Package ‘gridpattern’

August 16, 2022

Type Package

Title 'grid' Pattern Grobs

Version 1.0.1

Description Provides 'grid' grobs that fill in a user-defined area with various patterns. Includes enhanced versions of the geometric and image-based patterns originally contained in the 'ggpattern' package as well as original 'pch', 'polygon_tiling', 'regular_polygon', 'rose', 'text', 'wave', and 'weave' patterns plus support for custom user-defined patterns.

URL https://trevorldavis.com/R/gridpattern/,
https://github.com/trevorld/gridpattern

BugReports https://github.com/trevorld/gridpattern/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.1

Depends R (>= 3.4.0)

Imports glue, grDevices, grid, memoise, png, rlang, sf, utils

Suggests ambient, knitr, magick, ragg (>= 1.2.0), rmarkdown, svglite (>= 2.1.0), testthat, vdiffr

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Author Mike FC [aut] (Code/docs adapted from ggpattern),
Trevor L Davis [aut, cre] (<https://orcid.org/0000-0001-6341-4639>),
Thomas Lin Pedersen [ctb] (new_data_frame() copied from ggplot2)

Maintainer Trevor L Davis <trevor.l.davis@gmail.com>

Repository CRAN

Date/Publication 2022-08-16 18:40:02 UTC

R topics documented:

`alphaMaskGrob` *Mask grob using another grob to specify the (alpha) mask*

Description

`alphaMaskGrob()` masks a grob using another grob to specify the (alpha) mask.

Usage

```
alphaMaskGrob(
  maskee,
  masker,
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  name = NULL,
  gp = gpar(),
```

```
  vp = NULL
)
```

Arguments

- `maskee` Grob to be masked
- `masker` Grob that defines masking region
- `use_R4.1_masks` If `TRUE` use the grid mask feature introduced in R v4.1.0. If `FALSE` do a `rasterGrob` approximation. If `NULL` try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.
- `png_device` "png" graphics device to save intermediate raster data with if `use_R4.1_masks` is `FALSE`. If `NULL` and suggested package `ragg` is available and versions are high enough we directly capture masked raster via `ragg::agg_capture()`. Otherwise we will use png_device (default `ragg::agg_png()` if available else `grDevices::png()`) and `png::readPNG()` to manually compute a masked raster.
- `res` Resolution of desired `rasterGrob` in pixels per inch if `use_R4.1_masks` is `FALSE`.
- `name` A character identifier.
- `gp` An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
- `vp` A Grid viewport object (or NULL).

Value

A `grid` grob

Examples

```
 # Once took more >10s on a CRAN autocheck
if (capabilities("png") && require("grid")) {
  maskee <- patternGrob("circle", gp = gpar(col = "black", fill = "yellow"),
                        spacing = 0.1, density = 0.5)
  angle <- seq(2 * pi / 4, by = 2 * pi / 6, length.out = 7)
  x_hex_outer <- 0.5 + 0.5 * cos(angle)
  y_hex_outer <- 0.5 + 0.5 * sin(angle)
  x_hex_inner <- 0.5 + 0.25 * cos(rev(angle))
  y_hex_inner <- 0.5 + 0.25 * sin(rev(angle))
  gp <- gpar(lwd = 0, col = NA, fill = "white")
  masker <- grid::pathGrob(x = c(x_hex_outer, x_hex_inner),
                          y = c(y_hex_outer, y_hex_inner),
                          id = rep(1:2, each = 7),
                          rule = "evenodd", gp = gp)
  masked <- alphaMaskGrob(maskee, masker, use_R4.1_masks = FALSE)
  grid.newpage()
  grid.draw(masked)

  maskee_transparent <- rectGrob(gp = gpar(col = NA, fill = "blue"))
  gp <- gpar(lwd = 20, col = "black", fill = grDevices::rgb(0, 0, 0, 0.5))
  masker_transparent <- editGrob(masker, gp = gp)
```

```
  masked_transparent <- alphaMaskGrob(maskee_transparent,
                                      masker_transparent,
                                      use_R4.1_masks = FALSE)
  grid.newpage()
  grid.draw(masked_transparent)
}
```

`clippingPathGrob` *Clip grob using another grob to specify the clipping path*

Description

`clippingPathGrob()` clips a grob using another grob to specify the clipping path

Usage

```
clippingPathGrob(
  clippee,
  clipper,
  use_R4.1_clipping = getOption("ggpattern_use_R4.1_clipping",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  name = NULL,
  gp = gpar(),
  vp = NULL
)
```

Arguments

`clippee`	Grob to be clipped
`clipper`	Grob that defines clipping region
`use_R4.1_clipping`	If `TRUE` use the grid clipping path feature introduced in R v4.1.0. If `FALSE` do a `rasterGrob` approximation. If `NULL` try to guess an appropriate choice. Note not all graphic devices support the grid clipping path feature and the grid clipping path feature does not nest.
`png_device`	"png" graphics device to use if `use_R4.1_clipping` is `FALSE`. If `NULL` (default) will use `ragg::agg_png()` if the suggested package `ragg` is available else `grDevices::png()`.
`res`	Resolution of desired `rasterGrob` in pixels per inch if `use_R4.1_clipping` is `FALSE`.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`vp`	A Grid viewport object (or NULL).

Value

A `grid` grob

Examples

```
if (capabilities("png") && require("grid")) {
  clippee <- patternGrob("circle", gp = gpar(col = "black", fill = "yellow"),
                         spacing = 0.1, density = 0.5)
  angle <- seq(2 * pi / 4, by = 2 * pi / 6, length.out = 7)
  x_hex_outer <- 0.5 + 0.5 * cos(angle)
  y_hex_outer <- 0.5 + 0.5 * sin(angle)
  x_hex_inner <- 0.5 + 0.25 * cos(rev(angle))
  y_hex_inner <- 0.5 + 0.25 * sin(rev(angle))
  clipper <- grid::pathGrob(x = c(x_hex_outer, x_hex_inner),
                            y = c(y_hex_outer, y_hex_inner),
                            id = rep(1:2, each = 7),
                            rule = "evenodd")
  clipped <- clippingPathGrob(clippee, clipper, use_R4.1_clipping = FALSE)
  grid.newpage()
  grid.draw(clipped)
}
```

`grid.pattern` *Create patterned grobs*

Description

`grid.pattern()` draws patterned shapes onto the graphic device. `patternGrob()` returns the grid grob objects. `names_pattern` is a character vector of builtin patterns.

