typst Package

fractusist

User's Manual

Version **0.3.1**
April 2025

Guangxi Liu

Contents

1 Introduction

The package `fractusist`[1] creates a variety of wonderful fractals and curves in Typst. It has the following features:

- Generate fractals using L-system. The grammar, number of iterations, drawing styles, etc. could be customized.
- Over 30 preset parameters are provided for the L-system to facilitate the drawing of fractals.
- Generate fractals using iterative methods, including Fibonacci word fractal and Z-order curve.
- Generate fractals using recursive methods, including various fractal trees and Sierpiński carpet.
- Generate parametric curves, such as spirographs and Lissajous curves.

To use it, import the latest version of this package with:

```
#import "@preview/fractusist:0.3.1": *
```

This line will be omitted in the examples codes that follows.

Each drawing function generates a type of fractal or curve, with a variety of configurable parameters. And the fill and stroke style arguments are equivalent to those in the `curve` function[2]. The returned graph is contained within the `box` element.

In the following sections, the use of the corresponding drawing functions are described in detail depending on the generation method.

[1] https://typst.app/universe/package/fractusist
[2] https://typst.app/docs/reference/visualize/curve

2 L-system Fractal

2.1 Guide

An L-system[3] or Lindenmayer system is a parallel rewriting system and a type of formal grammar. An L-system consists of an alphabet of symbols that can be used to make strings, a collection of production rules that expand each symbol into some larger string of symbols, an initial "axiom" string from which to begin construction, and a mechanism for translating the generated strings into geometric structures.

When implementing drawing, use the following parameters:

- **variables** — a set of symbols containing elements that can be replaced (e.g. `F`)
- **constants** — a set of symbols containing elements that cannot be replaced (e.g. `+`, `-`)
- **axiom** — a string of symbols from above variables or constants defining the initial state of the system (e.g. `F`)
- **rules** — the way variables can be replaced with combinations of constants and other variables (e.g. `F` → `F-F++F-F`)

The rules of the L-system grammar are applied iteratively starting from the initial state. As many rules as possible are applied simultaneously, per iteration. The symbols in the resulting string are then parsed for drawing.

Currently the following symbols and corresponding drawing directives are supported (all other symbols are ignored):

- *Sd* — move forward by line length drawing a line, here *Sd* is a set of preset symbols
- *Sm* — move forward by line length without drawing a line, here *Sm* is a set of preset symbols
- `+` — turn left by turning angle
- `-` — turn right by turning angle
- `|` — reverse direction (i.e. turn by 180 degrees)
- `[` — save the current state (i.e. the position and direction)
- `]` — restore the last saved state

Here are the internal details inside the codes:

1. Generate string iteratively according the rules of the specific L-system.
2. Dynamically update vertex coordinates and the entire shape bounding box by parsing each symbol in generated string. At the same time, generate the corresponding drawing commands.
3. Assemble the drawing commands into the final `curve` function call and return it within the `box` object.

2.2 Reference

2.2.1 `dragon-curve`

Generate dragon curve[4].

[3] https://en.wikipedia.org/wiki/L-system
[4] https://en.wikipedia.org/wiki/Dragon_curve

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
dragon-curve(
  int,
  step-size: int float,
  stroke: stroke,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 16.

step-size `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the `box` element.

2.2.2 `hilbert-curve`

Generate 2D Hilbert curve[5].

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
hilbert-curve(
  int,
  step-size: int float,
  stroke: stroke,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 1 to 8.

step-size `int` or `float` *Settable*

[5] https://en.wikipedia.org/wiki/Hilbert_curve

The step size (in pt). Must be positive.

Default: `10`

`stroke` `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

`graph` `content` *Returned*

Returned graph, contained within the box element.

2.2.3 `peano-curve`

Generate 2D Peano curve (Hilbert II curve)[6].

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
peano-curve(
  int,
  step-size: int float,
  stroke: stroke,
) -> content
```

`n` `int` *Required* *Positional*

The number of iterations. Valid range is 1 to 5.

`step-size` `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

`stroke` `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

`graph` `content` *Returned*

Returned graph, contained within the box element.

[6] https://en.wikipedia.org/wiki/Peano_curve

2.2.4 `koch-curve`

Generate Koch curve.

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
koch-curve(
  int,
  step-size: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

`n` `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 6.

`step-size` `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

`fill` `fill` *Settable*

How to fill the curve.

Default: `none`

`stroke` `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

`graph` `content` *Returned*

Returned graph, contained within the `box` element.

2.2.5 `koch-snowflake`

Generate Koch snowflake[7].

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
koch-snowflake(
  int,
```

[7] https://en.wikipedia.org/wiki/Koch_snowflake

```
  step-size: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 6.

step-size `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

fill `fill` *Settable*

How to fill the curve.

Default: `none`

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the `box` element.

2.2.6 `sierpinski-curve`

Generate classic Sierpiński curve[8].

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
sierpinski-curve(
  int,
  step-size: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 7.

[8] https://en.wikipedia.org/wiki/Sierpi%C5%84ski_curve

step-size `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

fill `fill` *Settable*

How to fill the curve.

Default: none

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the box element.

2.2.7 `sierpinski-square-curve`

Generate Sierpiński square curve.

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
sierpinski-square-curve(
  int,
  step-size: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 7.

step-size `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

fill `fill` *Settable*

How to fill the curve.

Default: none

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the box element.

2.2.8 `sierpinski-arrowhead-curve`

Generate Sierpiński arrowhead curve.

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
sierpinski-arrowhead-curve(
  int,
  step-size: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 8.

step-size `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

fill `fill` *Settable*

How to fill the curve.

Default: `none`

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the box element.

2.2.9 `sierpinski-triangle`

Generate 2D Sierpiński triangle[9].

Note: This function has been superseded by `lsystem` and is only reserved for compatibility.

Parameters

```
sierpinski-triangle(
  int,
  step-size: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

`n` `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 6.

