Package ‘sfcurve’

September 13, 2024

Type Package

Title 2x2, 3x3 and Nxn Space-Filling Curves

Version 1.0.0

Date 2024-09-10

Depends R (>= 4.0.0)

Imports grid, Rcpp, methods, colorRamp2

Suggests rmarkdown, knitr, rgl, testthat, ComplexHeatmap, igraph, digest

VignetteBuilder knitr

Description Implementation of all possible forms of 2x2 and 3x3 space-filling curves, i.e., the generalized forms of the Hilbert curve <https://en.wikipedia.org/wiki/Hilbert_curve>, the Peano curve <https://en.wikipedia.org/wiki/Peano_curve> and the Peano curve in the meander type (Figure 5 in <https://eudml.org/doc/141086>). It can generates nxn curves expanded from any specific level-1 units. It also implements the H-curve and the three-dimensional Hilbert curve.

URL https://github.com/jokergoo/sfcurve, https://jokergoo.github.io/sfcurve/

License MIT + file LICENSE

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation yes

Author Zuguang Gu [aut, cre] (<https://orcid.org/0000-0002-7395-8709>)

Maintainer Zuguang Gu <z.gu@dkfz.de>

Repository CRAN

Date/Publication 2024-09-13 18:00:02 UTC

Contents

`all_traverse_paths` *All traverse paths of a sequence*

Description

All traverse paths of a sequence

Usage

```
all_traverse_paths(rules, p)

get_one_traverse_path(rules, p)

plot_traverse_paths(rules, p, type = c("all", "11|22", "12|21"))
```

Arguments

`rules`	An `sfc_rules` object.
`p`	An `sfc_sequence` sequence. `p` and `rules` should have the same universe base set. Please provide `p` as a small sequence because the total number of all traverse paths might be very huge.
`type`	If the value is `"11|22"`, it highlights the paths only via 1-1/2-2 corners. If the value is `"12|21"`, it highlights the paths only via 1-2/2-1 corners.

Details

Given an input sequence with rotations, `all_traverse_paths()` lists all combinations of expansion codes from the first letter to the last letter in `p` (i.e. all possible traverse paths).

`get_one_traverse_path()` returns one random traverse path.

Value

`all_traverse_paths()` returns a list of integer vectors.

`get_one_traverse_path()` returns an integer vector.

Examples

```
# expansion rules for the general 3x3 curves
p = SFC_RULES_3x3_COMBINED@rules$I[[3]]
get_one_traverse_path(SFC_RULES_3x3_COMBINED, p)
get_one_traverse_path(SFC_RULES_3x3_COMBINED, p)
get_one_traverse_path(SFC_RULES_3x3_COMBINED, p)
get_one_traverse_path(SFC_RULES_3x3_COMBINED, p)
#
p = SFC_RULES_3x3_COMBINED@rules$I[[3]]
plot_traverse_paths(SFC_RULES_3x3_COMBINED, p)
plot_traverse_paths(SFC_RULES_3x3_COMBINED, p, type = "11|22")
plot_traverse_paths(SFC_RULES_3x3_COMBINED, p, type = "12|21")

# 2x2 curve
p = sfc_2x2("I", 11)
plot_traverse_paths(SFC_RULES_2x2, p)

# Peano curve
p = sfc_3x3_peano("I", 1)
plot_traverse_paths(SFC_RULES_3x3_PEANO, p)

# Meander curve
p = sfc_3x3_meander("I", 1)
plot_traverse_paths(SFC_RULES_3x3_MEANDER, p)
```

BASE_I *Base patterns*

Description

A list of pre-defined base patterns. See the **Examples** section.

Usage

```
BASE_I

BASE_J

BASE_R

BASE_L

BASE_U

BASE_B

BASE_D

BASE_P

BASE_Q

BASE_C

BASE_LIST
```

Format

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `sfc_base` of length 1.

An object of class `list` of length 10.

Details

`BASE_I` and `BASE_J` are identical. They are only used to distinguish the two "going forward" patterns for the level-1 units with 11/22 corner values, i.e. bottom-left to top-right, and bottom-right to top-left.

Value

`sfc_base` objects.

Examples

```
BASE_I
BASE_J
BASE_R
BASE_L
BASE_U
BASE_B
BASE_D
BASE_P
BASE_Q
BASE_C
draw_multiple_curves(
    BASE_I, BASE_J, BASE_R, BASE_L, BASE_U,
    BASE_B, BASE_D, BASE_P, BASE_Q, BASE_C,
    nrow = 2
)
```

`draw_multiple_curves` *Draw multiple curves*

Description

Draw multiple curves

Usage

```
draw_multiple_curves(
  ...,
  list = NULL,
  nrow = NULL,
  ncol = NULL,
  extend = TRUE,
  title = FALSE,
  closed = FALSE,
  padding = unit(0, "pt"),
  lwd = 4,
  col = NULL
)
```

Arguments

`...`	A list of `sfc_sequence` objects or objects in its child classes, a list of `grid::grob` objects or a list of two-column coordinate matrices, i.e., all the forms that can represent curves in this package.
`list`	The list of curve object can also be directly specified as a "list" object.
`nrow`	Number of rows in the layout.
`ncol`	Number of columns in the layout.
`extend`	Whether to draw the entry and exit segments? It is only used when input is a list of `sfc_sequence` objects.
`title`	Whether to add titles on each panel? The title is constructed in the form of `initial_seed|expansion_codes`, e.g. `I|111`. The value can be a vector of user-defined strings.
`closed`	Whether the curves are closed? The value should be a logical vector. If it is `TRUE`, the last point is connected to the first point in the curve to form a closed curve. Length of `closed` can be 1 or the number of curves.
`padding`	Space around each curve. The value should be a `grid::unit` object with length 1.
`lwd`	Line width with length 1.
`col`	Color for the segments with length 1. If the value is `NULL`, it uses the "Spectral" color palettes from the **RColorBrewer** package.

Details

This function is used for quick comparison on curves.

Value

No value is returned.

Examples

```
# for all forms of curves initialized by base pattern 'R', with rotation 0, and on level 3
draw_multiple_curves(
    sfc_2x2("R", "111"),
    sfc_2x2("R", "112"),
    sfc_2x2("R", "121"),
    sfc_2x2("R", "122"),
    sfc_2x2("R", "211"),
    sfc_2x2("R", "212"),
    sfc_2x2("R", "221"),
    sfc_2x2("R", "222"),
    nrow = 2, title = TRUE)

# simply a list of sequences
# note they only contain I/R/L, so the base patterns I/R/L are internally used
draw_multiple_curves(
    sfc_sequence("IIII"),
    sfc_sequence("RRRR"),
    sfc_sequence("RRLL"),
    nrow = 1
)
```

`draw_rules_2x2` *Draw the expansion rules*

Description

Draw the expansion rules

Usage

```
draw_rules_2x2()

draw_rules_3x3_peano(flip = FALSE)

draw_rules_3x3_meander(flip = FALSE)
```

Arguments

`flip`	Whether to use the "flipped" rules? For the Peano curve and the Meander curve, there is also a "fliiped" version of the curve expansion rules. See the vignettes for details.

Details

The expansion rules define how the curve is expanded from level-0 to level-1.

Value

No value is returned.

Examples

```
draw_rules_2x2()
# the units in the main rules of the Peano curve are vertical
draw_rules_3x3_peano()
# the units in the flipped rules of the Peano curve are horizontal
draw_rules_3x3_peano(flip = TRUE)
# the units in the main rules of the Meander curve are "forward"
# i.e. the direction of the "wave" is the same as the direction of the curve
draw_rules_3x3_meander()
# the units in the flipped rules of the Meander curve are "backward"
draw_rules_3x3_meander(flip = TRUE)
```

`H0` *Seed sequences of the H-curve*

Description

Seed sequences of the H-curve

Usage

```
H0

H1

H2
```

Format

An object of class `matrix` (inherits from `array`) with 4 rows and 2 columns.