Usage

```
grid.pattern(
  pattern = "stripe",
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  legend = FALSE,
  prefix = "pattern_",
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_pattern
```

```
patternGrob(
  pattern = "stripe",
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  legend = FALSE,
  prefix = "pattern_",
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

pattern	Name of pattern. See Details section for a list of supported patterns.
x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Pattern parameters.
legend	Whether this is intended to be drawn in a legend or not.
prefix	Prefix to prepend to the name of each of the pattern parameters in `...`. For compatibility with `ggpattern` most underlying functions assume parameters beginning with `pattern_`.
default.units	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
name	A character identifier.
gp	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Format

An object of class `character` of length 17.

Details

Here is a list of the various patterns supported:

ambient Noise array patterns onto the graphic device powered by the `ambient` package. See `grid.pattern_ambient()` for more information.

circle Circle geometry patterns. See grid.pattern_circle() for more information.

crosshatch Crosshatch geometry patterns. See grid.pattern_crosshatch() for more information.

gradient Gradient array/geometry patterns. See grid.pattern_gradient() for more information.

image Image array patterns. See grid.pattern_image() for more information.

magick imagemagick array patterns. See grid.pattern_magick() for more information.

none Does nothing. See grid::grid.null() for more information.

pch Plotting character geometry patterns. See grid.pattern_pch() for more information.

placeholder Placeholder image array patterns. See grid.pattern_placeholder() for more information.

plasma Plasma array patterns. See grid.pattern_plasma() for more information.

polygon_tiling Polygon tiling patterns. See grid.pattern_polygon_tiling() for more information.

regular_polygon Regular polygon patterns. See grid.pattern_regular_polygon() for more information.

rose Rose array/geometry patterns. See grid.pattern_rose() for more information.

stripe Stripe geometry patterns. See grid.pattern_stripe() for more information.

text Text array/geometry patterns. See grid.pattern_text() for more information.

wave Wave geometry patterns. See grid.pattern_wave() for more information.

weave Weave geometry patterns. See grid.pattern_weave() for more information.

Custom geometry-based patterns See https://trevorldavis.com/R/gridpattern/dev/articles/developing-patterns.html for more information.

Custom array-based patterns See https://trevorldavis.com/R/gridpattern/dev/articles/developing-patterns.html for more information.

Value

A grid grob object (invisibly in the case of grid.pattern()). If draw is TRUE then grid.pattern() also draws to the graphic device as a side effect.

See Also

https://coolbutuseless.github.io/package/ggpattern/index.html for more details on the ggpattern package.

Examples

```
print(names_pattern)
 # Once took more >10s on a CRAN autocheck
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
```

```
  # geometry-based patterns
  # 'stripe' pattern
  grid.newpage()
  grid.pattern("stripe", x_hex, y_hex,
               colour="black", fill=c("yellow", "blue"), density = 0.5)

  # Can alternatively use "gpar()" to specify colour and line attributes
  grid.newpage()
  grid.pattern("stripe", x_hex, y_hex, gp = gpar(col="blue", fill="red", lwd=2))

  # 'weave' pattern
  grid.newpage()
  grid.pattern("weave", x_hex, y_hex, type = "satin",
               colour = "black", fill = "lightblue", fill2 =  "yellow",
               density = 0.3)

  # 'regular_polygon' pattern
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, colour = "black",
                               fill = c("blue", "yellow", "red"),
                               shape = c("convex4", "star8", "circle"),
                               density = c(0.45, 0.42, 0.4),
                               spacing = 0.08, angle = 0)

  # can be used to achieve a variety of 'tiling' effects
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, color = "transparent",
                               fill = c("white", "grey", "black"),
                               density = 1.0, spacing = 0.1,
                               shape = "convex6", grid = "hex")
  if (require("magick")) {
    # array-based patterns
    # 'image' pattern
    logo_filename <- system.file("img", "Rlogo.png" , package="png")
    grid.newpage()
    grid.pattern("image", x_hex, y_hex, filename=logo_filename, type="fit")

    # 'plasma' pattern
    grid.newpage()
    grid.pattern("plasma", x_hex, y_hex, fill="green")
  }
}
```

`grid.pattern_ambient` *Ambient patterned grobs*

Description

`grid.pattern_ambient()` draws noise patterns onto the graphic device powered by the `ambient` package.

Usage

```
grid.pattern_ambient(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "simplex",
  fill = gp$fill %||% "grey80",
  fill2 = "#4169E1",
  frequency = 0.01,
  interpolator = "quintic",
  fractal = switch(type, worley = "none", "fbm"),
  octaves = 3,
  lacunarity = 2,
  gain = 0.5,
  pertubation = "none",
  pertubation_amplitude = 1,
  value = "cell",
  distance_ind = c(1, 2),
  jitter = 0.45,
  res = getOption("ggpattern_res", 72),
  alpha = NA_real_,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`type`	Either cubic, perlin, simplex, value, white, or worley
`fill`	Fill colour
`fill2`	Second colour
`frequency`	Determines the granularity of the features in the noise.
`interpolator`	How should values between sampled points be calculated? Either `'linear'`, `'hermite'`, or `'quintic'` (default), ranging from lowest to highest quality.
`fractal`	The fractal type to use. Either `'none'`, `'fbm'` (default), `'billow'`, or `'rigid-multi'`. It is suggested that you experiment with the different types to get a feel for how they behaves.

`octaves` The number of noise layers used to create the fractal noise. Ignored if `fractal = 'none'`. Defaults to `3`.

`lacunarity` The frequency multiplier between successive noise layers when building fractal noise. Ignored if `fractal = 'none'`. Defaults to `2`.

`gain` The relative strength between successive noise layers when building fractal noise. Ignored if `fractal = 'none'`. Defaults to `0.5`.

`pertubation` The pertubation to use. Either `'none'` (default), `'normal'`, or `'fractal'`. Defines the displacement (warping) of the noise, with `'normal'` giving a smooth warping and `'fractal'` giving a more eratic warping.

`pertubation_amplitude` The maximal pertubation distance from the origin. Ignored if `pertubation = 'none'`. Defaults to `1`.

`value` The noise value to return. Either

- `'value'` (default) A random value associated with the closest point
- `'distance'` The distance to the closest point
- `'distance2'` The distance to the nth closest point (n given by `distance_ind[1]`)
- `'distance2add'` Addition of the distance to the nth and mth closest point given in `distance_ind`
- `'distance2sub'` Substraction of the distance to the nth and mth closest point given in `distance_ind`
- `'distance2mul'` Multiplication of the distance to the nth and mth closest point given in `distance_ind`
- `'distance2div'` Division of the distance to the nth and mth closest point given in `distance_ind`

`distance_ind` Reference to the nth and mth closest points that should be used when calculating `value`.

`jitter` The maximum distance a point can move from its start position during sampling of cell points.

`res` Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.

`alpha` Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).

`default.units` A string indicating the default units to use if `x` or `y` are only given as numeric vectors.

`name` A character identifier.

`gp` An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.

`draw` A logical value indicating whether graphics output should be produced.

`vp` A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

For more information about the noise types please see the relevant `ambient` documentation: `ambient::noise_cubic()`, `ambient::noise_perlin()`, `ambient::noise_simplex()`, `ambient::noise_value()`, `ambient::noise_white()`, and `ambient::noise_worley()`. `grid.pattern_plasma()` provides an alternative noise pattern that depends on `magick`.

Examples

```
if (requireNamespace("ambient")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_ambient(x_hex, y_hex, fill = "green", fill2 = "blue")
  grid::grid.newpage()
  grid.pattern_ambient(x_hex, y_hex, fill = "green", fill2 = "blue", type = "cubic")
}
```

`grid.pattern_circle` *Circle patterned grobs*

Description

`grid.pattern_circle()` draws a circle pattern onto the graphic device.