`step-size` `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

`fill` `fill` *Settable*

How to fill the curve.

Default: `none`

`stroke` `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

`graph` `content` *Returned*

Returned graph, contained within the `box` element.

2.2.10 `lsystem-names`

Get all names in L-system generator library.

Currently L-system generator library defines the parameters for the following fractals (cover all previous individual ones):

[9] https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle

Table 1 Fracal names in L-system generator library

Board	Cantor Set
Cesero fractal	Cross
Crystal	Dragon Curve
Fern 1	Fern 2
Fern 3	Fern 4
Gosper Curve	Hilbert Curve
Koch Curve	Koch Snowflake
Kolam	Levy Curve
Mango Leaf	McWorter Dendrite Fractal
Moore Curve	Peano Curve
Penrose Tiling	Pentaplexity
Quadratic Snowflake	Rectangle Island Curve
Rings	Rounded Peano Curve
Sierpinski Arrowhead Curve	Sierpinski Curve
Sierpinski Hexagon	Sierpinski Square Curve
Sierpinski Triangle	Smoother Peano Curve
Snake Kolam	

Parameters

```
lsystem-names -> array
```

names `array` *Returned*

Returned array of fractal names (type `str`).

2.2.11 `lsystem-use`

Get parameters in L-system generator library by name (see Table 1).

Parameters

```
lsystem-use(
  str,
) -> dictionary
```

name `str` *Required* *Positional*

The name in L-system generator library. The valid name here is taken from the array returned by `lsystem-names` function.

parameters `dictionary` *Returned*

Returned parameters set (type `dictionary`) for `lsystem` function.

It contains the following fields:
- `draw-forward-sym` `str`
- `move-forward-sym` `str`
- `axiom` `str`
- `rule-set` `dictionary`
- `angle` `int` `float`
- `cycle` `bool`

2.2.12 `lsystem`

General L-system generator. The rules of the L-system grammar, graph shape parameters and fill/stroke styles could be specified completely.

Internally, the length limit of the generated string after iteration is set to 5000000, but it may be relaxed in future versions.

Parameters

```
lsystem(
  draw-forward-sym: str,
  move-forward-sym: str,
  axiom: str,
  rule-set: dictionary,
  angle: float,
  cycle: bool,
  order: int,
  step-size: int float,
  start-angle: int float,
  padding: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

`draw-forward-sym` `str` *Settable*

The symbol set for moving forward by line length drawing a line.

Default: `"F"`

`move-forward-sym` `str` *Settable*

The symbol set for moving forward by line length without drawing a line.

Default: `""`

`axiom` `str` *Settable*

The starting string.

Default: `"F"`

`rule-set` `dictionary` *Settable*

The rewrite rule (type `dictionary`). Each key-value pair corresponds to a rule.

Default: `("F": "F-F++F-F")`

`angle` `float` *Settable*

The turning angle (in π radius). Valid range is $(0, 1)$.

Default: `1/3`

`cycle` `bool` *Settable*

Whether close the curve. `true` is close the curve.

Default: `false`

`order` `int` *Settable*

The number of iterations. Must be non-negative.

Note: The maximum value is limited by the actual length of the string after iteration.

Default: `3`

`step-size` `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

`start-angle` `int` or `float` *Settable*

The starting angle of direction (in π radius). Valid range is $[0, 2)$.

Default: `1`

`padding` `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

`fill` `fill` *Settable*

How to fill the curve.

Default: `none`

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

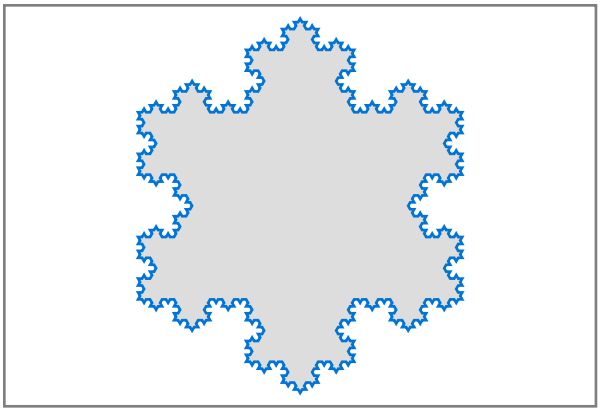
Returned graph, contained within the box element.

2.3 Examples

2.3.1 Koch Snowflake

A Koch snowflake using the function `koch-snowflake`.

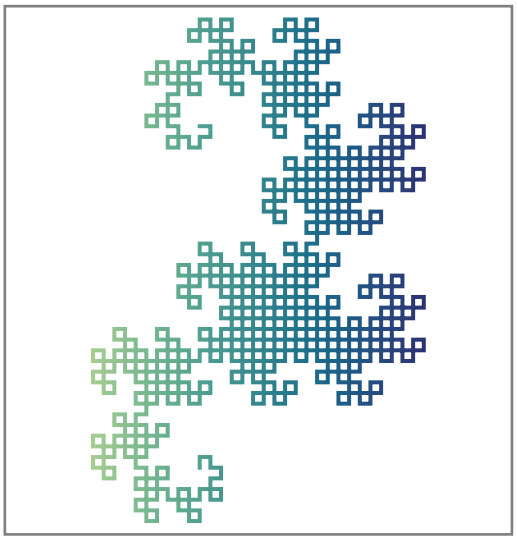
```
#koch-snowflake(
  4,
  step-size: 1.5,
  fill: silver,
  stroke: blue
)
```



2.3.2 Dragon curve

A dragon curve using the function `dragon-curve`.

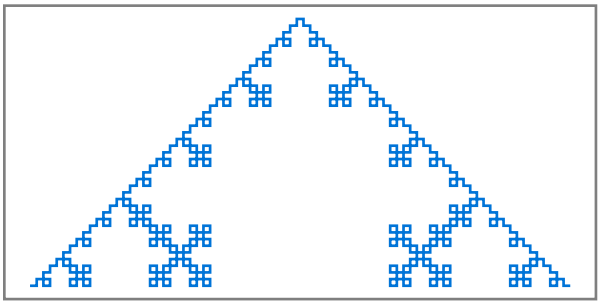
```
#dragon-curve(
  10,
  step-size: 4,
  stroke: stroke(
    paint: gradient.linear(..color.map.crest),
    thickness: 1.5pt,
    cap: "square"
  )
)
```



2.3.3 General L-system Fractals

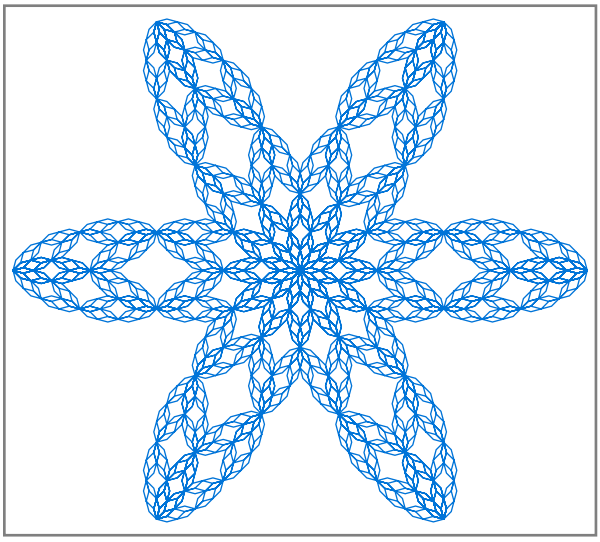
A variant of the Koch curve which uses only right angles. All parameters in function are customized.

