Package: ggincerta (via r-universe)

October 29, 2026

Title Extend 'ggplot2' with Layers and Scales for Spatial Uncertainty Visualization

Version 0.2.2

Description Provide specialized 'ggplot2' layers and scales for spatial uncertainty visualization, including bivariate choropleth maps, pixel maps, glyph maps, and dual maps.

Imports cli, colorspace, dplyr, grDevices, grid, gtable, purrr, RColorBrewer, rlang, scales, sf, withr

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Suggests ggChernoff, testthat (>= 3.0.0), vdiffr

Config/testthat/edition 3

Depends ggplot2 (>= 3.5.0), R (>= 4.1.0)

LazyData true

URL https://github.com/maggiexma/ggincerta

BugReports https://github.com/maggiexma/ggincerta/issues

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://maggiexma.r-universe.dev

Date/Publication 2026-10-29 03:55:44 UTC

RemoteUrl https://github.com/maggiexma/ggincerta

RemoteRef HEAD

RemoteSha dd9f64b6dca1f936faee324e1288377be222ef2b

Contents

bivar_fade_palette . . . 2
bivar_palette . . . 3
duo . . . 4
geom_sf_dualmap . . . 4
geom_sf_glyph . . . 7
GuideBivariate . . . 9
GuideGlyph . . . 10
GuideVSUP . . . 11
manual_bivariate_scale . . . 12
nc . . . 15
nc_sim . . . 15
scale_fill_pixel . . . 16
ScaleBivariate . . . 17
ScaleVSUP . . . 21
StatPixel . . . 23
vsup_palette . . . 25
vsup_quantize . . . 26

Index **28**

`bivar_fade_palette` *Colour fading palette*

Description

One or more supplied colours construct a value colour scale, then varies perceptual properties such as lightness, saturation, or transparency along the uncertainty dimension.

Usage

```
bivar_fade_palette(
  colours,
  n_breaks,
  fade = c("lighten", "alpha", "desaturate"),
  alpha_range = c(1, 0.3),
  max_light = 0.7,
  max_desat = 0.9,
  space = "Lab"
)
```

Arguments

`colours`	A character vector of colours used as key points in interpolation.

n_breaks	An integer or a length-two vector specifying the desired number of bins for each variable. The default is 4 for both variables. When nice.breaks = TRUE, this value is treated as a suggestion and the actual number of bins may differ. When nice.breaks = FALSE, exactly n_breaks equal-width bins are generated. Unequal desired numbers of bins are supported.
fade	A character string specifying the fading method: "lighten", "desaturate", or "alpha".
alpha_range	A numeric vector of length two specifying the range of transparency values used when fade = "alpha".
max_light	A numeric value specifying the maximum amount of lightening applied across uncertainty levels.
max_desat	A numeric value specifying the maximum amount of desaturation applied across uncertainty levels.
space	A character string specifying the colour space used for interpolation.

---

bivar_palette *Colour blending palette*

Description

This palette function constructs two colour ramps from white to the supplied endpoint colours, then blends them by additive averaging in grDevices::rgb colour space. The resulting palette contains one colour for each combination of the two binned variables.

Usage

```
bivar_palette(
  colours = NULL,
  n_breaks = c(4, 4),
  flip = c("none", "vertical", "horizontal", "both")
)
```

Arguments

colours	A character vector of two colours used as the endpoints of the two colour ramps.
n_breaks	An integer or a length-two vector specifying the desired number of bins for each variable. The default is 4 for both variables. When nice.breaks = TRUE, this value is treated as a suggestion and the actual number of bins may differ. When nice.breaks = FALSE, exactly n_breaks equal-width bins are generated. Unequal desired numbers of bins are supported.
flip	A character string specifying how to flip the palette: "none" (the default), "vertical", "horizontal", or "both".

duo *Format input and assign the "map" class*

Description

duo() and duo_pixel() create paired mapping objects that combine two variables, record their names, and assign the bivariate/pixel class as an attribute for use in aesthetic mappings.

Usage

```
duo(v1, v2)

duo_pixel(estimate, error)
```

Arguments

v1, v2	Input variables for duo().
estimate, error	Input variables for duo_pixel() representing the point estimate and its uncertainty.