Usage

```
grid.pattern_circle(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  type = NULL,
  subtype = NULL,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
```

```
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`colour`	Stroke colour
`fill`	Fill colour
`angle`	Rotation angle in degrees
`density`	Approx. fraction of area the pattern fills.
`spacing`	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
`xoffset`	Shift pattern along x axis ('snpc' units between 0 and 1).
`yoffset`	Shift pattern along y axis ('snpc' units between 0 and 1).
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
`linetype`	Stroke linetype
`linewidth`	Stroke linewidth
`size`	For backwards compatibility can be used to set `linewidth`
`grid`	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
`type`	Adjusts the repeating of certain aesthetics such as color. Can use any type in `names_hex`, `names_square`, or `names_weave`. See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `type` arguments.
`subtype`	See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `subtype` arguments.
`default.units`	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

See `grid.pattern_regular_polygon()` for a more general case of this pattern.

Examples

```
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_circle(x_hex, y_hex, fill = c("blue", "yellow"), density = 0.5)
  grid.newpage()
  grid.pattern_circle(x_hex, y_hex, density = 0.8, grid = "hex_circle",
                      gp = gpar(fill = c("blue", "yellow", "red")))
  grid.newpage()
  grid.pattern_circle(x_hex, y_hex, density = 1.2, grid = "hex_circle",
                      gp = gpar(fill = c("blue", "yellow", "red")))

  # using a "twill_zigzag" 'weave' pattern
  grid.newpage()
 grid.pattern_circle(x_hex, y_hex, fill = "blue", density = 0.5, type = "twill_zigzag")
}
```

---

grid.pattern_crosshatch

Crosshatch patterned grobs

Description

`grid.pattern_crosshatch()` draws a crosshatch pattern onto the graphic device.

Usage

```
grid.pattern_crosshatch(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  fill2 = fill,
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
```

```
    grid = "square",
    default.units = "npc",
    name = NULL,
    gp = gpar(),
    draw = TRUE,
    vp = NULL
  )
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Currently ignored
colour	Stroke colour
fill	Fill colour
fill2	The fill colour for the “top” crosshatch lines.
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
alpha	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
linetype	Stroke linetype
linewidth	Stroke linewidth
size	For backwards compatibility can be used to set `linewidth`
grid	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
default.units	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
name	A character identifier.
gp	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

`grid.pattern_weave()` which interweaves two sets of lines. For a single set of lines use `grid.pattern_stripe()`.

Examples

```
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_crosshatch(x_hex, y_hex, colour = "black", fill = "blue",
                          fill2 = "yellow", density = 0.5)
  grid.newpage()
  grid.pattern_crosshatch(x_hex, y_hex, density = 0.3,
                          gp = gpar(col = "blue", fill = "yellow"))
}
```

`grid.pattern_gradient` *Gradient patterned grobs*

Description

`grid.pattern_gradient()` draws a gradient pattern onto the graphic device.

Usage

```
grid.pattern_gradient(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  fill = gp$fill %||% "grey80",
  fill2 = "#4169E1",
  orientation = "vertical",
  alpha = gp$alpha %||% NA_real_,
  use_R4.1_gradients = getOption("ggpattern_use_R4.1_gradients",
    getOption("ggpattern_use_R4.1_features")),
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored
fill	Fill colour
fill2	Second colour
orientation	vertical, horizontal, or radial
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
use_R4.1_gradients	Whether to use the gradient feature introduced in R v4.1 or use a rasterGrob approximation. Note not all graphic devices support the grid gradient feature.
aspect_ratio	Override aspect ratio
key_scale_factor	Additional scale factor for legend
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

Examples

```
if (require("grid") && require("magick") && capabilities("png")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_gradient(x_hex, y_hex, fill = "green")
  grid.newpage()
  grid.pattern_gradient(x_hex, y_hex, fill = "green", orientation = "radial")
}
```

grid.pattern_image *Image patterned grobs*

Description

grid.pattern_image() draws an image pattern onto the graphic device.

Usage

```
grid.pattern_image(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  filename = "",
  type = "fit",
  scale = 1,
  gravity = switch(type, tile = "southwest", "center"),
  filter = "lanczos",
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored
filename	Image of filename or URL
type	Image scaling type
scale	Extra scaling
gravity	Position of image within area. magick::gravity_types() returns a vector of supported values.
filter	Filter to use when scaling. magick::filter_types() returns a vector of supported values.

alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
aspect_ratio	Override aspect ratio
key_scale_factor	Additional scale factor for legend
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Details

Here is a description of the type arguments:

expand Scale the image beyond the bounding box and crop it such that the image fully covers the width and the height of the region.

fit Scale the image such that either the width or the height of the image fits in the bounding box. Affected by gravity

none Position a single image in the region without attempting to scale to the bounding box size. Affected by scale and gravity.

squish Distort the image to cover the bounding box of the region.

tile Repeat the image to cover the bounding box. Affected by tile.

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_placeholder() is an image pattern that uses images downloaded from the internet.

Examples

```
if (require("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  logo_filename <- system.file("img", "Rlogo.png" , package = "png")
  grid.pattern_image(x_hex, y_hex, filename = logo_filename, type = "fit")

  # "tile" `type` image pattern depends on `magick` functionality
  # which is not reliable across platforms
  grid::grid.newpage()
  try(grid.pattern_image(x_hex, y_hex, filename = logo_filename,
                         type = "tile"))
}
```

grid.pattern_magick *Magick patterned grobs*

Description

grid.pattern_magick() draws a imagemagick pattern onto the graphic device. names_magick, names_magick_intensity, and names_magick_stripe are character vectors of supported type values plus subsets for shaded intensity and stripes.

Usage

```
grid.pattern_magick(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "hexagons",
  fill = "grey20",
  scale = 1,
  filter = "box",
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_magick

names_magick_intensity

names_magick_stripe
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored

`type`	Magick pattern types. `names_magick`, `names_magick_intensity`, and `names_magick_stripe` are character vectors of supported `type` values plus subsets for shaded intensity and stripes.
`fill`	Fill colour
`scale`	Extra scaling
`filter`	Filter to use when scaling. `magick::filter_types()` returns a vector of supported values.
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
`aspect_ratio`	Override aspect ratio
`key_scale_factor`	Additional scale factor for legend
`res`	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
`default.units`	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Format

An object of class `character` of length 54.

An object of class `character` of length 21.

An object of class `character` of length 19.

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

The `imagemagick` documentation http://www.imagemagick.org/script/formats.php for more information.

Examples

```
if (require("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_magick(x_hex, y_hex, type="octagons", fill="blue", scale=2)
}

# supported magick pattern names
print(names_magick)
```

`grid.pattern_pch` *Plotting character patterned grobs*

Description

`grid.pattern_pch()` draws a plotting character pattern onto the graphic device.

Usage

```
grid.pattern_pch(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  scale = 0.5,
  shape = 1L,
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`colour`	Stroke colour

fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
scale	For star polygons, multiplier (between 0 and 1) applied to exterior radius to get interior radius.
shape	An integer from `0` to `25` or `NA`. See `graphics::points()` for more details. Note we only support these shapes and do not support arbitrary ASCII / Unicode characters.
grid	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in `names_hex`, `names_square`, or `names_weave`. See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `type` arguments.
subtype	See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `subtype` arguments.
rot	Angle to rotate regular polygon (degrees, counter-clockwise).
alpha	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
linetype	Stroke linetype
linewidth	Stroke linewidth
size	For backwards compatibility can be used to set `linewidth`
default.units	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
name	A character identifier.
gp	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

`grid.pattern_regular_polygon()` which is used to implement this pattern.

Examples

```
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  gp <- gpar(col = "black", fill = "lightblue")

  # pch 0-6 are simple shapes with no fill
  grid.pattern_pch(x_hex, y_hex, shape = 0:6, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)

  # pch 7-14 are compound shapes with no fill
  grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 7:14, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)

  # pch 15-20 are filled with 'col'
  grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 15:20, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)

  # pch 21-25 are filled with 'fill'
  grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = 21:25, gp = gp,
                   spacing = 0.1, density = 0.4, angle = 0)

  # using a 'basket' weave `type` with two shapes
  grid.newpage()
  grid.pattern_pch(x_hex, y_hex, shape = c(1,4), gp = gp,
                   type = "basket",
                   spacing = 0.1, density = 0.4, angle = 0)
}
```

`grid.pattern_placeholder` *Placeholder image patterned grobs*

Description

`grid.pattern_placeholder()` draws a placeholder image pattern onto the graphic device. `names_placeholder` are character vectors of supported placeholder types.