```
#lsystem(
  draw-forward-sym: "F",
  axiom: "F",
  rule-set: ("F": "F+F-F-F+F"),
  angle: 1/2,
  cycle: false,
  order: 4,
  step-size: 2.5,
  start-angle: 0,
  stroke: blue
)
```



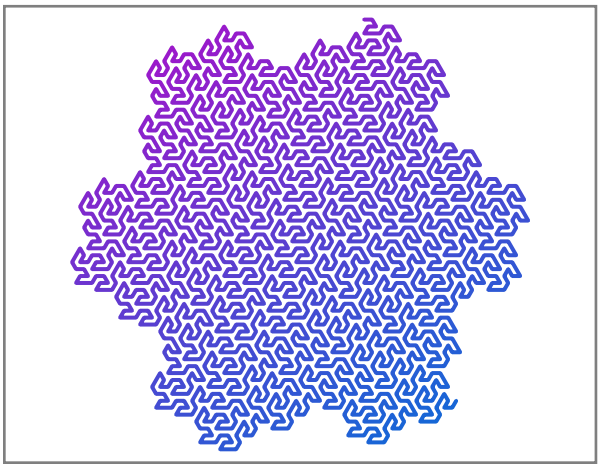
Another snowflake pattern uses custom parameters.

```
#lsystem(
  draw-forward-sym: "F",
  axiom: "[F]++++[F]++++[F]++++[F]++++[F]+
+++F",
  rule-set: ("F": "[+F--F]-F++F-"),
  angle: 1/12,
  cycle: false,
  order: 5,
  step-size: 4,
  start-angle: 0,
  stroke: stroke(
    paint: blue,
    thickness: 0.5pt,
    cap: "round",
    join: "round"
  )
)
```



A Gosper curve[10] using preset L-system grammar parameters.

```
#lsystem(
  ..lsystem-use("Gosper Curve"),
  order: 4,
  step-size: 3,
  start-angle: 0,
  stroke: stroke(
    paint: gradient.linear(purple, blue,
angle: 60deg),
    thickness: 1.5pt,
    cap: "round",
    join: "round"
  )
)
```



[10] https://en.wikipedia.org/wiki/Gosper_curve

3 Iterative Method Fractal

3.1 Guide

The iterative method fractal is internally implemented similarly to L-system.

Based on a specific algorithm and given parameters, iteratively generate a sequence of drawing instructions. Then parse the sequence to obtain the final Typst drawing primitive functions.

3.2 Reference

3.2.1 `fibonacci-word-fractal`

Generate Fibonacci word fractal[11].

Parameters

```
fibonacci-word-fractal(
  int,
  skip-last: bool,
  step-size: int float,
  start-dir: int,
  padding: int float,
  stroke: stroke,
) -> content
```

`n` `int` *Required* *Positional*

The number of iterations. Valid range is 3 to 24.

`skip-last` `bool` *Settable*

Whether skip the last symbol (Fibonacci word fractal becomes more symmetrical). `false` is not skip the last symbol.

Default: `true`

`step-size` `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

`start-dir` `int` *Settable*

Starting direction (0: right, 1: up, 2: left, 3: down).

Default: `0`

[11] https://en.wikipedia.org/wiki/Fibonacci_word_fractal

padding `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the box element.

3.2.2 `z-order-curve`

Generate Z-order curve[12].

Parameters

```
z-order-curve(
  int,
  step-size: int float,
  start-dir: int,
  padding: int float,
  stroke: stroke,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 1 to 8.

step-size `int` or `float` *Settable*

The step size (in pt). Must be positive.

