -> …

void drawF ( int depth, float len ) {
  if ( depth == 0 ) {
    // 'F' means draw line, move turtle
    line(0, 0, len, 0);
    translate(len, 0);
  } else {
    do what the right side of the rule states: …
  }
}
```

coordinates are interpreted relative to the turtle so this will draw a line in front of the turtle rather than in the upper left corner of the window

- reference to a production symbol → call to that function (depth decreases by 1, len may be adjusted by fractal's scale factor)
- +, - → rotate(*a*) or rotate(-*a*) (angle *a* depends on the fractal)
- [,] → pushMatrix(), popMatrix()

Implementing L-Systems – Pattern

- any other production rules also become drawing functions

```
// production rule: S → …

void drawS ( int depth, float len ) {
  if ( depth == 0 ) {
    do the action for symbol S (may be nothing)
  } else {
    do what the right side of the rule states: …
  }
}
```

- reference to a production symbol → call to that function (depth decreases by 1, len may be adjusted by fractal's scale factor)
- +, - → rotate(*a*) or rotate(-*a*) (angle *a* depends on the fractal)
- [,] → pushMatrix(), popMatrix()

Implementing L-Systems – Pattern

- create a drawing function for the whole pattern
- parameters for position, size, and maximum depth

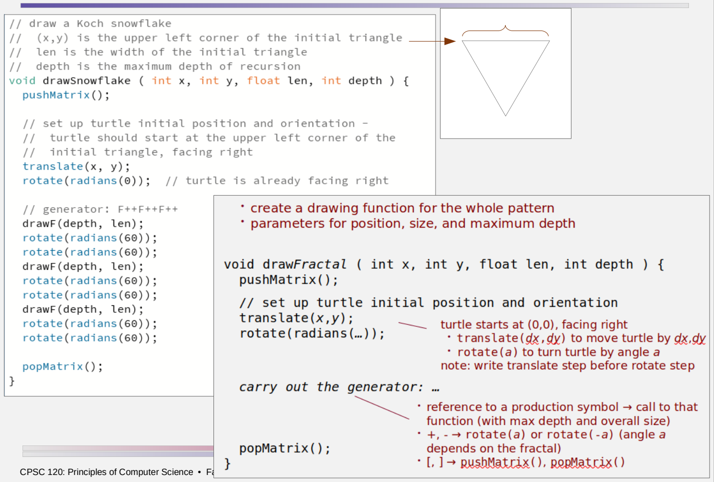
```
void drawFractal ( int x, int y, float len, int depth ) {
  pushMatrix();

  // set up turtle initial position and orientation
  translate(x,y);
  rotate(radians(…));

  carry out the generator: …

  popMatrix();
}
```

turtle starts at (0,0), facing right
- translate(*dx*,*dy*) to move turtle by *dx*,*dy*
- rotate(*a*) to turn turtle by angle *a*

note: write translate step before rotate step

- reference to a production symbol → call to that function (with max depth and overall size)
- +, - → rotate(*a*) or rotate(-*a*) (angle *a* depends on the fractal)
- [,] → pushMatrix(), popMatrix()

Example



```
// draw a Koch snowflake
//  (x,y) is the upper left corner of the initial triangle
//  len is the width of the initial triangle
//  depth is the maximum depth of recursion
void drawSnowflake ( int x, int y, float len, int depth ) {
  pushMatrix();

  // set up turtle initial position and orientation –
  //  turtle should start at the upper left corner of the
  //  initial triangle, facing right
  translate(x, y);
  rotate(radians(0));  // turtle is already facing right

  // generator: F++F++F++
  drawF(depth, len);
  rotate(radians(60));
  rotate(radians(60));
  drawF(depth, len);
  rotate(radians(60));
  rotate(radians(60));
  drawF(depth, len);
  rotate(radians(60));
  rotate(radians(60));

  popMatrix();
}
```

- create a drawing function for the whole pattern
- parameters for position, size, and maximum depth

```
void drawFractal ( int x, int y, float len, int depth ) {
  pushMatrix();

  // set up turtle initial position and orientation
  translate(x,y);
  rotate(radians(…));

  carry out the generator: …

  popMatrix();
}
```

turtle starts at (0,0), facing right
- translate(*dx*,*dy*) to move turtle by *dx*,*dy*
- rotate(*a*) to turn turtle by angle *a*

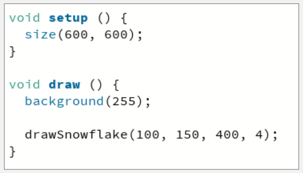
note: write translate step before rotate step

- reference to a production symbol → call to that function (with max depth and overall size)
- +, - → rotate(*a*) or rotate(-*a*) (angle *a* depends on the fractal)
- [,] → pushMatrix(), popMatrix()

Implementing L-Systems – Pattern & Example

- call the whole-fractal drawing function to actually draw the fractal

```
void setup () {
  …
}

void draw () {
  …
  drawFractal(…);
  …
}
```



```
void setup () {
  size(600, 600);
}

void draw () {
  background(255);

  drawSnowflake(100, 150, 400, 4);
}
```