Usage

```
grid.pattern_placeholder(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  type = "kitten",
```

```
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_placeholder
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored
type	Image source. names_placeholder is a vector of supported values. If you would like only greyscale images append bw to the name.
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
aspect_ratio	Override aspect ratio
key_scale_factor	Additional scale factor for legend
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Format

An object of class character of length 26.

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

Examples

```
if (require("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  # requires internet connection to download from placeholder image websites
  try(grid.pattern_placeholder(x_hex, y_hex, type="bear"))
}

print(names_placeholder)
```

---

grid.pattern_plasma *Plasma patterned grobs*

Description

`grid.pattern_plasma()` draws a plasma pattern onto the graphic device.

Usage

```
grid.pattern_plasma(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  fill = gp$fill %||% "grey80",
  scale = 1,
  alpha = gp$alpha %||% NA_real_,
  aspect_ratio = 1,
  key_scale_factor = 1,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`fill`	Fill colour
`scale`	Extra scaling

alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
aspect_ratio	Override aspect ratio
key_scale_factor	Additional scale factor for legend
res	Assumed resolution (in pixels per graphic device inch) to use when creating array pattern.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

grid.pattern_ambient() provides a noise pattern using the ambient package.

Examples

```
if (require("magick")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_plasma(x_hex, y_hex, fill = "green")
}
```

---

grid.pattern_polygon_tiling
Polygon tiling patterned grobs

Description

grid.pattern_polygon_tiling() draws a specified polygon tiling pattern onto the graphic device. names_polygon_tiling lists all supported types.

Usage

```
grid.pattern_polygon_tiling(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  type = "square",
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)

names_polygon_tiling
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`colour`	Stroke colour
`fill`	Fill colour
`angle`	Rotation angle in degrees
`spacing`	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
`xoffset`	Shift pattern along x axis ('snpc' units between 0 and 1).
`yoffset`	Shift pattern along y axis ('snpc' units between 0 and 1).
`type`	Name of polygon tiling to draw. See Details.
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value). Not supported for all polygon tiling `type`.
`linetype`	Stroke linetype
`linewidth`	Stroke linewidth

`size` For backwards compatibility can be used to set `linewidth`

`default.units` A string indicating the default units to use if `x` or `y` are only given as numeric vectors.

`name` A character identifier.

`gp` An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.

`draw` A logical value indicating whether graphics output should be produced.

`vp` A Grid viewport object (or NULL).

Format

An object of class `character` of length 36.

Details

`grid.pattern_polygon_tiling()` supports 1, 2, or 3 `fill` colors with the first colors (weakly) covering a larger area. Size of the pattern is controlled by `spacing`. We support the following polygon tiling `types`:

`elongated_triangular` Creates an elongated triangular tiling made of squares and triangles.

`herringbone` Creates a herringbone tiling made of rectangles.

`hexagonal` Creates a hexagonal tiling made of hexagons.

`pythagorean` Creates a Pythagorean tiling made of squares of two different sizes.

`rhombille` Creates a rhombille tiling made of rhombi.

`rhombitrihexagonal` Creates a rhombitrihexagonal tiling made out of dodecagons, hexagons, and squares.

`snub_square` Creates a snub square tiling made of squares and triangles.

`snub_trihexagonal` Creates a snub trihexagonal tiling made of hexagons and triangles.

`square` Creates a square tiling made of squares.

`tetrakis_square` Creates a tetrakis square tiling made of isosceles right triangles.

`triangular` Creates a triangular tiling made of equilateral triangles.

`trihexagonal` Creates a trihexagonal tiling made of hexagons and triangles.

`truncated_square` Creates a truncated square tiling made of octagons and squares.

`truncated_hexagonal` Creates a truncated hexagonal tiling made of dodecagons and triangles.

`truncated_trihexagonal` Creates a truncated trihexagonal tiling made of hexagons, squares, and triangles.

`2*.2**.2*.2**` Creates a polygon tiling made of rhombi.

`2**.3**.12*` Creates a polygon tiling made of rhombi, triangles, and twelve-pointed stars.

`3.3.3.3**` Creates a polygon tiling made of triangles.

`3.3*.3.3**` Creates a regular (star) polygon tiling made of triangles and three-pointed stars.

`3.3.3.12*.3.3.12*` Creates a regular (star) polygon tiling made of triangles and twelve-pointed stars.

3.3.8*.3.4.3.8* Creates a regular (star) polygon tiling made of triangles, squares, and eight-pointed stars.

3.3.8*.4**.8* Creates a regular (star) polygon tiling made of triangles, four-pointed stars, and eight-pointed stars.

3.4.6.3.12* Creates a regular (star) polygon tiling made of triangles, squares, hexagons, and twelve-pointed stars.

3.4.8.3.8* Creates a regular (star) polygon tiling made of triangles, squares, octagons, and eight-pointed stars.

3.6*.6** Creates a regular (star) polygon tiling made of triangles and six-pointed stars.

4.2*.4.2** Creates a polygon tiling made of squares and rhombi.

4.4*.4** Creates a regular (star) polygon tiling made of squares and four-pointed stars.

4.6.4*.6 Creates a regular (star) polygon tiling made of squares, hexagons, and four-pointed stars.

4.6*.4.6*.4.6* Creates a regular (star) polygon tiling made of squares and six-pointed stars.

4.8*.4**.8* Creates a polygon tiling of squares and eight-pointed stars.

6.6*.6.6* Creates a regular (star) polygon tiling made of hexagons and six-pointed stars.

8.4*.8.4* Creates a regular (star) polygon tiling made of octagons and four-pointed stars.

9.3.9.3* Creates a regular (star) polygon tiling made of triangles, nonagons, and three-pointed stars.

12.3*.12.3* Creates a regular (star) polygon tiling made of dodecagons and three-pointed stars.

12.12.4* Creates a regular (star) polygon tiling made of dodecagons and four-pointed stars.

18.18.3* Creates a regular (star) polygon tiling made of eighteen-sided polygons and three-pointed stars.

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

The tiling vignette `vignette("tiling", package = "gridpattern")` for more information about these tilings as well as more examples of polygon tiling using the `grid.pattern_regular_polygon()` function.

Examples

```
print(names_polygon_tiling)
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  gp1 <- gpar(fill = "yellow", col = "black")
  gp2 <- gpar(fill = c("yellow", "red"), col = "black")
  gp3 <- gpar(fill = c("yellow", "red", "blue"), col = "black")

  grid.pattern_polygon_tiling(x_hex, y_hex, type = "herringbone", gp = gp1)
```

```
  grid.newpage()
  grid.pattern_polygon_tiling(x_hex, y_hex, type = "hexagonal",
                              spacing = 0.2, gp = gp3)

  grid.newpage()
  grid.pattern_polygon_tiling(x_hex, y_hex, type = "pythagorean",
                              spacing = 0.2, gp = gp2)

  grid.newpage()
  grid.pattern_polygon_tiling(x_hex, y_hex, type = "snub_trihexagonal",
                              spacing = 0.2, gp = gp3)

  grid.newpage()
  grid.pattern_polygon_tiling(x_hex, y_hex, type = "rhombille",
                              spacing = 0.2, gp = gp3)
}
```

---

grid.pattern_regular_polygon *Regular polygon patterned grobs*

Description

grid.pattern_regular_polygon() draws a regular polygon pattern onto the graphic device.

Usage