Default: `10`

start-dir `int` *Settable*

Starting direction (0: horizontal, 1: vertical).

Default: `0`

padding `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

[12] https://en.wikipedia.org/wiki/Z-order_curve

Default: `0`

`stroke` `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

`graph` `content` *Returned*

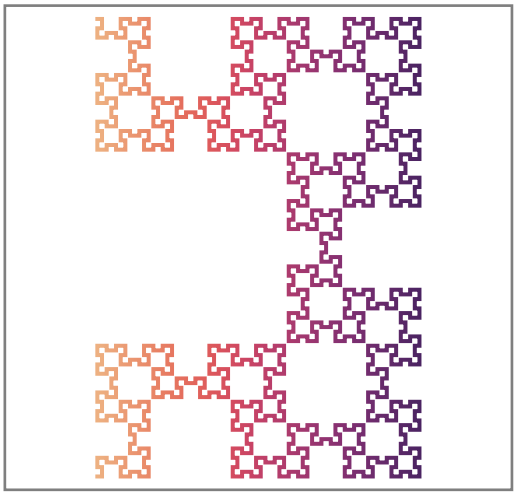
Returned graph, contained within the box element.

3.3 Examples

3.3.1 Fibonacci Fractal

A 17th order Fibonacci word fractal.

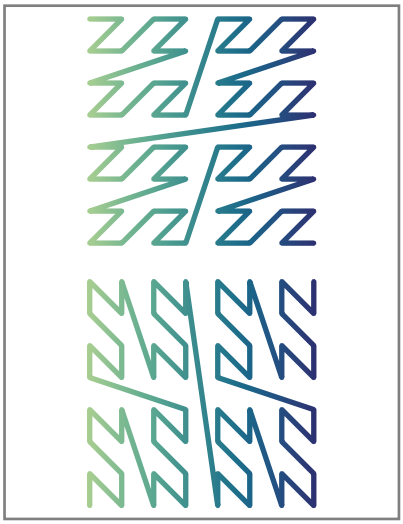
```
#fibonacci-word-fractal(
  17,
  step-size: 1.75,
  stroke: stroke(
    paint: gradient.linear(..color.map.flare),
    thickness: 1.5pt,
    cap: "square"
  )
)
```



3.3.2 Z-order Curves

Two Z-order curves with different orientations (`Z`-shape and `N`-shape).

```
#let stroke-stype = stroke(
  paint: gradient.linear(..color.map.crest, angle: 0deg),
  thickness: 2pt,
  cap: "round", join: "round"
)

#z-order-curve(
  3,
  step-size: 12, start-dir: 0, stroke: stroke-stype
)

#z-order-curve(
  3,
  step-size: 12, start-dir: 1, stroke: stroke-stype
)
```



4 Recursive Method Fractal

4.1 Guide

The recursive method fractal takes advantage of the self-similarity of the graph itself.

Based on a specific algorithm and given parameters, recursively generate a sequence of drawing instructions. Then parse the sequence to obtain the final Typst drawing primitive functions.

4.2 Reference

4.2.1 `sierpinski-carpet`

Generate Sierpiński carpet[13].

Parameters

```
sierpinski-carpet(
  int,
  size: int float,
  padding: int float,
  fill: fill,
  stroke: stroke,
) -> content
```

`n` `int` *Required* *Positional*

The number of iterations. Valid range is 0 to 5.

`size` `int` or `float` *Settable*

The width/height of the image (in pt). Must be positive.

Default: `243`

`padding` `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

`fill` `fill` *Settable*

How to fill the curve.

Default: `none`

`stroke` `stroke` *Settable*

[13] https://en.wikipedia.org/wiki/Sierpi%C5%84ski_carpet

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the box element.

4.2.2 `fractal-tree`

Generate fractal tree. The thickness and color of the branches vary with the level.

Parameters

```
fractal-tree(
  int,
  root-color: color,
  leaf-color: color,
  trunk-len: int float,
  trunk-rad: int float,
  theta: int float,
  angle: int float,
  ratio: float,
  padding: int float,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 1 to 14.

root-color `color` *Settable*

The root branch color.

Default: `rgb("#46230A")`

leaf-color `color` *Settable*

The leaf color.

Default: `rgb("#228B22")`

trunk-len `int` or `float` *Settable*

The initial length of the trunk (in pt). Must be positive.

Default: `100`

trunk-rad `int` or `float` *Settable*

The initial radius of the trunk (in pt). Must be positive.

Default: `3.0`

`theta` `int` or `float` *Settable*

The initial angle of the branch (in π radius). Valid range is $[0, 1]$.

Default: `1/2`

`angle` `int` or `float` *Settable*

The angle between branches in the same level (in π radius). Valid range is $[0, 1/2]$.

Default: `1/4`

`ratio` `float` *Settable*

The contraction factor between successive trunks. Valid range is $(0, 1)$.

Default: `0.8`

`padding` `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

`graph` `content` *Returned*

Returned graph, contained within the `box` element.

4.2.3 `random-fractal-tree`

Generate random fractal tree. The thickness and color of the branches vary with the level. And the direction of the branches is random.

Note: This function uses the package `suiji`[14] internally.

Parameters