An object of class `matrix` (inherits from `array`) with 16 rows and 2 columns.

An object of class `matrix` (inherits from `array`) with 16 rows and 2 columns.

Details

The three objects simply contain coordinates of points on the three base H-curves.

Value

Two-column matrices.

Examples

```
H0
draw_multiple_curves(H0, H1, H2, nrow = 1, closed = TRUE)
```

`hilbert_3d` *Three dimensional Hilbert curve*

Description

Three dimensional Hilbert curve

Usage

```
hilbert_3d(level = 2L)
```

Arguments

`level`	The level of the curve.

Details

There are many forms of 3D Hilbert curve. Here we only implement one specific form.

Value

A three-column matrix of coordinates of points on the 3D Hilbert curve.

See Also

Michael Bader. Space-Filling Curves: An Introduction with Applications in Scientific Computing, Springer Science & Business Media, 2012. doi:10.1007/9783642310461.

Examples

```
pos = hilbert_3d(2)
if(require(rgl) && interactive()) {
    plot3d(pos, type = "l", lwd = 4, col = 2)
}
```

`hilbert_curve` *Various curves in their standard forms*

Description

Various curves in their standard forms

Usage

```
hilbert_curve(level = 2L, by = "Cpp")

moore_curve(level = 2L)

beta_omega_curve(level = 2L)

peano_curve(level = 2L, pattern = "vvvvvvvvv", by = "Cpp")

meander_curve(level = 2L, pattern = "fffffffff", code = rep(1, level))

h_curve(iteration = 2L)
```

Arguments

`level`	Level of the curve.
`by`	Which implementation? Only for the testing purpose.
`pattern`	The orientation of units on level-2, i.e. the orientation of the 9 3x3 units. The value should be a string with 9 letters of "v"/"h" (vertical or horizontal) for the Peano curve, and "f"/"b" (forward or backward) for the Meander curve. The length of the string should be maximal 9. If the length is smaller than 9, the stringis automatically recycled.
`code`	Internally used.
`iteration`	Number of iterations.

Details

These are just special forms of `sfc_2x2()`, `sfc_3x3_peano()`, `sfc_3x3_meander()` and `sfc_h()`.

Value

A two-column matrix of coordinates of points on the curve.

Examples

```
hilbert_curve(2)
draw_multiple_curves(
    hilbert_curve(3),
    hilbert_curve(4),
    nrow = 1
)
draw_multiple_curves(
    moore_curve(3),
    moore_curve(4),
    nrow = 1
)
draw_multiple_curves(
    beta_omega_curve(3),
    beta_omega_curve(4),
    nrow = 1
)
draw_multiple_curves(
    peano_curve(2),
    peano_curve(3),
    nrow = 1
)
draw_multiple_curves(
    peano_curve(3, pattern = "vh"),
    peano_curve(3, pattern = "vvvhhhvvv"),
    nrow = 1
)
draw_multiple_curves(
    meander_curve(2),
    meander_curve(3),
    nrow = 1
)
draw_multiple_curves(
    meander_curve(3, pattern = "fbfbfbfbf"),
    meander_curve(3, pattern = "bbbbbffff"),
    nrow = 1
)
draw_multiple_curves(
    h_curve(1),
    h_curve(2),
    nrow = 1, closed = TRUE
)
```

`level1_unit_orientation,sfc_3x3_peano-method`

Level-1 unit in the Peano curve and Meander curve

Description

Level-1 unit in the Peano curve and Meander curve

Usage

```
## S4 method for signature 'sfc_3x3_peano'
level1_unit_orientation(p)
```

```
## S4 method for signature 'sfc_3x3_peano'
change_level1_unit_orientation(p, to = c("horizontal", "vertical"))

## S4 method for signature 'sfc_3x3_meander'
level1_unit_orientation(p)

## S4 method for signature 'sfc_3x3_meander'
change_level1_unit_orientation(p, to = c("forward", "backward"))
```

Arguments

p	For `level1_unit_orientation()`, it is an `sfc_3x3_peano` or `sfc_3x3_meander` unit on level 1. For `change_level1_unit_orientation()`, it is a normal `sfc_3x3_peano` or `sfc_3x3_meander` object.
to	A string of "vertical/horizontal" (on the `sfc_3x3_peano` object) or "forward/backward" (on the `sfc_3x3_meander` object).

Details

"vertical" and "horizontal" correspond to the direction of the "long segments" in a Peano curve (see **Examples**).

`level1_unit_orientation()` is normally used inside `sfc_apply()`. `change_level1_unit_orientation()` changes all level-1 units of a Peano curve or a Meander curve simultaneously.

Value

`level1_unit_orientation()` returns "verticalhorizontal" (on the `sfc_3x3_peano` object) or "forward/backward" (on the `sfc_3x3_meander` object).

`change_level1_unit_orientation()` returns an `sfc_3x3_peano` or `sfc_3x3_meander` object.

Examples

```
p = sfc_3x3_peano("I", 111)
# the first level-1 unit
level1_unit_orientation(p[1:9, TRUE])
# the fourth level-1 unit
level1_unit_orientation(p[1:9 + 27, TRUE])
p2 = change_level1_unit_orientation(p, "horizontal")
p3 = change_level1_unit_orientation(p, "vertical")
draw_multiple_curves(p, p2, p3,
    title = c("original", "all horizontal", "all vertical"), nrow = 1)
# by default, orientations of all level-1 units in Meander curve are forward
p = sfc_3x3_meander("I", 111)
level1_unit_orientation(p[1:9, TRUE])
p2 = change_level1_unit_orientation(p, "backward")
draw_multiple_curves(p, p2,
   title = c("all forward", "all backward"), nrow = 1)
```

plot_segments *Plot segments*

Description

Plot segments

Usage

```
plot_segments(x, grid = FALSE, title = FALSE, closed = FALSE, ...)
```

Arguments

`x`	A two-column matrix of coordinates of points.
`grid`	Whether to add grid lines on the plot?
`title`	The value should be `FALSE` or a string.
`closed`	Whether the curve is closed?
`...`	Other arguments passed to `sfc_grob()`.

Details

This function is only for a quick demonstration of curves represented as two-column coordinate matrices.

Value

No value is returned.

Examples

```
pos = cbind(c(0, 0, 1, 1, 2, 2, 3, 3, 2, 2, 1, 1),
            c(1, 2, 2, 3, 3, 2, 2, 1, 1, 0, 0, 1))
plot_segments(pos)
```

sfc_2x2 *Create space-filling curves*

Description

Create space-filling curves

Usage

```
sfc_2x2(seed, code = integer(0), rot = 0L)

sfc_3x3_peano(seed, code = integer(0), rot = 0L, level = NULL, flip = FALSE)

sfc_3x3_meander(seed, code = integer(0), rot = 0L, flip = FALSE)
```

Arguments

`seed`	The seed sequence. In most cases, the seed sequence is a single base pattern, which can be specified as a single letter, then `rot` controls the initial rotation of the base pattern. It also supports a sequence with more than one base patterns as the seed sequence. In this case, it can be specified as a string of more than one base letters, then `rot` can be set to a single rotation scalar which controls the rotation of the first letter, or a vector with the same length as the number of base letters.
`code`	A vector of the expansion code. The left side corresponds to the top levels of the curve and the right side corresponds to the bottom level of the curve. The value can be set as a vector e.g. `c(1, 2, 1)`, or as a string e.g. `"121"`, or as a number e.g. `121`.
`rot`	Rotation of the seed sequence, measured in the polar coordinate system, in degrees.
`level`	Specifically for `sfc_3x3_peano()`, since there is only one expansion code 1, it can also be generated by `rep(1, level)`.
`flip`	Whether to use the "flipped" rules? For the Peano curve and the Meander curve, there is also a "fliiped" version of curve expansion rules. On each level expansion in the Peano curve and the Meander curve, a point expands to nine points in 3x3 grids. Thus the value of `flip` can be set as a logical vector of length of nine that controls whether to use the flipped expansion for the corresponding unit. Besides such "1-to-9" mode, `flip` can also be set as a function which acccepts the number of current points in the curve and return a logical vector with the same length, i.e. the "all-to-all*9" mode.