Value

A list-like object containing pairs of values from the two variables, with attributes storing the variable names and the class.

Examples

```
value <- nc$value
sd <- nc$sd
res <- duo(value, sd)
res_pixel <- duo_pixel(value, sd)
class(res); class(res_pixel)
attr(res, "vars"); attr(res_pixel, "vars")
```

geom_sf_dualmap *Dual map*

Description

geom_sf_dualmap() generates a map where each region is represented by two visual components: the surrounding area is mapped using fill, and the central glyph is mapped using colour.

Usage

```
geom_sf_dualmap(
  mapping = NULL,
  data = NULL,
  ...,
  shape = "circle",
  max_angle = NULL,
  size = 1,
  point_fun = sf::st_point_on_surface,
  border_colour = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  angle_guide = TRUE,
  angle_name = waiver(),
  angle_order = 2,
  fill_scale = NULL
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example

of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`shape`	Glyph shape. One of `"circle"` (the default), `"square"`, `"triangle"`, `"hex"`, `"drop"`, or `"chernoff"`.
`max_angle`	Maximum value of the `angle` aesthetic used for rescaling glyph rotation.
`size`	A positive numeric scaling factor controlling glyph size.
`point_fun`	Function used to calculate the representative point for each region. The default is usually `sf::st_point_on_surface()`.
`border_colour`	Colour used for glyph borders.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. You can also set this to one of "polygon", "line", and "point" to override the default legend.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.
`angle_guide`	Logical indicating whether to display a guide for the `angle` aesthetic.
`angle_name`	Title used for the angle guide.
`angle_order`	Order of the angle guide relative to other guides.
`fill_scale`	Optional fill scale for the surrounding area. If `NULL`, a default grey gradient is added when the `fill` mapping is not created by `duo()`.

Details

This allows two variables, such as value and uncertainty, to be shown within the same geographic region using separate scales.

Value

A list of layers generated by `geom_sf()` and `geom_sf_glyph()`.

Examples

```
# Dual map with separate fill and colour mappings
ggplot(nc) +
  geom_sf_dualmap(aes(fill = sd, colour = value))
```

```
# Dual map using a bivariate colour scale for the central glyph
ggplot(nc) +
  geom_sf_dualmap(aes(fill = duo(value, sd), colour = sd_log2))
```

`geom_sf_glyph` *Glyph map*

Description

`geom_sf_glyph()` generates a glyph map sf layer. A glyph map is a centroid-based map, where each region is represented by a chosen glyph.

Usage

```
geom_sf_glyph(
  mapping = NULL,
  data = NULL,
  ...,
  shape = "circle",
  max_angle = NULL,
  size = 1,
  point_fun = sf::st_point_on_surface,
  border_colour = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  angle_guide = TRUE,
  angle_name = waiver(),
  angle_order = 99
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`shape` Glyph shape. One of `"circle"` (the default), `"square"`, `"triangle"`, `"hex"`, `"drop"`, or `"chernoff"`.

`max_angle` Maximum value of the `angle` aesthetic used for rescaling glyph rotation.

`size` A positive numeric scaling factor controlling glyph size.

`point_fun` Function used to calculate the representative point for each region. The default is usually `sf::st_point_on_surface()`.

`border_colour` Colour used for glyph borders.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes.

You can also set this to one of "polygon", "line", and "point" to override the default legend.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`angle_guide` Logical indicating whether to display a guide for the `angle` aesthetic.

`angle_name` Title used for the angle guide.

`angle_order` Order of the angle guide relative to other guides.

Details

Regular shape glyphs can be treated as ordinary point-like symbols and used together with `bivariate_scale()`. Drop-shaped glyphs use rotation angle to represent uncertainty. Chernoff glyphs are adapted from the `ggChernoff` package and work with the `smile` aesthetic.

Value

A list of ggplot2 layer objects.

Examples

```
# Regular glyph map
ggplot(nc) +
  geom_sf_glyph(aes(colour = value), shape = "hex")

# Rotated drop glyph map
ggplot(nc) +
  geom_sf_glyph(
    aes(colour = value, angle = sd),
    shape = "drop"
  )

# Chernoff face glyph map
if (requireNamespace("ggChernoff", quietly = TRUE)) {
  ggplot(nc) +
    geom_sf_glyph(
      aes(colour = value, smile = sd),
      shape = "chernoff"
    )
}
```

GuideBivariate *Bivariate colour guide*

Description

The bivariate colour guide is displayed as a grid, where each cell represents the mapping between a colour and the corresponding bin intervals of the two variables along the guide axes.