```
random-fractal-tree(
  int,
  seed: int,
  root-color: color,
  leaf-color: color,
  trunk-len: int float,
  trunk-rad: int float,
  theta: int float,
```

[14] https://typst.app/universe/package/suiji

```
  angle: int float,
  ratio: float,
  padding: int float,
) -> content
```

n `int` *Required* *Positional*

The number of iterations. Valid range is 1 to 14.

seed `int` *Settable*

The value of seed, effective value is an integer from $[0, 2^{32} - 1]$

Default: `42`

root-color `color` *Settable*

The root branch color.

Default: `rgb("#46230A")`

leaf-color `color` *Settable*

The leaf color.

Default: `rgb("#228B22")`

trunk-len `int` or `float` *Settable*

The initial length of the trunk (in pt). Must be positive.

Default: `100`

trunk-rad `int` or `float` *Settable*

The initial radius of the trunk (in pt). Must be positive.

Default: `3.0`

theta `int` or `float` *Settable*

The initial angle of the branch (in π radius). Valid range is $[0, 1]$.

Default: `1/2`

angle `int` or `float` *Settable*

The angle between branches in the same level (in π radius). Valid range is $[0, 1/2]$.

Default: `1/4`

`ratio` `float` *Settable*

The contraction factor between successive trunks. Valid range is $(0, 1)$.

Default: `0.8`

`padding` `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

`graph` `content` *Returned*

Returned graph, contained within the box element.

4.2.4 `pythagorean-tree`

Generate Pythagorean tree[15]. The color of the branches vary with the level.

Parameters

```
pythagorean-tree(
  int,
  root-color: color,
  leaf-color: color,
  trunk-len: int float,
  theta: float,
  start-angle: int float,
  padding: int float,
  filling: bool,
) -> content
```

`n` `int` *Required* *Positional*

The number of iterations. Valid range is 1 to 14.

`root-color` `color` *Settable*

The root branch color.

Default: `rgb("#46230A")`

`leaf-color` `color` *Settable*

The leaf color.

Default: `rgb("#228B22")`

[15] https://mathworld.wolfram.com/PythagorasTree.html

`trunk-len` `int` or `float` *Settable*

The initial length of the trunk (in pt). Must be positive.

Default: `50`

`theta` `float` *Settable*

The initial angle of the branch (in π radius). Valid range is $(0, 12)$.

Default: `1/5`

`start-angle` `int` or `float` *Settable*

The starting angle of base square bottom edge direction (in π radius). Valid range is $[0, 2)$.

Default: `100`

`padding` `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

`filling` `bool` *Settable*

Whether the drawing is filling. `false` is wireframe.

Default: `true`

`graph` `content` *Returned*

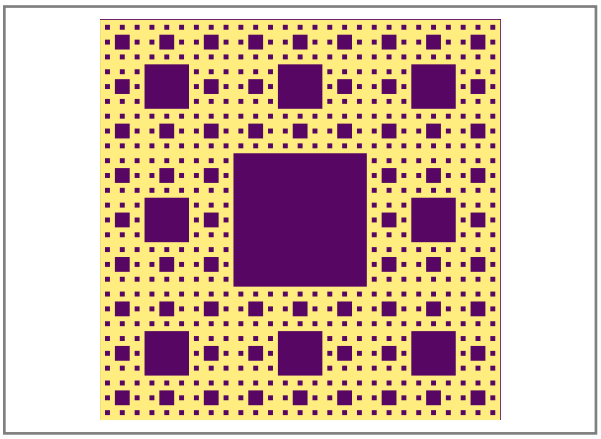
Returned graph, contained within the box element.

4.3 Examples

4.3.1 Sierpiński carpet

A Sierpiński carpet with different background and forground colors.

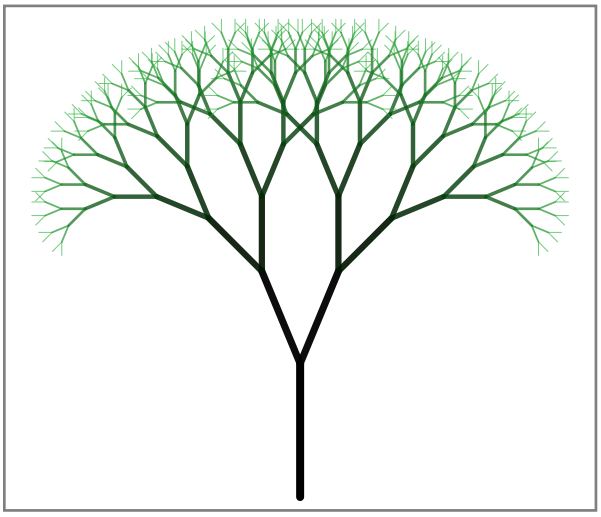
```
#box(fill: purple.darken(50%),
  sierpinski-carpet(
    4,
    size: 150,
    fill: yellow.lighten(50%),
    stroke: none
  )
)
```



4.3.2 Fractal Trees

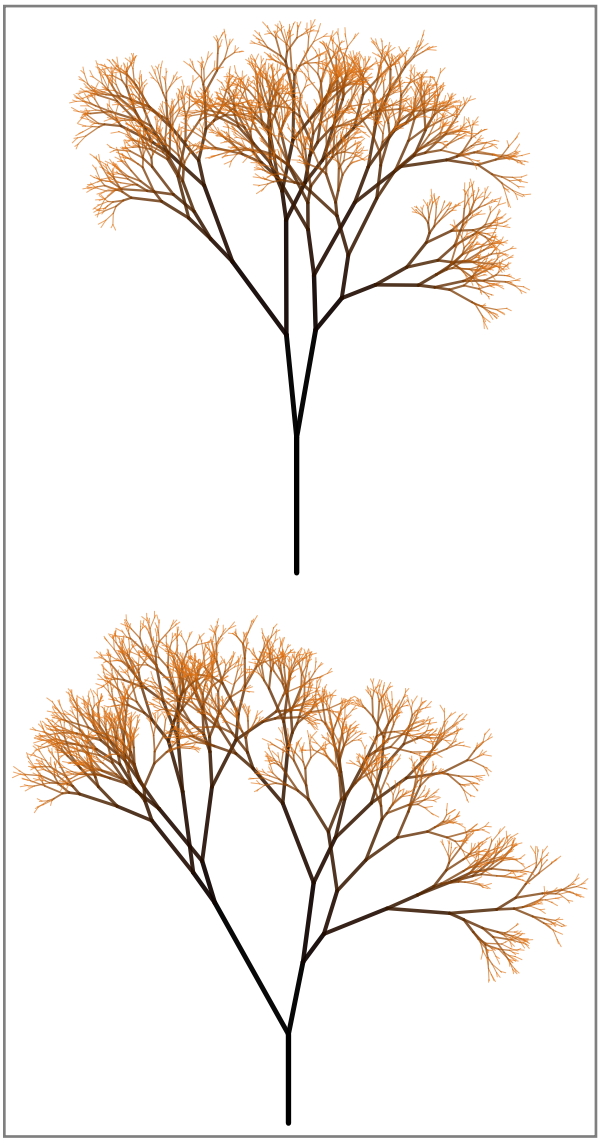
A fractal tree.