Details

- `sfc_2x2()` generates the Hilbert curve from the seed sequence.
- `sfc_3x3_peano()` generates the Peano curve from the seed sequence.
- `sfc_3x3_meander()` generates the Meander curve from the seed sequence.

Value

- `sfc_hilbert()` returns an `sfc_2x2` object.
- `sfc_peano()` returns an `sfc_3x3_peano` object.
- `sfc_meander()` returns an `sfc_3x3_meander` object.

These three classes are child classes of `sfc_nxn`.

Examples

```
sfc_2x2("I", "111") |> plot()
sfc_2x2("I", "111", rot = 90) |> plot()
sfc_2x2("IR", "111", rot = 90) |> plot()
sfc_3x3_peano("I", "111") |> plot()
sfc_3x3_peano("I", "111",
    flip = c(FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, TRUE, FALSE, FALSE)) |> plot()
sfc_3x3_peano("IJ", "111") |> plot()

sfc_3x3_peano("I", level = 4, flip = function(p) {
    p@rot %in% c(90, 270)
}) |> plot(lwd = 1)

level = 4
sfc_3x3_peano("I", level = level, flip = function(p) {
     if(length(p) == 9^(level-1)) {
         l = rep(FALSE, length(p))
         ind = 1:9^2 + 9^2*rep(c(0, 2, 4, 6, 8), each = 9^2)
         l[ind] = p@rot[ind] %in% c(90, 270)

         ind = 1:9^2 + 9^2*rep(c(1, 3, 5, 7), each = 9^2)
         l[ind] = p@rot[ind] %in% c(0, 180)

         l
    } else {
         rep(FALSE, length(p))
    }
}) |> plot(lwd = 1)

sfc_3x3_meander("I", "111") |> plot()
sfc_3x3_meander("I", "111",
    flip = c(TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, FALSE, FALSE, TRUE)) |> plot()
sfc_3x3_meander("IR", "111") |> plot()
```

`sfc_3x3_combined` *General 3x3 space-filling curves*

Description

General 3x3 space-filling curves

Usage

```
sfc_3x3_combined(seed, level = 0, rot = 0L, flip = FALSE)

## S4 method for signature 'sfc_3x3_combined'
sfc_expand(p, code = NULL, flip = FALSE)

draw_rules_3x3_combined(flip = FALSE)
```

Arguments

`seed`	The seed sequence. In most cases, the seed sequence is a single base pattern, which can be specified as a single letter, then `rot` controls the initial rotation of the base pattern. It also supports a sequence with more than one base patterns as the seed sequence. In this case, it can be specified as a string of more than one base letters, then `rot` can be set to a single rotation scalar which controls the rotation of the first letter, or a vector with the same length as the number of base letters.
`level`	Level of the curve. Currently it is restricted to an integer no bigger than 5.
`rot`	Rotation of the seed sequence, measured in the polar coordinate system, in degrees.
`flip`	The same setting as in `sfc_3x3_peano()` or `sfc_3x3_meander()`.
`p`	An `sfc_3x3_combined` object.
`code`	Ignore. The traverse codes are selected randomly.

Details

This type of 3x3 curve uses the combintation of base patterns from both the Peano curve and the Meander curve. On each level, the traverse path is randomly selected.

Value

`sfc_3x3_combined()` returns an `sfc_3x3_combined` object.

Examples

```
draw_multiple_curves(
    sfc_3x3_combined("I", level = 3),
    sfc_3x3_combined("I", level = 3),
    sfc_3x3_combined("I", level = 3),
    nrow = 1
)
draw_rules_3x3_combined()
draw_rules_3x3_combined(flip = TRUE)
```

`sfc_4x4_meander` *4x4 space-filling curves in meander type*

Description

4x4 space-filling curves in meander type

Usage

```
sfc_4x4_meander(seed, code = integer(0), rot = 0L, flip = FALSE, type = 1L)

## S4 method for signature 'sfc_4x4_meander'
sfc_expand(p, code, flip = FALSE)

draw_rules_4x4_meander(type = 1, flip = FALSE)
```

Arguments

seed	The seed sequence. In most cases, the seed sequence is a single base pattern, which can be specified as a single letter, then `rot` controls the initial rotation of the base pattern. It also supports a sequence with more than one base patterns as the seed sequence. In this case, it can be specified as a string of more than one base letters, then `rot` can be set to a single rotation scalar which controls the rotation of the first letter, or a vector with the same length as the number of base letters.
code	A vector of the expansion code. The left side corresponds to the higher levels (more to the top-level) of the curve and the right side corresponds to the lower level (more to the bottom-level) of the curve. The value can be set as a vector e.g. `c(1, 2, 1)`, or as a string e.g. `"121"`, or as a number e.g. `121`.
rot	Rotation of the seed sequence, measured in the polar coordinate system, in degrees.
flip	The same setting as in `sfc_3x3_peano()` or `sfc_3x3_meander()`.
type	Which type of rules to use? 1 for `SFC_RULES_4x4_MEANDER_1` and 2 for `SFC_RULES_4x4_MEANDER_2`.
p	An `sfc_4x4_meander` object.

Details

It is an extension of the 3x3 Meander curves to mode 4. For simplicity, it only supports `I`/`R`/`L` base patterns.

Value

`sfc_4x4_meander()` returns an `sfc_4x4_meander` object.

Examples

```
draw_multiple_curves(
    sfc_4x4_meander("I", "11", type = 1),
    sfc_4x4_meander("I", "12", type = 1),
    sfc_4x4_meander("I", "11", type = 2),
    sfc_4x4_meander("I", "12", type = 2),
    nrow = 2
)
seed = paste(rep(paste0("R", sapply(0:10, function(i) strrep("I", i))), each = 2), collapse="")
sfc_4x4_meander(seed, 1) |> plot()
draw_rules_4x4_meander(type = 1)
draw_rules_4x4_meander(type = 2)
```

sfc_apply,sfc_nxn-method

Apply to every unit in the sfc_nxn curve

Description

Apply to every unit in the sfc_nxn curve

Usage

```
## S4 method for signature 'sfc_nxn'
sfc_apply(p, depth = 1, fun = function(x) x)
```

Arguments

`p`	An `sfc_nxn` object.
`depth`	An integer between 0 and `level-1` of the curve.
`fun`	A function of which the argument `x` is a subunit in the curve. The subunit is an `sfc_nxn` object but only contains the current sub-sequence. The function should return an `sfc_seuqence` object with the same length as of `x`. The function can take an optional second argument which the index of the current subunit in the curve.

Details

This function is mainly used to flip subunits on various levels on the curve, thus mainly on the Peano curve and the Meander curve. A depth of 0 corresponds to the complete curve. A depth of 1 corresponds to the nine first-level units, et al.