```
grid.pattern_regular_polygon(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  scale = 0.5,
  shape = "convex4",
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
```

```
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Currently ignored
colour	Stroke colour
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
scale	For star polygons, multiplier (between 0 and 1) applied to exterior radius to get interior radius.
shape	Either "convex" or "star" followed by the number of exterior vertices or alternatively "circle", "square", "null", "rhombille_rhombus", "tetrakis_left", or "tetrakis_right". For example "convex5" corresponds to a pentagon and "star6" corresponds to a six-pointed star. The "square" shape is larger than the "convex4" shape and is rotated an extra 45 degrees, it can be used to generate a multi-colored "checkers" effect when density is 1. The "null" shape is not drawn, it can be used to create holes within multiple-element patterns. The "rhombille_rhombus" shape draws a rhombus while the "tetrakis_left" or "tetrakis_right" shapes draw an isosceles right triangle. These latter three non-regular-polygon shapes are intended to help generate rhombille and tetrakis square tilings.
grid	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in `names_hex`, `names_square`, or `names_weave`. See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `type` arguments.
subtype	See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `subtype` arguments.

`rot`	Angle to rotate regular polygon (degrees, counter-clockwise).
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
`linetype`	Stroke linetype
`linewidth`	Stroke linewidth
`size`	For backwards compatibility can be used to set `linewidth`
`default.units`	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

`grid.pattern_circle()` for a special case of this pattern. The tiling vignette features more examples of regular polygon tiling using this function `vignette("tiling", package = "gridpattern")`.

Examples

```
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))

  # 'density', 'rot', and 'shape' are vectorized
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, colour = "black",
                               fill = c("blue", "yellow", "red"),
                               shape = c("convex4", "star8", "circle"),
                               density = c(0.45, 0.42, 0.4),
                               spacing = 0.08, angle = 0)

  # checker pattern using "square" shape
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, shape = "square",
                               colour = "transparent",
                               fill = c("black", "red", "blue", "yellow"),
                               angle = 0, density = 1.0, spacing = 0.2)

  # checker pattern using the default "convex4" shape
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, density = 1.0,
                               colour = "black", fill = "blue")
```

```
  # using a "twill_zigzag" 'weave' pattern
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, fill = c("blue", "yellow"),
                               shape = c("circle", "star8"),
                               density = c(0.5, 0.6), type = "twill_zigzag")

  # hexagon tiling
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, color = "transparent",
                               fill = c("white", "grey", "black"),
                               density = 1.0, spacing = 0.1,
                               shape = "convex6", grid = "hex")

  # triangle tiling
  grid.newpage()
  grid.pattern_regular_polygon(x_hex, y_hex, fill = "green",
                               density = 1.0, spacing = 0.1,
                               shape = "convex3", grid = "hex")

}
```

`grid.pattern_rose` *Rose curve patterned grobs*

Description

`grid.pattern_rose()` draws a rose curve pattern onto the graphic device.

Usage

```
grid.pattern_rose(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  frequency = 0.1,
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
```

```
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
...	Currently ignored
colour	Stroke colour
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
frequency	The "angular frequency" parameter of the rose pattern.
grid	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in `names_hex`, `names_square`, or `names_weave`. See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `type` arguments.
subtype	See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `subtype` arguments.
rot	Angle to rotate rose (degrees, counter-clockwise).
alpha	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
linetype	Stroke linetype
linewidth	Stroke linewidth
size	For backwards compatibility can be used to set `linewidth`

use_R4.1_masks — If TRUE use the grid mask feature introduced in R v4.1.0. If FALSE do a rasterGrob approximation. If NULL try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.

png_device — "png" graphics device to save intermediate raster data with if use_R4.1_masks is FALSE. If NULL and suggested package ragg is available and versions are high enough we directly capture masked raster via ragg::agg_capture(). Otherwise we will use png_device (default ragg::agg_png() if available else grDevices::png()) and png::readPNG() to manually compute a masked raster.

res — Resolution of desired rasterGrob in pixels per inch if use_R4.1_masks is FALSE.

default.units — A string indicating the default units to use if x or y are only given as numeric vectors.

name — A character identifier.

gp — An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.

draw — A logical value indicating whether graphics output should be produced.

vp — A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If draw is TRUE then also draws to the graphic device as a side effect.

See Also

See https://en.wikipedia.org/wiki/Rose_(mathematics) for more information.

Examples

```
if (require("grid") && capabilities("png")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  gp <- gpar(fill = c("blue", "red", "yellow", "green"), col = "black")

  grid.newpage()
  grid.pattern_rose(x_hex, y_hex,
                    spacing = 0.15, density = 0.5, angle = 0,
                    frequency = 1:4, gp = gp)
  grid.newpage()
  grid.pattern_rose(x_hex, y_hex,
                    spacing = 0.15, density = 0.5, angle = 0,
                    frequency = 1/1:4, gp = gp)
  grid.newpage()
  grid.pattern_rose(x_hex, y_hex,
                    spacing = 0.18, density = 0.5, angle = 0,
                    frequency = c(3/2, 7/3, 5/4, 3/7), gp = gp)
}
```

grid.pattern_stripe *Stripe patterned grobs*

Description

grid.pattern_stripe() draws a stripe pattern onto the graphic device.

Usage

```
grid.pattern_stripe(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored
colour	Stroke colour
fill	Fill colour
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).

`xoffset`	Shift pattern along x axis ('snpc' units between 0 and 1).
`yoffset`	Shift pattern along y axis ('snpc' units between 0 and 1).
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
`linetype`	Stroke linetype
`linewidth`	Stroke linewidth
`size`	For backwards compatibility can be used to set `linewidth`
`grid`	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
`default.units`	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

`[grid.pattern_crosshatch()]` and `[grid.pattern_weave()]` for overlaying stripes.

Examples

```
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.pattern_stripe(x_hex, y_hex, colour = "black",
                      fill = c("red", "blue"), density = 0.4)

  # Can alternatively use "gpar()" to specify colour and line attributes
  grid.newpage()
  grid.pattern_stripe(x_hex, y_hex, density = 0.3,
                      gp = gpar(col = "blue", fill = "yellow"))
}
```

grid.pattern_text *Text character patterned grobs*

Description

grid.pattern_text() draws a text character pattern onto the graphic device.