Usage

```
GuideBivariate

guide_bivariate(
  theme = NULL,
  title = NULL,
  order = 0,
  position = NULL,
```

```
  rotated = FALSE,
  angle = 45
)
```

Arguments

theme	A theme object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide partially overrides, and is combined with, the plot's theme. Arguments that apply to a single legend are respected, most of which have the legend-prefix. Arguments that apply to combined legends (the legend box) are ignored, including legend.position, legend.justification.*, legend.location and legend.box.*.
title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
order	positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
position	A character string indicating where the legend should be placed relative to the plot panels. One of "top", "right", "bottom", "left", or "inside".
rotated	Logical indicating whether the guide should be displayed in a rotated orientation.
angle	A numeric value specifying the rotation angle of the guide.

GuideGlyph *Drop glyph guide*

Description

Guide for drop glyph consists of two axes and a sequence of example drop grobs illustrating the mapping between uncertainty and glyph rotation angles.

Usage

```
GuideGlyph

guide_glyph(
  theme = NULL,
  title = waiver(),
  order = 99,
  position = NULL,
  direction = "vertical"
)
```

Arguments

theme	A theme object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide partially overrides, and is combined with, the plot's theme. Arguments that apply to a single legend are respected, most of which have the legend-prefix. Arguments that apply to combined legends (the legend box) are ignored, including legend.position, legend.justification.*, legend.location and legend.box.*.
title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
order	positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
position	A character string indicating where the legend should be placed relative to the plot panels. One of "top", "right", "bottom", "left", or "inside".
direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical".

Details

Rotation angles range from π to $-\pi$, corresponding to the minimum and maximum uncertainty values respectively.

GuideVSUP *VSUP guide*

Description

This guide is displayed as a fan-shaped legend. Uncertainty levels are arranged along the radial axis, while value groups are arranged along the arc axis. As uncertainty increases towards the centre of the guide, the number of value groups progressively decreases until only a single group remains.

Usage

```
GuideVSUP

guide_vsup(theme = NULL, title = waiver(), order = 0, position = NULL)
```

Arguments

theme	A theme object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide partially overrides, and is combined with, the plot's theme. Arguments that apply to a single legend are respected, most of which have the legend-prefix. Arguments that apply to combined legends (the legend box) are ignored, including legend.position, legend.justification.*, legend.location and legend.box.*.

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
order	positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
position	A character string indicating where the legend should be placed relative to the plot panels. One of "top", "right", "bottom", "left", or "inside".

---

manual_bivariate_scale

Create your own bivariate colour scale

Description

These scales allow users to provide all colours used for the combinations of two variables in a bivariate colour scale.

Usage

```
manual_bivariate_scale(
  aesthetics,
  ...,
  values,
  name = waiver(),
  breaks = list(waiver(), waiver()),
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  na.translate = TRUE,
  drop = FALSE,
  guide = guide_bivariate(),
  n_breaks = c(4, 4),
  quantile = FALSE,
  var1_name = NULL,
  var2_name = NULL,
  super = ScaleBivariate
)

scale_fill_bivariate_manual(
  ...,
  values,
  name = waiver(),
  var1_name = NULL,
```

```
  var2_name = NULL,
  n_breaks = c(4, 4),
  quantile = FALSE,
  breaks = list(waiver(), waiver()),
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "fill",
  guide = guide_bivariate()
)

scale_colour_bivariate_manual(
  ...,
  values,
  name = waiver(),
  var1_name = NULL,
  var2_name = NULL,
  n_breaks = c(4, 4),
  quantile = FALSE,
  breaks = list(waiver(), waiver()),
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "colour",
  guide = guide_bivariate()
)

scale_color_bivariate_manual(
  ...,
  values,
  name = waiver(),
  var1_name = NULL,
  var2_name = NULL,
  n_breaks = c(4, 4),
  quantile = FALSE,
  breaks = list(waiver(), waiver()),
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "colour",
  guide = guide_bivariate()
)
```