```
#fractal-tree(
  9,
  root-color: black,
  leaf-color: green.transparentize(40%),
  trunk-len: 50,
  trunk-rad: 3.0,
  angle: 1/8,
  ratio: 0.75
)
```



Two random fractal trees, only the seeds are different.

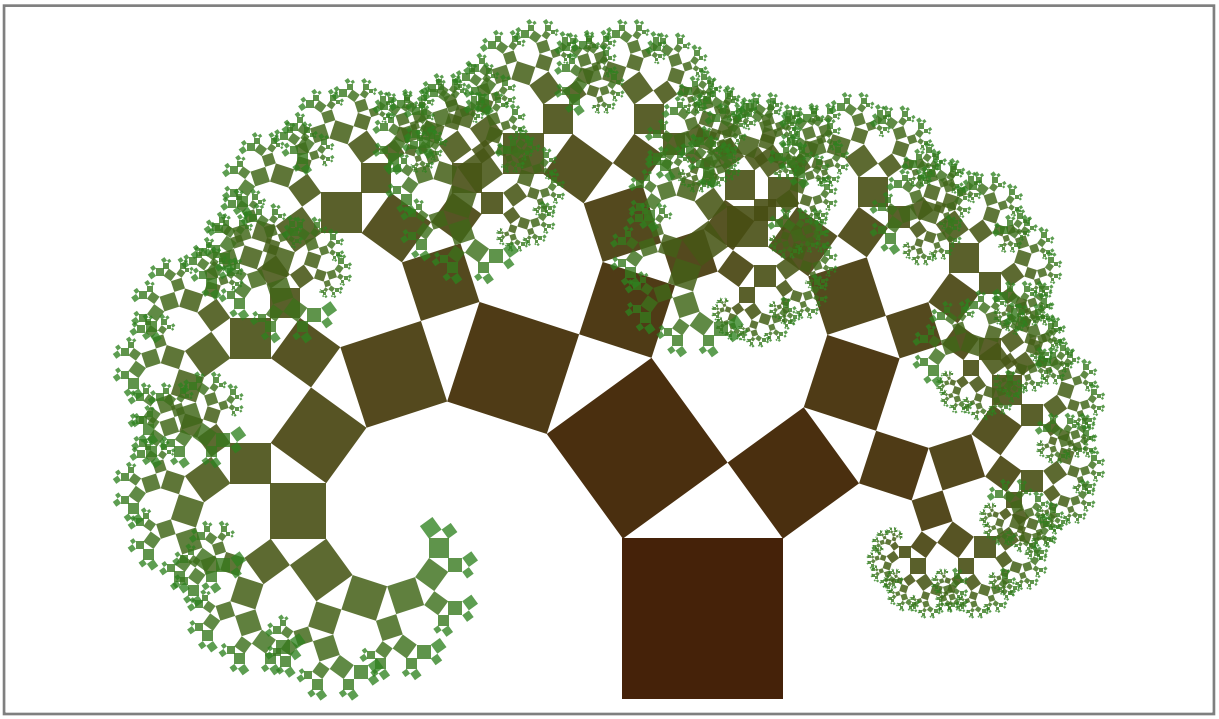
```
#random-fractal-tree(
  12,
  seed: 1,
  root-color: black,
  leaf-color: orange.transparentize(40%),
  trunk-len: 50,
  trunk-rad: 2.0,
  angle: 0.18,
  ratio: 0.78
)

#random-fractal-tree(
  12,
  seed: 12,
  root-color: black,
  leaf-color: orange.transparentize(40%),
  trunk-len: 50,
  trunk-rad: 2.0,
  angle: 0.18,
  ratio: 0.78
)
```



4.3.3 Pythagorean Tree

A 12th order Pythagorean tree.