Usage

```
grid.pattern_text(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  angle = 30,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  scale = 0.5,
  shape = "X",
  grid = "square",
  type = NULL,
  subtype = NULL,
  rot = 0,
  alpha = gp$alpha %||% NA_real_,
  size = gp$fontsize %||% 12,
  fontfamily = gp$fontfamily %||% "sans",
  fontface = gp$fontface %||% "plain",
  use_R4.1_masks = getOption("ggpattern_use_R4.1_masks",
    getOption("ggpattern_use_R4.1_features")),
  png_device = NULL,
  res = getOption("ggpattern_res", 72),
  default.units = "npc",
  name = NULL,
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.

...	Currently ignored
colour	Stroke colour
angle	Rotation angle in degrees
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
scale	For star polygons, multiplier (between 0 and 1) applied to exterior radius to get interior radius.
shape	A character or expression vector. See `label` argument of `grid::textGrob()` for more details.
grid	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
type	Adjusts the repeating of certain aesthetics such as color. Can use any type in `names_hex`, `names_square`, or `names_weave`. See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `type` arguments.
subtype	See for `pattern_hex()`, `pattern_square()`, and `pattern_weave()` for more information about supported `subtype` arguments.
rot	Angle to rotate regular polygon (degrees, counter-clockwise).
alpha	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
size	Fontsize
fontfamily	The font family. See `grid::gpar()` for more details.
fontface	The font face. See `grid::gpar()` for more details.
use_R4.1_masks	If `TRUE` use the grid mask feature introduced in R v4.1.0. If `FALSE` do a `rasterGrob` approximation. If `NULL` try to guess an appropriate choice. Note not all graphic devices support the grid mask feature.
png_device	"png" graphics device to save intermediate raster data with if `use_R4.1_masks` is `FALSE`. If `NULL` and suggested package `ragg` is available and versions are high enough we directly capture masked raster via `ragg::agg_capture()`. Otherwise we will use `png_device` (default `ragg::agg_png()` if available else `grDevices::png()`) and `png::readPNG()` to manually compute a masked raster.
res	Resolution of desired `rasterGrob` in pixels per inch if `use_R4.1_masks` is `FALSE`.
default.units	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
name	A character identifier.
gp	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

Examples

```
if (require("grid") && capabilities("png")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))

  playing_card_symbols <- c("\u2660", "\u2665", "\u2666", "\u2663")
  grid.newpage()
  grid.pattern_text(x_hex, y_hex,
                   shape = playing_card_symbols,
                   colour = c("black", "red", "red", "black"),
                   size = 18, spacing = 0.1, angle = 0)
}
```

`grid.pattern_wave` *Wave patterned grobs*

Description

`grid.pattern_wave()` draws a wave pattern onto the graphic device.

Usage

```
grid.pattern_wave(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  amplitude = 0.5 * spacing,
  frequency = 1/spacing,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
  grid = "square",
  type = "triangle",
  default.units = "npc",
  name = NULL,
```

```
  gp = gpar(),
  draw = TRUE,
  vp = NULL
)
```

Arguments

`x`	A numeric vector or unit object specifying x-locations of the pattern boundary.
`y`	A numeric vector or unit object specifying y-locations of the pattern boundary.
`id`	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same `id` belong to the same boundary.
`...`	Currently ignored
`colour`	Stroke colour
`fill`	Fill colour
`angle`	Rotation angle in degrees
`density`	Approx. fraction of area the pattern fills.
`spacing`	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
`xoffset`	Shift pattern along x axis ('snpc' units between 0 and 1).
`yoffset`	Shift pattern along y axis ('snpc' units between 0 and 1).
`amplitude`	Wave amplitude (“snpc” units)
`frequency`	Linear frequency (inverse “snpc” units)
`alpha`	Alpha (between 0 and 1) or `NA` (default, preserves colors' alpha value).
`linetype`	Stroke linetype
`linewidth`	Stroke linewidth
`size`	For backwards compatibility can be used to set `linewidth`
`grid`	Adjusts placement and density of certain graphical elements. `"square"` (default) is a square grid. `"hex"` is a hexagonal grid suitable for hexagonal and triangular tiling. `"hex_circle"` is a hexagonal grid suitable for circle packing. `"elongated_triangle"` is a grid used for the "elongated triangle" tiling.
`type`	Either “sine” or “triangle” (default).
`default.units`	A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
`name`	A character identifier.
`gp`	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
`draw`	A logical value indicating whether graphics output should be produced.
`vp`	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

Use `grid.pattern_stripe()` for straight lines instead of waves.

Examples

```
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  grid.newpage()
  grid.pattern_wave(x_hex, y_hex, colour = "black", type = "sine",
                    fill = c("red", "blue"), density = 0.4,
                    spacing = 0.15, angle = 0,
                    amplitude = 0.05, frequency = 1 / 0.20)

  # zig-zag pattern is a wave of `type` "triangle"
  grid.newpage()
  grid.pattern_wave(x_hex, y_hex, colour = "black", type = "triangle",
                      fill = c("red", "blue"), density = 0.4,
                      spacing = 0.15, angle = 0, amplitude = 0.075)

}
```

`grid.pattern_weave` *Weave patterned grobs*

Description

`grid.pattern_weave()` draws a weave pattern onto the graphic device.

Usage

```
grid.pattern_weave(
  x = c(0, 0, 1, 1),
  y = c(1, 0, 0, 1),
  id = 1L,
  ...,
  colour = gp$col %||% "grey20",
  fill = gp$fill %||% "grey80",
  fill2 = fill,
  angle = 30,
  density = 0.2,
  spacing = 0.05,
  xoffset = 0,
  yoffset = 0,
  alpha = gp$alpha %||% NA_real_,
  linetype = gp$lty %||% 1,
  linewidth = size %||% gp$lwd %||% 1,
  size = NULL,
```

```
    grid = "square",
    type = "plain",
    subtype = NA,
    default.units = "npc",
    name = NULL,
    gp = gpar(),
    draw = TRUE,
    vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-locations of the pattern boundary.
y	A numeric vector or unit object specifying y-locations of the pattern boundary.
id	A numeric vector used to separate locations in x, y into multiple boundaries. All locations within the same id belong to the same boundary.
...	Currently ignored
colour	Stroke colour
fill	The fill colour for the horizontal "weft" lines.
fill2	The fill colour for the vertical "warp" lines.
angle	Rotation angle in degrees
density	Approx. fraction of area the pattern fills.
spacing	Spacing between repetitions of pattern ('snpc' units between 0 and 1).
xoffset	Shift pattern along x axis ('snpc' units between 0 and 1).
yoffset	Shift pattern along y axis ('snpc' units between 0 and 1).
alpha	Alpha (between 0 and 1) or NA (default, preserves colors' alpha value).
linetype	Stroke linetype
linewidth	Stroke linewidth
size	For backwards compatibility can be used to set linewidth
grid	Adjusts placement and density of certain graphical elements. "square" (default) is a square grid. "hex" is a hexagonal grid suitable for hexagonal and triangular tiling. "hex_circle" is a hexagonal grid suitable for circle packing. "elongated_triangle" is a grid used for the "elongated triangle" tiling.
type	The weave type. See pattern_weave() for more details.
subtype	The weave subtype. See pattern_weave() for more details.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
name	A character identifier.
gp	An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw	A logical value indicating whether graphics output should be produced.
vp	A Grid viewport object (or NULL).

Value

A grid grob object invisibly. If `draw` is `TRUE` then also draws to the graphic device as a side effect.

See Also

`pattern_weave()`

Examples

```
if (require("grid")) {
  x_hex <- 0.5 + 0.5 * cos(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  y_hex <- 0.5 + 0.5 * sin(seq(2 * pi / 4, by = 2 * pi / 6, length.out = 6))
  gp <- gpar(colour = "black", fill = "lightblue", lwd=0.5)

  # Plain weave (default weave)
  grid.pattern_weave(x_hex, y_hex, fill2 = "yellow",
                     gp = gp, spacing = 0.1, density = 0.3)

  # Irregular matt weave
  grid.newpage()
  grid.pattern_weave(x_hex, y_hex,  type = "matt_irregular",
                     fill2 = "yellow", gp = gp, spacing = 0.1, density = 0.3)

  # Twill weave
  grid.newpage()
  grid.pattern_weave(x_hex, y_hex, type = "twill",
                     fill2 = "yellow", gp = gp, spacing = 0.1, density = 0.3)

  # Zig-zag twill
  grid.newpage()
  grid.pattern_weave(x_hex, y_hex, type = "twill_zigzag",
                     fill2 = "yellow", gp = gp, spacing = 0.05, density = 0.7)

  # Herringbone twill with density 1
  grid.newpage()
  gp$col <- NA
  grid.pattern_weave(x_hex, y_hex, type = "twill_herringbone",
                     fill2 = "yellow", gp = gp, spacing = 0.05, density = 1.0)

}
```

---

guess_has_R4.1_features

Guess whether "active" graphics device supports the grid graphics features introduced in R v4.1.