Arguments

aesthetics	The names of the aesthetics that this scale works with.
...	Other arguments passed to ggplot2::discrete_scale().
values	A character vector of colours. The length should be at least the product of n_breaks, giving one colour for each combination of the two binned variables.
name	The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
breaks	A list of two numeric vectors specifying bin boundaries for the two variables. If waiver(), breaks are computed automatically according to n_breaks, nice.breaks, and quantile.
labels	A list of two character vectors or labelling functions used to label the bin boundaries for the two variables. If waiver(), default labels are generated from the breaks.
limits	A list of two vectors specifying the limits for the two variables. For continuous variables, each element specifies the numeric range to include. For discrete variables, it specifies the levels to include.
transform	A list of one or two transformations applied to continuous variables before binning. Automatic pretty, equal-width, and quantile breaks are computed on the transformed scale. Each element can be a transformation name or a transformer object accepted by scales::as.transform().
na.value	If na.translate = TRUE, what aesthetic value should the missing values be displayed as? Does not apply to position scales where NA is always placed at the far right.
na.translate	Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.
drop	Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE includes the levels in the factor. Please note that to display every level in a legend, the layer should use show.legend = TRUE.
guide	A function used to create a guide or its name. See guides() for more information.
n_breaks	An integer or a length-two vector specifying the desired number of bins for each variable. The default is 4 for both variables. When nice.breaks = TRUE, this value is treated as a suggestion and the actual number of bins may differ. When nice.breaks = FALSE, exactly n_breaks equal-width bins are generated. Unequal desired numbers of bins are supported.
quantile	A logical value or length-two logical vector indicating whether the variables should be divided using quantile-based bins. If quantile breaks are not unique, equal-width bins are used instead.
var1_name, var2_name	Optional names for v1 and v2. Used as axis titles in the legend. If NULL, the default, the names are taken from the mapping.
super	The super class to use for the constructed scale

Examples

```
ggplot(nc, aes(fill = duo(value, sd))) +
  geom_sf() +
  scale_fill_bivariate_manual(
    values = c(
      "#F7F4F9", "#D4B9DA", "#C994C7", "#980043",
      "#E0ECF4", "#BFD3E6", "#9EBCDA", "#8856A7",
      "#D0D1E6", "#A6BDDB", "#74A9CF", "#2B8CBE",
      "#B8E186", "#7FBC41", "#4D9221", "#276419"
    ),
    n_breaks = 4
  )
```

---

nc *North Carolina SIDS data*

Description

The dataset nc is derived from the North Carolina shapefile (nc.shp) included in the **sf** package. Two random variables, value and sd, have been added for demonstration purposes.

Further details about the original data can be found in the spdep package vignette.

Usage

```
nc
```

Format

A sf object.

Examples

```
head(nc)

plot(sf::st_geometry(nc))
```

---

nc_sim *Simulated North Carolina spatial patterns*

Description

A simulated areal dataset based on the North Carolina county boundaries. It contains three spatial patterns, linear trend, hotspot, and CAR smooth field, each generated at medium and strong signal strengths.

Usage

```
nc_sim
```

Format

An sf object with 600 rows. It contains the original variables from nc, together with:

pattern Spatial pattern type.

strength Signal strength.

noise_level Noise level used in the simulation.

value_sim Simulated value.

Examples

```
head(nc_sim)

plot(sf::st_geometry(nc_sim))
```

---

scale_fill_pixel *Pixel fill scale*

Description

Pixel fill scale

Usage

```
scale_fill_pixel(
  type = "seq",
  palette = "Oranges",
  direction = 1,
  name = waiver(),
  ...
)
```

Arguments

type	One of "seq" (sequential), "div" (diverging) or "qual" (qualitative)
palette	A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding output values (e.g., scales::pal_area()).
direction	Sets the order of colours in the scale. If 1, the default, colours are as output by RColorBrewer::brewer.pal(). If -1, the order of colours is reversed.
name	The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
...	Additional arguments passed to ggplot2::continuous_scale().

`ScaleBivariate` *Bivariate colour scale constructor*

Description

`bivariate_scale()` maps binned combinations of two variables to colour dimensions and their combinations in perceptual colour space, supporting the construction of bivariate choropleth maps.