```
#pythagorean-tree(
  12,
  leaf-color: rgb("#228B22C0"),
  trunk-len: 60
)
```



5 Parametric Curve

5.1 Guide

Plot the curve according to the parametric equations. Typically, the parametric equation is expressed in the form $x = x(t), y = y(t)$ in the direct coordinate system.

5.2 Reference

5.2.1 `lissajous-curve`

Generate Lissajous curve[16].

The original parametric equations for the graph are:

$$x(t) = A\sin(at + \delta)$$
$$y(t) = B\sin(bt)$$

Parameters

```
lissajous-curve(
  int,
  int,
  int float,
  x-size: int float,
  y-size: int float,
  padding: int float,
  fill: fill,
  fill-rule: str,
  stroke: stroke,
) -> content
```

a `int` *Required* *Positional*

The frequency of x-axis. Valid range is 1 to 100.

b `int` *Required* *Positional*

The frequency of y-axis. Valid range is 1 to 100.

d `int` or `float` *Settable*

The phase offset of x-axis (in π radius). Valid range is $[0, 2]$.

x-size `int` or `float` *Settable*

The width of the image (in pt). Must be positive.

[16] https://en.wikipedia.org/wiki/Lissajous_curve

Default: `100`

`y-size` `int` or `float` *Settable*

The height of the image (in pt). Must be positive.

Default: `100`

`padding` `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

`fill` `fill` *Settable*

How to fill the curve.

Default: `none`

`fill-rule` `str` *Settable*

The drawing rule used to fill the curve. Valid value is `"non-zero"` or `"even-odd"`.

Default: `"non-zero"`

`stroke` `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

`graph` `content` *Returned*

Returned graph, contained within the `box` element.

5.2.2 `hypotrochoid`

Generate hypotrochoid[17].

The original parametric equations for the graph are:

$$x(t) = (a-b)\cos t + h\cos\left(\frac{a-b}{b}t\right)$$

$$y(t) = (a-b)\sin t - h\sin\left(\frac{a-b}{b}t\right)$$

[17] https://en.wikipedia.org/wiki/Hypotrochoid

Parameters

```
hypotrochoid(
  int,
  int,
  int,
  size: int float,
  padding: int float,
  fill: fill,
  fill-rule: str,
  stroke: stroke,
) -> content
```

a `int` *Required* *Positional*

The radius of exterior circle. Valid range is 1 to 100.

b `int` *Required* *Positional*

The radius of interior circle. Valid range is 1 to 100.

h `int` *Required* *Positional*

The distance from the center of the interior circle. Valid range is 1 to 100.

size `int` or `float` *Settable*

The width/height of the image (in pt). Must be positive.

Default: `100`

padding `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

fill `fill` *Settable*

How to fill the curve.

Default: none

fill-rule `str` *Settable*

The drawing rule used to fill the curve. Valid value is `"non-zero"` or `"even-odd"`.

Default: `"non-zero"`

stroke `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

graph `content` *Returned*

Returned graph, contained within the box element.

5.2.3 `epitrochoid`

Generate epitrochoid[18].

The original parametric equations for the graph are:

$$x(t) = (a+b)\cos t - h\cos\left(\frac{a+b}{b}t\right)$$

$$y(t) = (a+b)\sin t - h\sin\left(\frac{a+b}{b}t\right)$$

Parameters