Currently, `sfc_apply()` only works on curves with a single base pattern as the seed.

Value

An `sfc_nxn` object.

Examples

```
p = sfc_3x3_peano("I", level = 3)
# flip the global curve
draw_multiple_curves(
    p,
    sfc_apply(p, 0, sfc_flip_unit),
    nrow = 1
)

# flip all the subunits on depth = 1
draw_multiple_curves(
    p,
    sfc_apply(p, 1, sfc_flip_unit),
    nrow = 1
)

# flip all the subunits on depth = 2
draw_multiple_curves(
    p,
    sfc_apply(p, 2, sfc_flip_unit),
    nrow = 1
)

# flip all level-1 patterns on the Peano curve to horizontal
# only works on the lowest subunit,
p2 = sfc_apply(p, 2, function(x) {
    if(level1_unit_orientation(x) == "vertical") {
        sfc_flip_unit(x)
    } else {
        x
    }
})
# then on depth=1, only flip the unit with odd index
p3 = sfc_apply(p2, 1, function(x, i) {
    if(i %% 2 == 1) {
        sfc_flip_unit(x)
    } else {
        x
    }
})
draw_multiple_curves(p2, p3, nrow = 1)

# flip all level-1 patterns to vertical
p3 = sfc_apply(p, 2, function(x) {
    if(level1_unit_orientation(x) == "horizontal") {
        sfc_flip_unit(x)
    } else {
        x
    }
})
draw_multiple_curves(p, p3, nrow = 1)
```

`sfc_base` *Constructor of the sfc_base class*

Description

Constructor of the sfc_base class

Usage

```
sfc_base(
    letter,
    in_direction,
    out_direction,
    grob = NULL,
    primary = TRUE,
    open = TRUE
  )
```

Arguments

`letter`	A single letter to represent the base pattern.
`in_direction`	The direction of the segment that enters the point, measured in the polar coordinate system, in degrees.
`out_direction`	The direction of the segment that leaves the point, measured in the polar coordinate system, in degrees.
`grob`	A `grid::grob()` object of this base pattern. If it is not set, it is generated according to `in_direction` and `out_direction`.
`primary`	Currently, going forward, turning left and turning right can be set as primary base patterns because other high-level patterns can be built from them.
`open`	Can the base pattern be connected to other base patterns?

Details

The "base pattern" is designed not only for single point but also for combination of points that form a "base curve". However, currently, it is fixed to the single point base pattern.

Currently, this package supports 2x2 and 3x3 space-filling curves that fills grids in 2D space constructed by the Gaussian integers. And when the curve expands, we only allow the segments to go forward, backward, left and right. Thus there are the following base patterns pre-defined in this package:

- `BASE_I`/`BASE_J`: go forward.
- `BASE_R`: turn right.
- `BASE_L`: turn left.
- `BASE_U`: go backward.
- `BASE_B`: leave the start point where the start point is closed.
- `BASE_D`: leave the start point where the start point is closed.
- `BASE_P`: return to the end point where the end point is closed.
- `BASE_Q`: return to the end point where the end point is closed.
- `BASE_C`: self-closed.

The base pattern determines the final form of the curve.

Value

An `sfc_base` object.

Examples

```
BASE_I
```

`sfc_expand,sfc_2x2-method`

Expand the curve to the next level

Description

Expand the curve to the next level

Usage

```
## S4 method for signature 'sfc_2x2'
sfc_expand(p, code, flip = FALSE)

## S4 method for signature 'sfc_3x3_peano'
sfc_expand(p, code = 1, flip = FALSE)

## S4 method for signature 'sfc_3x3_meander'
sfc_expand(p, code, flip = FALSE)
```

Arguments

`p`	An `sfc_2x2` object or other related objects.
`code`	Expansion code, a single integer.
`flip`	Whethe to flip level-1 units? The value should be a logical vector of length one or the same as the length of `p`.

Details

For the Hilbert curve and Meander curve, as long as the expansion code of the first base pattern in the sequence is determinted, the expansion codes for other base patterns in the sequence are all determined. For the Peano curve, since there is only one traverse path on any level, code is ignored.

These functions are mainly used internally.

Value

An object in the same class as the input.

Examples

```
p = sfc_2x2("I", 11)
sfc_expand(p, 2) # I|211
p = sfc_3x3_peano("I", 11)
sfc_expand(p, 2) # I|211
p = sfc_3x3_meander("I", 11)
sfc_expand(p, 2) # I|211
```

sfc_expand_by_rules,sfc_rules,sfc_nxn-method

Expand a sequence

Description

Expand a sequence

Usage

```
## S4 method for signature 'sfc_rules,sfc_nxn'
sfc_expand_by_rules(p, seq, code = 1L, flip = FALSE, by = "Cpp")

## S4 method for signature 'sfc_rules,factor'
sfc_expand_by_rules(p, seq, code = 1L, flip = FALSE, by = "Cpp")

## S4 method for signature 'sfc_rules,character'
sfc_expand_by_rules(p, seq, code = 1L, flip = FALSE, by = "Cpp")
```

Arguments

`p`	An `sfc_rules` object.
`seq`	An `sfc_nxn` object or other objects.
`code`	The expansion code.
`flip`	For the Peano curve and the Meander curves, each unit can be flipped without affecting other parts in the curve. This argument controls whether to flip the unit. Since currently it only works on the Peano curve and the Meander curve, `flip` should be a logical vector of length one or with the same length as `seq`. Whether it flips horizontally, vertically or against the diagonal line is automatically choosen. The value of `flip` can also be a function which takes the current curve as the only argument.
`by`	Which implementation? Only for the testing purpose.

Value

If `seq` is an `sfc_nxn` object, the function also returns an "expanded" `sfc_nxn` object. Or else it returns an `sfc_sequence` object.

Examples

```
sfc_expand_by_rules(SFC_RULES_2x2, sfc_2x2("I"))
```

sfc_generator *Generate a nxn curve based on expansion rules*

Description

Generate a nxn curve based on expansion rules

Usage

```
sfc_generator(
  rules,
  name,
  envir = topenv(parent.frame()),
  flippable = FALSE,
  verbose = TRUE
)
```

Arguments

`rules`	An `sfc_rules` object.
`name`	Name of the curve. The name will be used for the functions that will be generated.
`envir`	Environment where the functions are exported.
`flippable`	Whether `rules` can have flipped version? If it is `TRUE`, the generated function also accepts the `flip` argument.
`verbose`	Whether to print messages?

Details

Two functions will be exported:

- `sfc_{name}()`
- `draw_rules_{name}()`

For the simplicity, flipping is not supported yet.

Value

No value is returned.