Usage

```
ScaleBivariate

bivariate_scale(
  aesthetics,
  ...,
  name = waiver(),
  breaks = list(waiver(), waiver()),
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  na.translate = TRUE,
  drop = FALSE,
  guide = waiver(),
  colours = c("#EDF8B1", "#2C7FB8"),
  palette_fun = NULL,
  palette_params = list(),
  n_breaks = c(4, 4),
  nice.breaks = TRUE,
  quantile = FALSE,
  var1_name = NULL,
  var2_name = NULL,
  super = ScaleBivariate
)

scale_fill_bivariate(
  ...,
  name = waiver(),
  var1_name = NULL,
  var2_name = NULL,
  colours = c("#EDF8B1", "#2C7FB8"),
  palette_fun = NULL,
  palette_params = list(),
  n_breaks = c(4, 4),
  nice.breaks = TRUE,
  quantile = FALSE,
  breaks = list(waiver(), waiver()),
```

```
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  aesthetics = "fill",
  guide = guide_bivariate()
)

scale_color_bivariate(
  ...,
  name = waiver(),
  var1_name = NULL,
  var2_name = NULL,
  colours = c("#EDF8B1", "#2C7FB8"),
  palette_fun = NULL,
  palette_params = list(),
  n_breaks = c(4, 4),
  nice.breaks = TRUE,
  quantile = FALSE,
  breaks = list(waiver(), waiver()),
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  aesthetics = "colour",
  guide = guide_bivariate()
)

scale_colour_bivariate(
  ...,
  name = waiver(),
  var1_name = NULL,
  var2_name = NULL,
  colours = c("#EDF8B1", "#2C7FB8"),
  palette_fun = NULL,
  palette_params = list(),
  n_breaks = c(4, 4),
  nice.breaks = TRUE,
  quantile = FALSE,
  breaks = list(waiver(), waiver()),
  labels = list(waiver(), waiver()),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  na.value = NA,
  aesthetics = "colour",
  guide = guide_bivariate()
)
```

Arguments

aesthetics	The names of the aesthetics that this scale works with.
...	Other arguments passed to ggplot2::discrete_scale().
name	The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
breaks	A list of two numeric vectors specifying bin boundaries for the two variables. If waiver(), breaks are computed automatically according to n_breaks, nice.breaks, and quantile.
labels	A list of two character vectors or labelling functions used to label the bin boundaries for the two variables. If waiver(), default labels are generated from the breaks.
limits	A list of two vectors specifying the limits for the two variables. For continuous variables, each element specifies the numeric range to include. For discrete variables, it specifies the levels to include.
transform	A list of one or two transformations applied to continuous variables before binning. Automatic pretty, equal-width, and quantile breaks are computed on the transformed scale. Each element can be a transformation name or a transformer object accepted by scales::as.transform().
na.value	If na.translate = TRUE, what aesthetic value should the missing values be displayed as? Does not apply to position scales where NA is always placed at the far right.
na.translate	Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.
drop	Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE includes the levels in the factor. Please note that to display every level in a legend, the layer should use show.legend = TRUE.
guide	A function used to create a guide or its name. See guides() for more information.
colours	A character vector of colours used as key points in the colour ramp that variables are mapped to. For details on how supplied colours are used to construct the resulting palette, see bivar_palette() and bivar_fade_palette().
palette_fun	A palette function that, when called with colours and n_breaks, returns a character vector of colours for all binned combinations. If NULL, the default, bivar_palette() is used.
palette_params	A list of additional arguments passed to palette_fun. See bivar_palette() and bivar_fade_palette() for available arguments.
n_breaks	An integer or a length-two vector specifying the desired number of bins for each variable. The default is 4 for both variables. When nice.breaks = TRUE, this value is treated as a suggestion and the actual number of bins may differ. When nice.breaks = FALSE, exactly n_breaks equal-width bins are generated. Unequal desired numbers of bins are supported.

`nice.breaks` A logical value or length-two logical vector indicating whether automatically generated breaks should use pretty, human-readable values. When `TRUE`, breaks are generated using `pretty()` on the transformed scale, with `n_breaks` treated as a suggestion. The resulting breaks may extend beyond the data range and the actual number of bins may differ from `n_breaks`. When `FALSE`, exactly `n_breaks` equal-width bins are generated.