```
epitrochoid(
  int,
  int,
  int,
  size: int float,
  padding: int float,
  fill: fill,
  fill-rule: str,
  stroke: stroke,
) -> content
```

a `int` *Required* *Positional*

The radius of exterior circle. Valid range is 1 to 100.

b `int` *Required* *Positional*

The radius of interior circle. Valid range is 1 to 100.

h `int` *Required* *Positional*

The distance from the center of the interior circle. Valid range is 1 to 100.

size `int` or `float` *Settable*

The width/height of the image (in pt). Must be positive.

[18] https://en.wikipedia.org/wiki/Epitrochoid

Default: `100`

`padding` `int` or `float` *Settable*

The spacing around the content (in pt). Must be non-negative.

Default: `0`

`fill` `fill` *Settable*

How to fill the curve.

Default: `none`

`fill-rule` `str` *Settable*

The drawing rule used to fill the curve. Valid value is `"non-zero"` or `"even-odd"`.

Default: `"non-zero"`

`stroke` `stroke` *Settable*

How to stroke the curve.

Default: `black + 1pt`

`graph` `content` *Returned*

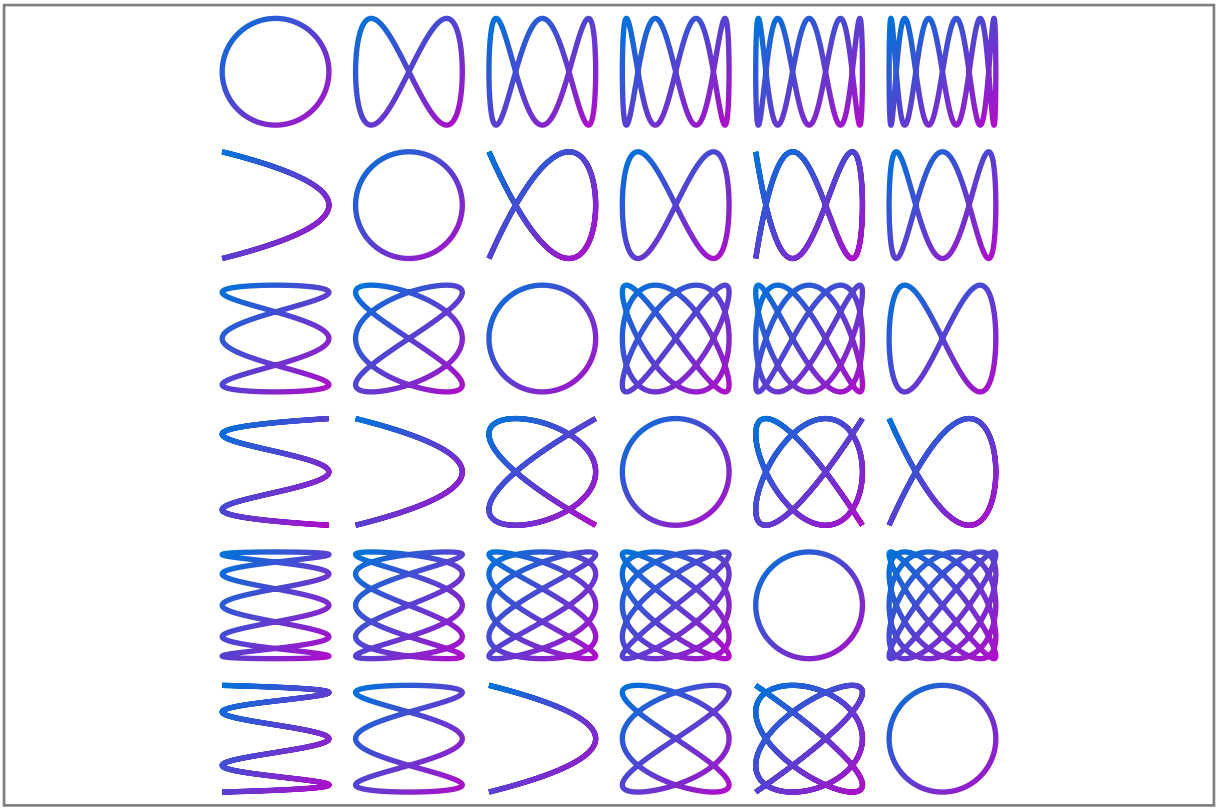
Returned graph, contained within the box element.

5.3 Examples

5.3.1 Lissajous Curves

Lissajous curves with various parameters.

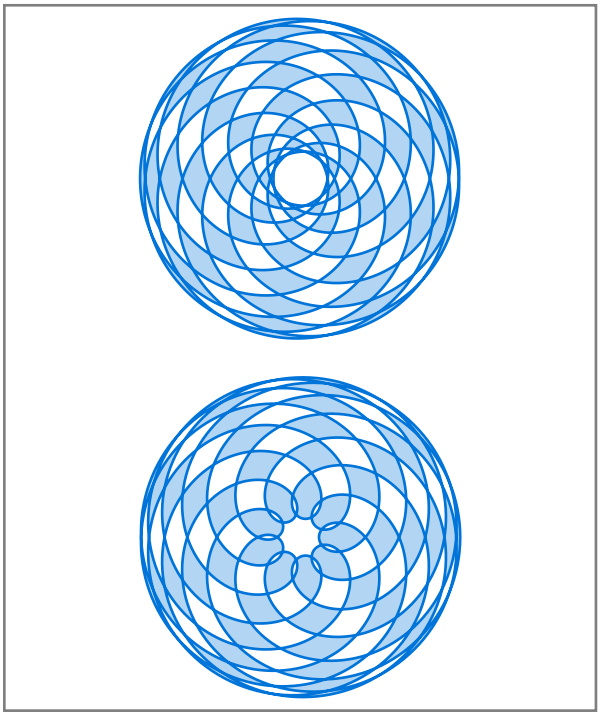
```
#let lc = lissajous-curve.with(
  x-size: 40,
  y-size: 40,
  stroke: gradient.linear(blue, purple, angle: 45deg) + 2pt
)

#grid(
  columns: 6,
  gutter: 10pt,
  lc(1,1,1/2), lc(1,2,1/2), lc(1,3,1/2), lc(1,4,1/2), lc(1,5,1/2), lc(1,6,1/2),
  lc(2,1,1/2), lc(2,2,1/2), lc(2,3,1/2), lc(2,4,1/2), lc(2,5,1/2), lc(2,6,1/2),
  lc(3,1,1/2), lc(3,2,1/2), lc(3,3,1/2), lc(3,4,1/2), lc(3,5,1/2), lc(3,6,1/2),
  lc(4,1,1/2), lc(4,2,1/2), lc(4,3,1/2), lc(4,4,1/2), lc(4,5,1/2), lc(4,6,1/2),
  lc(5,1,1/2), lc(5,2,1/2), lc(5,3,1/2), lc(5,4,1/2), lc(5,5,1/2), lc(5,6,1/2),
  lc(6,1,1/2), lc(6,2,1/2), lc(6,3,1/2), lc(6,4,1/2), lc(6,5,1/2), lc(6,6,1/2)
)
```



5.3.2 Spirograph Curves

A hypotrochoid curve and an epitrochoid curve.

```
#hypotrochoid(
  9,
  16,
  5,
  size: 120,
  fill: blue.lighten(70%),
  fill-rule: "even-odd",
  stroke: blue
)

#epitrochoid(
  9,
  10,
  15,
  size: 120,
  fill: blue.lighten(70%),
  fill-rule: "even-odd",
  stroke: blue
)
```



6 Roadmap

This page lists planned features for this package.

- ☐ General infrastructure for iterative/recursive method fractals
- ☐ More flexible graphic configuration
- ☐ More attractive fractals and curves
- ☐ More fractal types, such as iterated function system (IFS), escape-time fractals, etc.
- ☐ Accelerate graph generation based on the WebAssembly plugin