Examples

```
UNIVERSE_4x4_PEANO = c("I", "R", "L")

RULES_4x4_PEANO = list()
RULES_4x4_PEANO[["I"]][[1]] = sfc_unit("RIILLIIRRIILLIIR", rot = 0, universe = UNIVERSE_4x4_PEANO)
RULES_4x4_PEANO[["I"]][[2]] = sfc_hflip(RULES_4x4_PEANO[["I"]][[1]])
```

```
RULES_4x4_PEANO[["R"]][[1]] = sfc_unit("IIIRRIILLIIRRIIL", rot = 0, universe = UNIVERSE_4x4_PEANO)
RULES_4x4_PEANO[["R"]][[2]] = sfc_rotate(sfc_unit("LIIRRIILLIIRRIII",
                                rot = 270, universe = UNIVERSE_4x4_PEANO), 90)
RULES_4x4_PEANO[["L"]][[1]] = sfc_hflip(RULES_4x4_PEANO[["R"]][[2]])
RULES_4x4_PEANO[["L"]][[2]] = sfc_hflip(RULES_4x4_PEANO[["R"]][[1]])

SFC_RULES_4x4_PEANO = sfc_rules(rules = RULES_4x4_PEANO,
        name = "Peano 4x4",
        bases = BASE_LIST[UNIVERSE_4x4_PEANO])

sfc_generator(SFC_RULES_4x4_PEANO, "4x4_peano")
draw_rules_4x4_peano()
sfc_4x4_peano("I", 111) |> plot()
```

sfc_grob,sfc_base-method

The graphics object

Description

The graphics object

Usage

```
## S4 method for signature 'sfc_base'
sfc_grob(p)

## S4 method for signature 'sfc_sequence'
sfc_grob(
  p,
  bases = NULL,
  extend = FALSE,
  title = FALSE,
  closed = FALSE,
  lwd = 4,
  col = NULL,
  ...
)

## S3 method for class 'sfc_sequence'
plot(
  x,
  bases = NULL,
  grid = FALSE,
  extend = FALSE,
  title = FALSE,
  closed = FALSE,
  ...
)

## S4 method for signature 'sfc_nxn'
sfc_grob(
  p,
  bases = p@rules@bases,
  extend = FALSE,
  title = FALSE,
  closed = FALSE,
  ...
)

## S3 method for class 'sfc_nxn'
plot(x, grid = FALSE, extend = FALSE, title = FALSE, closed = FALSE, ...)

## S4 method for signature 'matrix'
sfc_grob(p, title = NULL, closed = FALSE, lwd = 4, col = NULL, ...)
```

Arguments

`p`	The corresponding object.
`bases`	A list of base patterns, normally `BASE_LIST` is used.
`extend`	Whether to add the entry and exit segments?
`title`	Whether to add title on the top of the plot? The title is constructed in the form of `initial_seed|expansion_code`, e.g. `I|111`. The value can also be a string.
`closed`	Whether the curve is closed?
`lwd`	Line width.
`col`	Color for segments. If the value is `NULL`, it uses the "Spectral" color palettes.
`...`	Other arguments passed to `grid::viewport()` or `sfc_grob()`.
`x`	The corresponding object.
`grid`	Whether to add grid lines on the plot?

Details

If `p` is an `sfc_sequence` and `p` contains base patterns defined in `"I/J/R/L/U/B/D/P/Q/C"`, the default `BASE_LIST` is automatically used for bases. If `p` is an `sfc_nxn` object, `bases` is already stored in `p` and it is passed to this function automatically.

Value

A `grid::grob()` object.

Examples

```
sfc_grob(BASE_I)
plot(sfc_2x2("I", "11"))
plot(sfc_2x2("I", "11"), extend = TRUE, title = TRUE, grid = TRUE)
plot(sfc_sequence("IIIRRR"))
```

sfc_h *H-curve*

Description

H-curve

Usage

```
sfc_h(h, iteration = 1, connect = c("h", "v"), random = FALSE)

expand_h(h1, h2 = h1, h3 = h1, h4 = h1, connect = "hhhh")
```

Arguments

`h`	The seed of the H-curve. The value should be one of `H0`, `H1` or `H2`.
`iteration`	Number of iterations.
`connect`	How the four subunits are connected to form the H-curve on the next level. See **Details**.
`random`	Whether to generate subunits randomly on each iteration.
`h1`	The first subunit on the bottom left.
`h2`	The second subunit on the top left.
`h3`	The third subunit on the top right.
`h4`	The fourth subunit on the bottom right.

Details

An H-curve on level k is composed with four subunits on level k-1. If we number the four subunits in the following order:

```
2   3
1   4
```

where subunit 1 connects to subunit 2, subunit 2 connects to subunit 3, et al., and e.g. subunit 1 connects to subunit 2 via its toprigth corner. Since H-curve can be thought of as a closed curve, to, e.g. let subunit 1 to connect to subunit 2, its topright corner needs to be opened. There are two segments on subunit 1 that can be removed/opened: the horizontal segment and the vertical segment on the topright corner in subunit 1.

In this way, in `sfc_h()`, the argument `connect` only accepts a single value of `"h"` or `"v"` where the types of segments for all the four subunits are the same, i.e. whether all the horizontal corner segments are opened or whether all the vertical corner segments are opened. In `expand_h()`, the argument `connect` can be set to a string with four letters or a vector of length four where the type of segments of the four subunits can be set separately.

In the random mode, each subunit is generated randomly, the type of the open segment is choosen randomly, also each subunit has a probability of 0.5 to rotate by 90 degrees.

Value

A two-column matrix of coordinates of points on the curve.

Examples

```
draw_multiple_curves(
    sfc_h(H0, iteration = 2),
    sfc_h(H2, iteration = 2),
    closed = TRUE, nrow =1
)
draw_multiple_curves(
    sfc_h(H1, iteration = 3, random = TRUE),
    sfc_h(H1, iteration = 3, random = TRUE),
    closed = TRUE, nrow = 1
)
draw_multiple_curves(
    expand_h(H0, connect = "hvvh"),
    expand_h(H1, connect = "vvhh"),
    closed = TRUE, nrow = 1
)

# set the four subunits separately
h1 = expand_h(H0, connect = "hhhh")
h2 = expand_h(H0, connect = "vvvv")
h3 = expand_h(H0, connect = "hvhv")
h4 = expand_h(H0, connect = "hvvh")
expand_h(h1, h2, h3, h4, connect = "vhvh") |>
    plot_segments(closed = TRUE)

fun = function(h, iteration) {
    for(i in 1:iteration) h = expand_h(h, connect = "vhvh")
    h
}
fun(H0, 4) |> plot_segments(closed = TRUE)
```

`sfc_is_compatible,sfc_sequence,sfc_sequence-method`
Whether two sfc_sequence objects are compatible

Description

Whether two sfc_sequence objects are compatible

Usage

```
## S4 method for signature 'sfc_sequence,sfc_sequence'
sfc_is_compatible(p1, p2, strict = TRUE)

## S4 method for signature 'sfc_sequence,sfc_rules'
```

```
sfc_is_compatible(p1, p2)

## S4 method for signature 'sfc_rules,sfc_sequence'
sfc_is_compatible(p1, p2)
```

Arguments

p1	An sfc_sequence object.
p2	An sfc_sequence object.
strict	TRUE or FALSE, see **Details**.

Details

The function compares whether the two universe base pattern sets are identical. If `strict` is `TRUE`, the order of the two universe sets should also be the same. If `strict` is `FALSE`, the universe set of `p2` can be a subset of the universe set of `p1`.

Value

A logical scalar.

Examples

```
p1 = sfc_2x2("I")
p2 = sfc_2x2("R")
sfc_is_compatible(p1, p2)

p1 = sfc_2x2("I")
p2 = sfc_sequence("R")
sfc_is_compatible(p1, p2)
sfc_is_compatible(p1, p2, strict = FALSE)

p1 = sfc_sequence("ABC")
p2 = sfc_sequence("DEF")
sfc_is_compatible(p1, p2)
```

---

sfc_level,sfc_nxn-method

The level of the curve

Description

The level of the curve

Usage

```
## S4 method for signature 'sfc_nxn'
sfc_level(p)
```

Arguments

`p`	An `sfc_nxn` object.

Value

An integer.

Examples

```
p = sfc_2x2("I", "11")
sfc_level(p)

p = sfc_2x2("I", "1111")
sfc_level(p)
```

`sfc_mode,sfc_nxn-method` *The mode of the curve*

Description

The mode of the curve

Usage

```
## S4 method for signature 'sfc_nxn'
sfc_mode(p)

## S4 method for signature 'sfc_rules'
sfc_mode(p)
```

Arguments

`p`	The corresponding object.