`quantile` A logical value or length-two logical vector indicating whether the variables should be divided using quantile-based bins. If quantile breaks are not unique, equal-width bins are used instead.

`var1_name`, `var2_name` Optional names for `v1` and `v2`. Used as axis titles in the legend. If `NULL`, the default, the names are taken from the mapping.

`super` The super class to use for the constructed scale

Details

It can be automatically dispatched in `aes()` using `duo()` and works with any ggplot2 geom.

Value

A `ScaleBivariate` ggproto object.

See Also

ggplot2::Scale for the base ggproto class that all scale objects inherit from.

Examples

```
# Basic bivariate map
ggplot(nc) +
  geom_sf(aes(fill = duo(value, sd)))

# Use an alternative bivariate palette
ggplot(nc) +
  geom_sf(aes(fill = duo(value, sd))) +
  scale_fill_bivariate(
    palette_fun = bivar_fade_palette,
    colours = c("#F6E8C3", "orange", "red")
  )

# Customize the desired number of bins
ggplot(nc) +
  geom_sf(aes(fill = duo(value, sd))) +
  scale_fill_bivariate(n_breaks = c(3, 4))

# Use exactly four equal-width bins for each variable
ggplot(nc) +
  geom_sf(aes(fill = duo(value, sd))) +
  scale_fill_bivariate(
    n_breaks = 4,
```

```
  nice.breaks = FALSE
)
```

ScaleVSUP *Value-Suppressing Uncertainty Palette (VSUP) scale*

Description

This scale implements Value-Suppressing Uncertainty Palettes (VSUPs), proposed by Correll et al. (2018). VSUPs reduce colour variation as uncertainty increases, so that value differences are progressively suppressed in less reliable regions.

Usage

```
ScaleVSUP

scale_fill_vsup(
  name = waiver(),
  colours = RColorBrewer::brewer.pal(3, "YlGnBu"),
  layers = 4,
  branch = 2L,
  breaks = list(NULL, NULL),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  title_value = "Value",
  title_uncertainty = "Uncertainty",
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "fill",
  max_light = 0.7,
  max_desat = 0.9,
  pow_light = 1,
  pow_desat = 1,
  space = "Lab",
  guide = guide_vsup(),
  ...
)

scale_colour_vsup(
  name = waiver(),
  colours = RColorBrewer::brewer.pal(3, "YlGnBu"),
  layers = 4,
  branch = 2L,
  breaks = list(NULL, NULL),
  limits = list(NULL, NULL),
  transform = list("identity", "identity"),
  title_value = "Value",
```

```
  title_uncertainty = "Uncertainty",
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "colour",
  max_light = 0.7,
  max_desat = 0.9,
  pow_light = 1,
  pow_desat = 1,
  space = "Lab",
  guide = guide_vsup(),
  ...
)
```

Arguments

name	The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.
colours	A character vector of colours used as key points in the value colour scale. See `vsup_palette()` for details.
layers	An integer specifying the number of uncertainty levels in the VSUP hierarchy.
branch	An integer specifying the branching factor used to determine the number of value bins across uncertainty levels. The maximum number of value bins is `branch^(layers - 1)`, with fewer value bins used at higher uncertainty levels.
breaks	A list of two numeric vectors specifying break points for the value and uncertainty variables, respectively. If an element is `NULL`, breaks are generated automatically as equal-width intervals on the transformed scale. The value dimension uses `branch^(layers - 1) + 1` break points, while the uncertainty dimension uses `layers + 1` break points.
limits	A list of two numeric vectors specifying the ranges used for the value and uncertainty variables. If an element is `NULL`, the finite range of the corresponding variable is used. Values outside the specified limits are mapped to `na.value`.
transform	A list of one or two transformations applied to the value and uncertainty variables before quantization. Automatic breaks are generated as equal-width intervals on the transformed scale. Each element can be a transformation name or a transformer object accepted by `scales::as.transform()`.
title_value, title_uncertainty	Optional titles for the value and uncertainty dimensions in the guide.
na.value	If `na.translate = TRUE`, what aesthetic value should the missing values be displayed as? Does not apply to position scales where `NA` is always placed at the far right.
na.translate	Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.
aesthetics	The names of the aesthetics that this scale works with.