Description

`guess_has_R_4.1_features()` guesses whether "active" graphics device supports the grid graphics features introduced in R v4.1. If it guesses it does it returns `TRUE` else `FALSE`.

Usage

```
guess_has_R4.1_features(
  features = c("clippingPaths", "gradients", "masks", "patterns")
)
```

Arguments

`features`	Character vector of features to guess support for. Will return `TRUE` only if guesses support for all requested features. **"clippingPaths"** Supports clipping path feature **"gradients"** Supports (both linear and radial) gradient feature **"masks"** Supports (alpha) mask feature **"patterns"** Supports (tiling) pattern feature

Value

`TRUE` if we guess all `features` are supported else `FALSE`

See Also

https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/definitions/definitions.html for more info about the new grid graphics features introduced in R v4.1.

Examples

```
# If R version (weakly) greater than 4.1 should be TRUE
pdf(tempfile(fileext = ".pdf"))
print(guess_has_R4.1_features())
invisible(dev.off())

# Should be FALSE
postscript(tempfile(fileext = ".ps"))
print(guess_has_R4.1_features())
invisible(dev.off())
```

`mean_col` *Compute average color*

Description

`mean_col()` computes an average color.

Usage

```
mean_col(...)
```

Arguments

`...`	Colors to average

Details

We currently compute an average color by using the quadratic mean of the colors' RGBA values.

Value

A color string of 9 characters: `"#"` followed by the red, blue, green, and alpha values in hexadecimal.

Examples

```
mean_col("black", "white")
mean_col(c("black", "white"))
mean_col("red", "blue")
```

`pattern_hex` *Hex pattern matrix*

Description

`pattern_hex()` returns an integer matrix indicating where each color (or other graphical element) should be drawn on a (horizontal) hex grid for a specified hex pattern type and subtype. `names_hex` lists the currently supported hex `types`.

Usage

```
pattern_hex(type = "hex", subtype = NULL, nrow = 5L, ncol = 5L)

names_hex
```

Arguments

`type`	Currently just supports "hex".
`subtype`	An integer indicating number of colors (or other graphical elements).
`nrow`	Number of rows (height).
`ncol`	Number of columns (width).

Format

An object of class `character` of length 5.

Details

"hex" Attempts to use a uniform coloring if it exists. For subtype `1L`, `2L`, and `3L` we use the "hex1" pattern. For subtype `4L` we use the "hex2" pattern. For subtype `7L` we use the "hex3" pattern. Else a uniform coloring does not exist and we use the "hex_skew" pattern.

"hex1" Provides the 1-uniform colorings of a hexagonal tiling. Only exists for `subtype` `1L`, `2L`, or `3L`.

"hex2" Provides the 2-uniform colorings of a hexagonal tiling. Only exists for `subtype` `2L` or `4L`.

"hex3" Provides the 3-uniform colorings of a hexagonal tiling. Only exists for `subtype` `2L` or `7L`.

"hex_skew" For the "hex_skew" `type` we cycle through `subtype` elements on the horizontal line and "main" diagonal line. For some `subtype` numbers this may lead to noticeable color repeats on the "skew" diagonal line. If `subtype` is strictly greater than `2L` then a hexagon should never touch another hexagon of the same color.

Value

A matrix of integer values indicating where the each color or other graphical elements should be drawn on a horizontal hex grid (i.e. hexagons are assumed to be pointy side up). Indices `[1,1]` of the matrix corresponds to the bottom-left of the grid while indices `[1,ncol]` corresponds to the bottom-right of the grid. The even rows are assumed to be on the **left** of the ones on the odd rows (for those in the same column in the matrix). This matrix has a "pattern_hex" subclass which supports a special `print()` method.

See Also

`grid.pattern_regular_polygon()` for drawing to a graphics device hexagons, triangles, circles, etc. in hexagon patterns. The tiling vignette features several examples of regular polygon tiling using this both the "hex" and "hex_circle" types `vignette("tiling", package = "gridpattern")`. For more information on uniform colorings of a hexagonal tiling see https://en.wikipedia.org/wiki/Hexagonal_tiling#Uniform_colorings.

Examples

```
# supported hex names
print(names_hex)

# 1-uniform 3-color
hex_3color <- pattern_hex("hex1", 3L, nrow = 7L, ncol = 9L)
print(hex_3color)

# 2-uniform 4-color
hex_4color <- pattern_hex("hex2", 4L, nrow = 7L, ncol = 9L)
print(hex_4color)
```

`pattern_square` *Square pattern matrix*

Description

`pattern_square()` returns an integer matrix indicating where each color (or other graphical element) should be drawn on a rectangular grid for a specified square pattern type and subtype. `names_square` lists the currently supported square `types` (excluding those in `names_weave`).

Usage

```
pattern_square(type = "diagonal", subtype = NULL, nrow = 5L, ncol = 5L)

names_square
```

Arguments

`type`	Either "diagonal" (default), "diagonal_skew", "horizontal", "vertical", or any `type` in `names_weave`. See Details.
`subtype`	See Details. For "diagonal", "diagonal_skew", "horizontal", or "vertical" an integer of the desired number of colors (or other graphical elements).
`nrow`	Number of rows (height).
`ncol`	Number of columns (width).

Format

An object of class `character` of length 6.

Details

"horizontal", "vertical" "horizontal" and "vertical" simply cycle through the colors either horizontally or vertically. Use `subtype` to indicate the (integer) number of colors (or other graphical elements). "horizontal" will produce horizontal stripes of color whereas "vertical" will produce vertical stripes.

"diagonal", "diagonal_skew" "diagonal" and "diagonal_skew" simply cycle through the colors both horizontally and vertically. Use `subtype` to indicate the (integer) number of colors (or other graphical elements). If two colors are requested this provides the standard two-color checkerboard pattern. If there are more than three colors than "diagonal" will have colored diagonals going from top left to bottom right while "diagonal_skew" will have them going form bottom left to top right.

"square" "square" attempts a uniform coloring using "square_tiling" before falling falling back on "diagonal". If `subtype` is `1L`, `2L`, `3L`, or `4L` uses "square_tiling" else uses "diagonal".

"square_tiling" "square_tiling" supports uniform coloring for (non-staggered) square tilings. Use `subtype` to either indicate the (integer) number of colors or a string with four integers such as `"1231"` (will fill in a 2x2 matrix by row which will then be tiled). Supports up to a max of four colors.

any pattern from `names_weave` We simply convert the logical matrix returned by `pattern_weave()` into an integer matrix by having any `TRUE` set to `1L` and `FALSE` set to `2L`. Hence the various weave patterns only support (up to) two-color patterns. See `pattern_weave()` for more details about supported `type` and `subtype`.

Value

A matrix of integer values indicating where the each color (or other graphical element) should be drawn on a rectangular grid. Indices `[1,1]` of the matrix corresponds to the bottom-left of the grid while indices `[1,ncol]` corresponds to the bottom-right of the grid. This matrix has a "pattern_square" subclass which supports a special `print()` method.