Value

An integer.

Examples

```
p = sfc_2x2("I", "1")
sfc_mode(p)
p = sfc_3x3_peano("I", "1")
sfc_mode(p)
sfc_mode(SFC_RULES_2x2)
sfc_mode(SFC_RULES_3x3_PEANO)
```

sfc_previous_point,sfc_base-method

The previous and the next point

Description

The previous and the next point

Usage

```
## S4 method for signature 'sfc_base'
sfc_previous_point(p, x, rot, length = 1)

## S4 method for signature 'sfc_base'
sfc_next_point(p, x, rot, length = 1)
```

Arguments

p	An sfc_base object.
x	The coordinate of the current point.
rot	Rotation of the current point.
length	Length of the segment between the previous/next point and the current point.

Value

A vector of length two.

Examples

```
sfc_previous_point(BASE_R, c(0, 0), 0)
sfc_previous_point(BASE_R, c(0, 0), 90)
sfc_previous_point(BASE_R, c(0, 0), 180)
sfc_previous_point(BASE_R, c(1, 0), 0)
sfc_previous_point(BASE_R, c(1, 0), 90)
sfc_previous_point(BASE_R, c(1, 0), 180)
sfc_next_point(BASE_R, c(0, 0), 0)
sfc_next_point(BASE_R, c(0, 0), 90)
sfc_next_point(BASE_R, c(0, 0), 180)
sfc_next_point(BASE_R, c(1, 0), 0)
sfc_next_point(BASE_R, c(1, 0), 90)
sfc_next_point(BASE_R, c(1, 0), 180)
```

sfc_reduce,sfc_nxn-method

Reduce a curve

Description

Reduce a curve

Usage

```
## S4 method for signature 'sfc_nxn'
sfc_reduce(p, to = sfc_level(p) - 1)

## S4 method for signature 'matrix'
sfc_reduce(p, to = level - 1, mode = NULL)

add_base_structure(gb, level = 1)
```

Arguments

`p`	An `sfc_nxn` object.
`to`	Which level to reduce to? Value should be between 1 and `sfc_level(p) - 1`.
`mode`	Mode of the curve.
`gb`	A grob object returned by `sfc_grob()` or a `sfc_nxn` object then `sfc_grob()` is internally applied.
`level`	The level of the unit.

Details

The reduction is applied on the coordinates of points.

`add_base_structure()` adds a base structure on a certain level to the curve.

Value

A two-column matrix of coordinates of the reduced curve.

Examples

```
p = sfc_3x3_peano("I", level = 3)
draw_multiple_curves(
    p,
    sfc_reduce(p, 2),
    sfc_reduce(p, 1),
    nrow = 1)
p = hilbert_curve(level = 4)
draw_multiple_curves(
    p,
    sfc_reduce(p, 3),
    sfc_reduce(p, 2),
    nrow = 1)
p = hilbert_curve(3)
draw_multiple_curves(
    add_base_structure(p, level = 1),
    add_base_structure(p, level = 2),
    nrow = 1
)
```

---

sfc_rotate,sfc_sequence-method

Transformations of a sequence

Description

Transformations of a sequence

Usage

```
## S4 method for signature 'sfc_sequence'
sfc_rotate(p, rot)

## S3 method for class 'sfc_sequence'
e1 ^ e2

## S4 method for signature 'sfc_sequence'
sfc_hflip(p, fix_ends = FALSE, bases = NULL)

## S4 method for signature 'sfc_sequence'
sfc_vflip(p, fix_ends = FALSE, bases = NULL)

## S4 method for signature 'sfc_sequence'
sfc_dflip(p, slop = 1L, fix_ends = FALSE, bases = NULL)

## S4 method for signature 'sfc_sequence'
sfc_reverse(p)

## S3 method for class 'sfc_sequence'
rev(x)
```

Arguments

`p, e1`	An `sfc_sequence` object.
`rot, e2`	Rotation measured in the polar coordinate system, in degrees.

`fix_ends`	By default, the curve is flipped as a complete whole, which means, the associated entry and exit directions of the curve is also adjusted accordingly. When flipping subunits in a curve, e.g. level-1 subunits in a Peano curve, we want the entry and exit direction of the subunit not changed so that the subunits are still connected in the curve after the flipping. In this case, `fix_ends` can be set to `TRUE`, then only the subunits are flipped while the connections to neighbouring subunits are not affected. See the **Examples** section.
`bases`	A list of base patterns, consider to use `BASE_LIST`. It is only used when `fix_ends = TRUE`.
`slop`	Slop of the diagonal. Value can only be 1 or -1.
`x`	An `sfc_sequence` object.

Details

- `sfc_rotate()` and `^()` rotate each base pattern.
- `sfc_hflip()` flips a sequence horizontally.
- `sfc_vflip()` flips a sequence vertically.
- `sfc_dflip()` flips a sequence against a diagonal line (with slop 1 or -1).

Value

An `sfc_sequence` object.

Examples

```
p = sfc_3x3_meander("R", 2, rot = -90)
draw_multiple_curves(
    p,
    sfc_hflip(p),
    sfc_hflip(p, fix_ends = TRUE),
    nrow = 1)
p = sfc_3x3_meander("L", 2, rot = -90)
draw_multiple_curves(
    p,
    sfc_vflip(p),
    sfc_vflip(p, fix_ends = TRUE),
    nrow = 1)
p = sfc_3x3_peano("I", 2)
draw_multiple_curves(
    p,
    sfc_dflip(p, 1),
    sfc_dflip(p, 1, fix_ends = TRUE),
    nrow = 1)
```

sfc_rules *Constructor of the sfc_rules class*

Description

Constructor of the sfc_rules class

Usage

```
sfc_rules(rules, bases, flip = list(), name = "sfc_rules")
```

Arguments

`rules`	A list of rules.
`bases`	A list of base patterns.
`flip`	A list of rules. They are "flipped" version of `rules`. The value can also simply be `TRUE`, then the flipped version is automatically generated from `rules`.
`name`	A self-defined string.

Details

It is mainly used internally.

`rules` is a two-level list. It is in a format of `rules[[ base ]][[ expansion_code ]] = sfc_unit()`. In the following example where we define the expansion rules for the 2x2 curve:

```
UNIVERSE_2x2 = c("I", "R", "L", "U", "B", "D", "P", "Q", "C")
RULES_2x2 = list()
RULES_2x2[["I"]][[1]] = sfc_unit(c("R", "L", "L", "R"), rot = 0, universe = UNIVERSE_2x2)
```

`I` is the level-0 base pattern, `[[1]]` corresponds to the first form of expansion to level-1, and the value assigned is a `sfc_unit()` object which is basically a list of base patterns.

Then we also need to provide the base patterns which define how to extend the curve. The list of base patterns is assigned to the bases argument. In the same example, we set bases as:

```
list("I" = BASE_I, "R" = BASE_R, "L" = BASE_L, "U" = BASE_U, ...)
```

where e.g. `BASE_I` is a pre-defined base pattern in the `sfc_base` class.

There are the following pre-defined rules:

- `SFC_RULES_2x2`
- `SFC_RULES_3x3_PEANO`
- `SFC_RULES_3x3_MEANDER`
- `SFC_RULES_3x3_COMBINED`
- `SFC_RULES_4x4_MEANDER_1`
- `SFC_RULES_4x4_MEANDER_2`

Check `https://github.com/jokergoo/sfcurve/blob/master/R/zz_global.R` to see how these pre-defined rules are constructed.