`max_light` A numeric value specifying the maximum amount of lightening applied across uncertainty levels.

`max_desat` A numeric value specifying the maximum amount of desaturation applied across uncertainty levels.

`pow_light`, `pow_desat` Numeric values controlling the rate of lightening and desaturation across uncertainty levels.

`space` A character string specifying the colour space used for colour interpolation.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`...` Other arguments passed to `ggplot2::discrete_scale()`.

See Also

Correll et al. (2018) doi:10.1145/3173574.3174216 for technical details.

`StatPixel` *Pixel map*

Description

`geom_sf_pixel()` generates a pixel map layer on areal sf data. Each region is tessellated into small pixels, with pixel colours mapped from values sampled from a specified distribution.

Usage

```
StatPixel

geom_sf_pixel(
  mapping = NULL,
  data = NULL,
  n = 60,
  distribution = "uniform",
  seed = NULL,
  pixel_shape = "hex",
  flat_topped = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
n	integer of length 1 or 2, number of grid cells in x and y direction (columns, rows)
distribution	Distribution used to sample pixel values within each region. Currently supports "uniform" (the default) and "normal".
seed	Integer seed used for reproducible sampling.
pixel_shape	Shape of the generated pixels. One of "hex" (the default), "square", or "rect". "rect" is when dividing the x and y ranges into the same number of intervals, so cells may be rectangular.
flat_topped	logical; if TRUE generate flat topped hexagons, else generate pointy topped
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. You can also set this to one of "polygon", "line", and "point" to override the default legend.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

Mappings in `geom_sf_pixel()` is also supplied with `duo_pixel()` inside `aes()`, which automatically dispatches an scale.

Since `sf::st_intersection()` is used internally, operating directly on geographic (s2) sf objects can be slow, especially when a large number of pixels are generated. Projecting data to a planar coordinate system in advance is recommended.

Value

A list of ggplot2 layer objects.

Examples

```
# Transform sf data into a planar crs for faster geometric intersection
nc_flat <- sf::st_transform(nc, sf::st_crs(3857))

# Basic pixel map
ggplot(nc_flat, aes(fill = duo_pixel(value, sd))) +
  geom_sf_pixel(n = 40)

# Control pixel shape and resolution
ggplot(nc_flat, aes(fill = duo_pixel(value, sd))) +
  geom_sf_pixel(n = 30, pixel_shape = "square")
```

---

vsup_palette *Value-Suppressing Uncertainty Palettes*

Description

`vsup_palette()` handles colour generation for VSUP scales. It modifies the supplied base colours by progressively desaturating and lightening them as uncertainty increases.

Usage

```
vsup_palette(
  leaf_info,
  colours,
  layers = 4,
  branch = 2,
```

```
  max_light = 0.7,
  max_desat = 0.9,
  pow_light = 1,
  pow_desat = 1,
  space = "Lab"
)
```

Arguments

leaf_info	A data frame describing the VSUP bin structure, produced by vsup_quantize(). It stores the leaf identifiers, uncertainty layers, and value positions used to assign colours.
colours	A character vector of colours used as key points for interpolating the value colour scale at the lowest uncertainty level.
layers	An integer specifying the number of uncertainty levels in the VSUP hierarchy.
branch	An integer specifying the branching factor used to determine the number of value bins across uncertainty levels. The maximum number of value bins is branch^(layers - 1), with fewer value bins used at higher uncertainty levels.
max_light	A numeric value specifying the maximum amount of lightening applied across uncertainty levels.
max_desat	A numeric value specifying the maximum amount of desaturation applied across uncertainty levels.
pow_light, pow_desat	Numeric values controlling the rate of lightening and desaturation across uncertainty levels.
space	A character string specifying the colour space used for colour interpolation.

---

vsup_quantize *Tree quantization for Value-Suppressing Uncertainty Palettes*

Description

Quantizes value and uncertainty variables into the hierarchical tree structure used by Value-Suppressing Uncertainty Palettes (VSUPs).

Usage