See Also

`grid.pattern_regular_polygon()` for drawing to a graphics device polygons in multiple color/size/shape patterns. `pattern_weave()` for more information on "weave" patterns.

Examples

```
# supported square names
print(names_square)

# (main) diagonal has colors going from top left to bottom right
diagonal <- pattern_square("diagonal", 4L, nrow = 7L, ncol = 9L)
print(diagonal)

# skew diagonal has colors going from bottom left to top right
skew <- pattern_square("diagonal_skew", 4L, nrow = 7L, ncol = 9L)
print(skew)

horizontal <- pattern_square("horizontal", 4L, nrow = 8L, ncol = 8L)
print(horizontal)

vertical <- pattern_square("vertical", 4L, nrow = 8L, ncol = 8L)
print(vertical)

# uniform coloring using 4 colors
color4 <- pattern_square("square_tiling", 4L, nrow = 7L, ncol = 9L)
print(color4)

# uniform coloring using 3 colors
color3 <- pattern_square("square_tiling", 3L, nrow = 7L, ncol = 9L)
print(color3)

# also supports the various 'weave' patterns
zigzag <- pattern_square("twill_zigzag", nrow = 15L, ncol = 9L)
print(zigzag)
```

`pattern_weave` *Weave pattern matrix*

Description

`pattern_weave()` returns a logical matrix indicating where the warp lines should be "up" for a specified weave pattern type and subtype. `names_weave` is a character vector listing supported weave pattern types.

Usage

```
pattern_weave(type = "plain", subtype = NULL, nrow = 5L, ncol = 5L)

names_weave
```

Arguments

`type`	Type of weave. See Details.
`subtype`	Subtype of weave. See Details.
`nrow`	Number of rows (length of warp).
`ncol`	Number of columns (length of weft).

Format

An object of class `character` of length 10.

Details

Here is a list of the various weave `types` supported:

basket A simple criss-cross pattern using two threads at a time. Same as the "matt_irregular" weave but with a default `subtype` of `2L`.

matt A simple criss-cross pattern using 3 (or more) threads at a time. Same as the "matt_irregular" weave but with a default `subtype` of `3L`.

matt_irregular A generalization of the "plain" weave. A character `subtype` `"U/D(L+R)"` is a standard matt weave specification: `U` indicates number warp up, `D` indicates number warp down, `L` indicates number of warp up in repeat, and `R` indicates number of warp down in repeat. An integer `subtype` `N` will be interpreted as a `"N/N(N+N)"` irregular matt weave. A character `subtype` `"U/D"` will be interpreted as a `"U/D(U+D)"` irregular matt weave. Has a default `subtype` of `"3/2(4+2)"`.

plain A simple criss-cross pattern. Same as the "matt_irregular" weave but with a default `subtype` of `1L`.

rib_warp A plain weave variation that emphasizes vertical lines. An integer `subtype` `N` will be interpreted as a "matt_irregular" `"N/N(1+1)"` weave. A character `subtype` `"U/D"` will be interpreted as a "matt_irregular" `"U/D(1+1)"` weave. Default `subtype` of `2L`.

satin A "regular" satin weave is a special type of the elongated twill weave with a move number carefully chosen so no twill line is distinguishable. Same as the "twill_elongated" weave but with a default `subtype` of `5L`.

twill A simple diagonal pattern. Same as the "twill_elongated" weave but with a default `subtype` of `"2/1"`.

twill_elongated A generalization of the "twill" weave. A character `subtype` `"U/D(M)"` is a standard twill weave specification: `U` indicates number warp up, `D` indicates number warp down, and `M` indicates the "move" number. A character `subtype` `"U/D"` will be interpreted as a `"U/D(1)"` elongated twill weave. An integer `subtype` `N` will provide a `"{N-1}/1(1)"` elongated twill weave if `N` is less than 5, 6, or greater than 14 otherwise it will provide a `"{N-1}/1(M)"` weave where `M` is the largest possible regular "satin" move number. Default `subtype` of `"4/3(2)"`.

twill_herringbone Adds a (vertical) "herringbone" effect to the specified "twill_elongated" weave. Default `subtype` of `"4/3(2)"`.

twill_zigzag Adds a (vertical) "zig-zag" effect to the specified "twill_elongated" weave. Default `subtype` of `"4/3(2)"`.

For both "matt" and "twill" weaves the `U/D` part of the subtype can be further extended to `U1/D1*U2/D2`, `U1/D1*U2/D2*U3/D3`, etc. For the "matt" weave the "(L+R)" part of the subtype can be further extended to `(L1+R1+L2+R2)`, `(L1+R1+L2+R2+L3+R3)`, etc.

Value

A matrix of logical values indicating where the "warp" is "up" (if `TRUE`) or "down" (if `FALSE`). Indices `[1,1]` of the matrix corresponds to the bottom-left of the weave while indices `[1,ncol]` corresponds to the bottom-right of the weave. This matrix has a "pattern_weave" subclass which supports a special `print()` method.

See Also

`grid.pattern_weave()` for drawing weaves onto a graphics device. See https://textilestudycenter.com/derivatives-of-plain-weave/ for further information on the "matt" family of weaves, https://textilelearner.net/twill-weave-features-classification-derivatives-and-uses/ for further information on the "twill" family of weaves, and https://texwiz101.blogspot.com/2012/03/features-and-classification-of-satin.html for further information on "satin" weaves.

Examples

```
# supported weave names
print(names_weave)

plain <- pattern_weave("plain", nrow = 7, ncol = 9)
print(plain)

matt_irregular <- pattern_weave("matt_irregular", nrow = 9, ncol = 11)
print(matt_irregular)

satin <- pattern_weave("satin", nrow = 9, ncol = 11)
print(satin)
```

```
twill <- pattern_weave("twill", nrow = 9, ncol = 11)
print(twill)

twill_zigzag <- pattern_weave("twill_zigzag", nrow = 18, ncol = 11)
print(twill_zigzag)
```

`star_scale` *Compute regular star polygon scale or angles*

Description

`star_scale()` computes star `scale` value given an internal or external angle. `star_angle()` computes star angle (internal or external) given a `scale` value.

Usage

```
star_scale(n_vertices, angle, external = FALSE)

star_angle(n_vertices, scale, external = FALSE)
```

Arguments

`n_vertices`	Number of exterior vertices.
`angle`	Angle in degrees.
`external`	If `TRUE` angle should be considered an external angle.
`scale`	Scale from 0 to 1.

Details

`grid.pattern_regular_polygon()` parameterizes regular star polygons with the number of its external vertices and a `scale` that equals the fraction of the radius of the circle that circumscribes the interior vertices divided by the radius of the circle that circumscribes the exterior vertices. These helper functions help convert between that parameterization and either the internal or external angle of the regular star polygon.

Value

`star_scale()` returns a numeric value between 0 and 1 intended for use as the `scale` argument in `grid.pattern_regular_polygon()`. `star_angle()` returns a numeric value between 0 and 360 (degrees).

Examples

```
# |8/3| star has internal angle 45 degrees and external angle 90 degrees
scale <- star_scale(8, 45)
scale2 <- star_scale(8, 90, external = TRUE)
all.equal(scale, scale2)
star_angle(8, scale)
star_angle(8, scale, external = TRUE)

if (require("grid")) {
  grid.pattern_regular_polygon(shape = "star8", scale = scale, angle = 0,
                               spacing = 0.2, density = 0.8)
}
```

Index