Value

An `sfc_rules` object.

`SFC_RULES_2x2` *Rules*

Description

A list of pre-defined expansion rules for different curves.

Usage

```
SFC_RULES_2x2

SFC_RULES_3x3_PEANO

SFC_RULES_3x3_MEANDER

SFC_RULES_3x3_COMBINED

SFC_RULES_4x4_MEANDER_1

SFC_RULES_4x4_MEANDER_2
```

Format

An object of class `sfc_rules` of length 1.

An object of class `sfc_rules` of length 1.

An object of class `sfc_rules` of length 1.

An object of class `sfc_rules` of length 1.

An object of class `sfc_rules` of length 1.

An object of class `sfc_rules` of length 1.

Details

`SFC_RULES_3x3_PEANO`, `SFC_RULES_3x3_MEANDER` and `SFC_RULES_3x3_COMBINED`, `SFC_RULES_MEANDER_4x4_1`, `SFC_RULES_MEANDER_4x4_2` also contain the "flipped" expansion rules.

`SFC_RULES_3x3_COMBINED` is a combination of `SFC_RULES_3x3_PEANO` and `SFC_RULES_3x3_MEANDER` where in `SFC_RULES_3x3_PEANO`, `J` is replaced by its original pattern `I`.

`SFC_RULES_4x4_MEANDER_1` and `SFC_RULES_4x4_MEANDER_2` are extension rules of Meander curves (3x3) on the 4x4 mode It is only for the demonstration purpose, thus only `I/R/L` are supported.

Value

`sfc_rules` objects.

Examples

```
SFC_RULES_2x2
SFC_RULES_3x3_PEANO
SFC_RULES_3x3_MEANDER
SFC_RULES_3x3_COMBINED
SFC_RULES_4x4_MEANDER_1
SFC_RULES_4x4_MEANDER_2
```

---

sfc_segments,sfc_nxn-method

Coordinates of the points on the curve

Description

Coordinates of the points on the curve

Usage

```
## S4 method for signature 'sfc_nxn'
sfc_segments(p, bases = p@rules@bases, start = c(0, 0), ...)

## S4 method for signature 'sfc_sequence'
sfc_segments(p, bases = NULL, start = c(0, 0), by = "Cpp")
```

Arguments

`p`	An `sfc_nxn` or `sfc_sequence` object.
`bases`	A list of base patterns, consider to use `BASE_LIST`.
`start`	Coordinate of the start point.
`...`	Other argument.
`by`	Which implementation? Only for the testing purpose.

Details

For the `sfc_segments()` on the `sfc_sequence` object, if `bases` is not set, it uses `BASE_LIST` internally. Make sure the sequence only contains the pre-defined base patterns.

Value

A two-column matrix of coordinates of points on the curve.

Examples

```
p = sfc_2x2("I", "11")
loc = sfc_segments(p)
plot(loc, type = "l", asp = 1)
```

`sfc_sequence` *Constructor of the sfc_sequence class*

Description

Constructor of the sfc_sequence class

Usage

```
sfc_sequence(seq, rot = 0L, universe = NULL)

sfc_seed(seq, rot = 0L, universe = NULL)

sfc_unit(seq, rot = 0L, universe = NULL)
```

Arguments

`seq`	A sequence of base patterns. The value can be a vector of letters or a single string.
`rot`	The corresponding rotations of base patterns. If it has length one and the sequence contains R/L/I (right/left/forward), `rot` controls the rotation of the first base pattern and the rotations for remaining base patterns in the sequence are automatically calculated.
`universe`	The universe of base patterns. A vector of letters.

Details

This funtion is very low-level. Normally, users do not need to directly use this constructor.

`sfc_seed` class is the same as the `sfc_sequence` class. It is used specifically as the "seed sequence" when generating the curves.

`sfc_unit` class also inherits the `sfc_sequence` class but has one additionally slot: `corner`. It is used specifically when defining the expansion rules.

Value

An `sfc_sequence` object.

Examples

```
sfc_sequence("ABCD", rot = c(0, 90, 180, 270), universe = c("A", "B", "C", "D"))
```

sfc_shape,sfc_2x2-method
Shape of the curve

Description

Shape of the curve

Usage

```
## S4 method for signature 'sfc_2x2'
sfc_shape(p)

all_2x2_shapes(level = 2)

## S4 method for signature 'sfc_3x3_peano'
sfc_shape(p)

all_3x3_peano_shapes(level = 2)

## S4 method for signature 'sfc_3x3_meander'
sfc_shape(p)

all_3x3_meander_shapes(level = 2)
```

Arguments

`p`	An `sfc_2x2` object.
`level`	Level of the 2x2 curve.

Details

The shape of the curve is defined as a form of the curve without considering entry/exit directions, rotation, flipping (reflection) nor reversing.

2x2 curve:

The process of selecting the shape segment of the curve denoted as `P` is:

1. The entry-point should locate in the bottom left subunit and the exit-point should locate in the bottom right subunit. We try the four rotations (0, 90, 180, 270), and the four rotations on the horizontally flipped curve. Once we find the transformed curve that satisfies this criterion, we name it as `P2`.
2. We also generate `P3` which is a horizontally flipped version of `rev(P2)`.
3. We compare the first point `p` of `P2` and `P3`, and select the one whose `p` has the smaller x-coordinate (i.e. more to the left of the curve). If the x-coordinates of `p` are the same in `P2` and `P3`, we select the one whose `p` has the smaller y-coordinate.

3x3 Peano curve:

The process of selecting the shape segment of the curve denoted as `P` is:

1. The entry-point should locate in the bottom left subunit and the exit-point should locate in the top right subunit. We try the four rotations (0, 90, 180, 270). Once we find the transformed curve that satisfies this criterion, we name it as `P2`.
2. We also generate `P3` which is a 180 degrees rotation on the reversed `P2`, `P4` which is a diagonal flip with slop of 1 on `P2` and `P5` which is a diagonal flip with slop of 1 on `P3`.
3. We calculate the "UID" of `P(2-5)` and pick the one with the smallest UID as the final curve.

The UID of a 3x3 Peano curve is based on the hierarchical indices of the units on it. The hierarchy of the Peano curve is traversed in a depth-first manner. On each node, the orientation of the corresponding unit is calculated where vertical is 1 and horizontal is 2. The digits are concatenated into a long string.

`all_3x3_peano_shapes()` only calculates all shapes for Peano curve on level 2.

`all_3x3_meander_shapes()` only considers the Meander curve with all subunits on all levels in forward orientation.

Value

`sfc_shape()` returns a two-column data frame of the xy-coordinates of the shape curve.

`all_2x2_shapes()` returns a list of `n` two-column data frames where each data frame corresponds to the xy-coordnates of the corresponding shape curve.

Examples

```
p1 = sfc_2x2("I", 11)
p2 = sfc_2x2("R", 22)
draw_multiple_curves(
    p1, p2,
    sfc_shape(p1), sfc_shape(p2),
    col = "black")
sl = all_2x2_shapes(2)
draw_multiple_curves(list = sl, lwd = 2, col = "black")
sl = all_2x2_shapes(3)
draw_multiple_curves(list = sl, lwd = 2, col = "black")
p = sfc_3x3_peano("I", 11)
draw_multiple_curves(
    p, sfc_dflip(p),
    sfc_shape(p), sfc_shape(sfc_dflip(p)),
    col = "black")

sl = all_3x3_peano_shapes()
length(sl)
# the first 8 shapes
draw_multiple_curves(sl[1:8], col = "black")

sl = all_3x3_meander_shapes(2)
draw_multiple_curves(list = sl, lwd = 2, col = "black")
```

`sfc_universe,sfc_rules-method` *The universe base pattern set*

Description

The universe base pattern set

Usage

```
## S4 method for signature 'sfc_rules'
sfc_universe(p)

## S4 method for signature 'sfc_sequence'
sfc_universe(p)
```

Arguments

`p`	The corresponding object.