```
vsup_quantize(
  v,
  u,
  layers = 4,
  branch = 2L,
  breaks = list(NULL, NULL),
  limits = list(NULL, NULL),
  transform = list("identity", "identity")
)
```

Arguments

v	A numeric vector containing the value variable.
u	A numeric vector containing the uncertainty variable.
layers	An integer specifying the number of uncertainty levels.
branch	An integer specifying the branching factor of the VSUP tree. The maximum number of value bins is branch^(layers - 1).
breaks	A list of two numeric vectors specifying break points for the value and uncertainty variables, respectively. If an element is NULL, breaks are generated automatically as equal-width intervals on the transformed scale. The value dimension requires branch^(layers - 1) + 1 break points and the uncertainty dimension requires layers + 1 break points.
limits	A list of two numeric vectors specifying the ranges used for the value and uncertainty variables. If an element is NULL, the finite data range is used. Values outside these limits are not assigned to a VSUP leaf.
transform	A list of one or two transformations applied to v and u before quantization. Each element can be a transformation name or a transformer object accepted by scales::as.transform().

Details

The uncertainty variable is divided into layers intervals. The number of value bins varies across uncertainty levels according to branch, with the maximum number of value bins equal to branch^(layers - 1). Higher uncertainty levels therefore use progressively fewer value bins.

When breaks are not supplied, both variables are divided using equal-width intervals on their transformed scales. User-supplied breaks are restricted to the corresponding limits and must provide the required number of break points after limits are applied.

Value

A list with the following components:

value A factor containing the VSUP leaf identifier assigned to each observation. Observations outside the limits or with missing values are returned as NA.

leaf_info A data frame describing each VSUP leaf, including its leaf identifier, uncertainty layer, normalized value position, and value midpoint.

value_breaks The value break points on the original data scale.

uncertainty_breaks The uncertainty break points on the original data scale.

Index

∗ **datasets**
    nc, 15
    nc_sim, 15

aes(), *5*, *7*, *24*
annotation_borders(), *6*, *8*, *24*

bivar_fade_palette, 2
bivar_fade_palette(), *19*
bivar_palette, 3
bivar_palette(), *19*
bivariate_scale *(*ScaleBivariate*)*, 17

duo, 4
duo(), *6*
duo_pixel *(*duo*)*, 4
duo_pixel(), *25*

fortify(), *5*, *7*, *24*

geom_sf(), *6*
geom_sf_dualmap, 4
geom_sf_glyph, 7
geom_sf_glyph(), *6*
geom_sf_pixel *(*StatPixel*)*, 23
ggplot(), *5*, *7*, *24*
ggplot2::continuous_scale(), *16*
ggplot2::discrete_scale(), *14*, *19*, *23*
ggplot2::Scale, *20*
guide_bivariate *(*GuideBivariate*)*, 9
guide_glyph *(*GuideGlyph*)*, 10
guide_vsup *(*GuideVSUP*)*, 11
GuideBivariate, 9
GuideGlyph, 10
guides(), *14*, *19*, *23*
GuideVSUP, 11

key glyphs, *6*, *8*, *25*

labs(), *10–12*
layer(), *5*, *6*, *8*, *24*, *25*

manual_bivariate_scale, 12

nc, 15
nc_sim, 15

pretty(), *20*

RColorBrewer::brewer.pal(), *16*

scale_color_bivariate *(*ScaleBivariate*)*, 17
scale_color_bivariate_manual *(*manual_bivariate_scale*)*, 12
scale_colour_bivariate *(*ScaleBivariate*)*, 17
scale_colour_bivariate_manual *(*manual_bivariate_scale*)*, 12
scale_colour_vsup *(*ScaleVSUP*)*, 21
scale_fill_bivariate *(*ScaleBivariate*)*, 17
scale_fill_bivariate_manual *(*manual_bivariate_scale*)*, 12
scale_fill_pixel, 16
scale_fill_vsup *(*ScaleVSUP*)*, 21
ScaleBivariate, 17
scales::as.transform(), *14*, *19*, *22*, *27*
scales::pal_area(), *16*
ScaleVSUP, 21
sf::st_intersection(), *25*
sf::st_point_on_surface(), *6*, *8*
StatPixel, 23

theme, *10*, *11*

vsup_palette, 25
vsup_palette(), *22*
vsup_quantize, 26
vsup_quantize(), *26*

waiver(), *10–12*