Value

A vector of base patterns.

Examples

```
sfc_universe(SFC_RULES_2x2)
sfc_universe(SFC_RULES_3x3_PEANO)
```

`sfc_validate,sfc_sequence-method` *Validate the sequence*

Description

Validate the sequence

Usage

```
## S4 method for signature 'sfc_sequence'
sfc_validate(p, by = "sfc_2x2")

## S4 method for signature 'character'
sfc_validate(p, by = "sfc_2x2")
```

Arguments

`p`	An `sfc_sequence` object or a character string. If it is a character string, rotation of zero is assigned to the first letter.
`by`	One of `sfc_2x2`, `sfc_3x3_peano` and `sfc_3x3_meander`.

Details

It is mainly used to validate a seed sequence whether they follow the forward-left-right rule.

Value

A logical scalar.

Examples

```
try(sfc_validate("LLLLL"))
try(sfc_validate(sfc_sequence("IIIII", rot = c(0, 90, 180, 270, 0),
       universe = sfc_universe(SFC_RULES_2x2))))
```

`show,sfc_base-method` *Print the object*

Description

Print the object

Usage

```
## S4 method for signature 'sfc_base'
show(object)

## S4 method for signature 'sfc_nxn'
show(object)

## S4 method for signature 'sfc_rules'
show(object)

## S4 method for signature 'sfc_sequence'
show(object)
```

Arguments

`object`	The corresponding object.

Value

No value is returned.

`unit_orientation` *Flip units*

Description

Flip units

Usage

```
unit_orientation(p, index = "")

## S4 method for signature 'sfc_nxn'
sfc_flip_unit(p, index = "", to = NULL)

## S4 method for signature 'sfc_unit'
sfc_flip_unit(p, bases)
```

Arguments

`p`	The corresponding object.
`index`	A string of digits representing the path on the hierarchy of the curve. The left side corresponds to the top level and the right side corresponds to the bottom level on the curve. For the 2x2 curve, the digits can only be 1-4, and for the Peano and Meander curves, the digits can be 1-9. The hierarchical index should be specified in a format of `i1:i2:i3:...` where `:` can be replaced by any non-number character. For 2x2 and 3x3 curves, `:` can be omitted and the hierarchical index can be specified as `i1i2i3...`. See examples in `sfc_index()`. The value can also be a vector where each flipping is applied in sequence.
`to`	The orientation to flip to. If the specified unit already has such orientation, the function returns the original curve.
`bases`	Normally use `BASE_LIST`.

Details

The orientation of a unit is the orientation of the line connected by the entry-corner and exit-corner of that unit.

A unit in the curve is represented as a square block (e.g. `2^k x 2^k` for the 2x2 curve and `3^k x 3^k` for the Peano and Meander curves, `k` between 1 and the level of the curve). In the 2x2 curve, if an unit can be flipped, it is symmetric, thus flipping in the 2x2 curve does not change its form. The flipping is mainly applied on the Peano curve and the Meander curves. Peano curve only allows flippings by the diagonals and the Meander curve only allows flipping horizontally or vertically. The type of flipping is choosen automatically in the function.

Currently, `sfc_flip_unit()` only works on curves with a single base pattern as the seed.

Value

`unit_orientation()` returns a string one of "vertical", "horizontal", "diagonal_1" and "diagonal_-1".

`sfc_flip_unit` returns an `sfc_nxn` object.

Examples

```
p = sfc_3x3_meander("I", 11)
draw_multiple_curves(
    p,
    sfc_flip_unit(p, "1"), # bottom left
    sfc_flip_unit(p, "2"), # bottom middle
    sfc_flip_unit(p, "3"), # bottom right
    nrow = 2)

p = sfc_3x3_peano("I", level = 3)
draw_multiple_curves(
    p,
    sfc_flip_unit(p, ""),
    sfc_flip_unit(p, "2"),
    sfc_flip_unit(p, "2:1"),
    nrow = 2)

p = sfc_3x3_peano("I", level = 2)
draw_multiple_curves(p,
    sfc_flip_unit(p, c("4", "7")),
    sfc_flip_unit(p, c("1", "2", "3", "5", "6", "8", "9")),
    nrow = 1)
```

`[.sfc_nxn` *Subunit in the curve*

Description

Subunit in the curve

Usage

```
## S3 method for class 'sfc_nxn'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'sfc_nxn'
sfc_index(p, index = "")

test_sfc_index(p, index)
```

Arguments

`x`	An `sfc_nxn` object.
`i`, `index`	A string of digits representing the path on the hierarchy of the curve. The left side corresponds to the top level and the right side corresponds to the bottom level on the curve. For the 2x2 curve, the digits can only be 1-4, and for the Peano and Meander curves, the digites can be 1-9. The hierarchical index should be specified in a format of `i1:i2:i3:...` where `:` can be replaced by any non-number character. For 2x2 and 3x3 curves, `:` can be omitted and the hierarchical index can be specified as `i1i2i3...`. See the **Examples** section.
`j`	A value of `TRUE` or `FALSE` that controls whether to keep the `sfc_nxn` class or degenerate to the `sfc_sequence` class.
`...`	Ignore.
`drop`	A value of `TRUE` or `FALSE` that controls whether to keep the `sfc_nxn` class or degenerate to the `sfc_sequence` class.
`p`	An `sfc_nxn` object.

Details

`sfc_index()` only works on square curves (i.e. a curve with a single base letter as seed.)

`test_sfc_index()` is a helper function for demonstrating `sfc_index()`.

Value

`sfc_index()` returns an integer vector.

Examples

```
p = sfc_2x2("I", "11111")
p["3:2:1"]
# for 2x2 and 3x3 curves, ":" can be omitted
p["321"]
p["3:2:1", TRUE] # or p["3:2:1", drop = FALSE]
# only for testing
p = sfc_2x2("I", "11111")
om = par(no.readonly = TRUE)
par(mfrow = c(2, 2))
test_sfc_index(p, "3")
test_sfc_index(p, "3:2")
test_sfc_index(p, "3:2:1")
test_sfc_index(p, "3:2:1:1")
par(om)

p = sfc_3x3_meander("I", "11111")
om = par(no.readonly = TRUE)
par(mfrow = c(2, 2))
test_sfc_index(p, "7")
test_sfc_index(p, "7:5")
test_sfc_index(p, "7:5:9")
test_sfc_index(p, "7:5:9:2")
```

```
par(om)
```

`[.sfc_sequence` *Utility functions*

Description

Utility functions

Usage

```
## S3 method for class 'sfc_sequence'
x[i]

## S3 replacement method for class 'sfc_sequence'
x[i] <- value

## S3 method for class 'sfc_sequence'
length(x)

## S3 method for class 'sfc_sequence'
c(...)
```

Arguments

`x`	An `sfc_sequence` object.
`i`	Numeric index.
`value`	An `sfc_sequence` object.
`...`	A list of `sfc_sequence` objects or other arguments.

Details

For efficiency, `c.sfc_sequence()` does not check whether the input `sfc_sequence` objects are compatible.

Value

`length.sfc_sequence()` returns an integer scalar.

`c.sfc_sequence()` returns an `sfc_sequence` object.

Examples

```
p = sfc_sequence("ABCDEFGH")
p
p[1:4]
p[1:4] = p[8:5]
p
length(p)
c(p, p)
```

Index