Package 'ggtaichi'

August 24, 2026

Type Package

Title Taichi-Diagram Visualization for Two Data Sources

Version 0.2.0

Description A data visualization design that compares two (usually on a par with each other) data sources on one grid of taichi (yin-yang) diagrams, where the two interlocking fish of every symbol are filled by the two sources, while inheriting 'ggplot2' features.

License GPL (>= 3)

Encoding UTF-8

Language en-US

LazyData true

URL https://pursuitofdatascience.github.io/ggtaichi/, https://github.com/PursuitOfDataScience/ggtaichi

BugReports https://github.com/PursuitOfDataScience/ggtaichi/issues

Depends R (>= 3.5.0)

Imports rlang, grid, grDevices, ggplot2 (>= 3.4.0), ggnewscale (>= 0.4.5)

RoxygenNote 7.3.2

Suggests rmarkdown, knitr, testthat (>= 3.0.0), vdiffr, gganimate

VignetteBuilder knitr

NeedsCompilation no

Author Youzhi Yu [aut, cre]

Maintainer Youzhi Yu <yuyouzhi666@icloud.com>

Repository CRAN

Date/Publication 2026-08-24 17:40:02 UTC

Contents

`cafes_tg` *Synthetic café orders: espresso vs. matcha*

Description

A small, deliberately *synthetic* two-source dataset for demos and vignettes: weekly orders (per 100 customers) of espresso and matcha drinks across eight fictional neighbourhoods over a 12-week season. It provides an evergreen alternative to the COVID-era `pitts_tg` / `states_tg` data, and because both columns share the same units it is the natural demo for `shared_limits` / `shared_legend` in `geom_taichi()`. The values are simulated with a fixed seed (espresso cools off over the season while matcha picks up, at neighbourhood-specific rates, plus noise); the generating script ships in `data-raw/cafes_tg.R` in the source repository.

Usage

```
cafes_tg
```

Format

A data frame with 96 rows and 4 columns:

week Week of the season, 1 to 12.

neighbourhood One of eight fictional neighbourhoods (factor).

espresso Weekly espresso orders per 100 customers.

matcha Weekly matcha orders per 100 customers.

Source

Simulated by the package author; see `data-raw/cafes_tg.R`.

Examples

```
library(ggplot2)
ggplot(cafes_tg, aes(x = week, y = neighbourhood)) +
  geom_taichi(yin = matcha, yang = espresso, shared_legend = TRUE) +
  theme_taichi()
```

`geom_taichi` *Taichi*

Description

The taichi geom turns each cell of a heatmap-like grid into a taichi (yin-yang) diagram. The two interlocking "fish" of the diagram use luminance to show the values from two data sources on the same plot, so four dimensions of data can be expressed at once: the `x` and `y` position of every taichi symbol plus the `yin` and `yang` values that fill its two halves. With the optional eyes enabled and mapped to data (see `eyes`, `yin_eye_size`, `yang_eye_size`), a single glyph can carry up to six dimensions.

Usage

```
geom_taichi(
  yin,
  yang,
  yin_name = NULL,
  yang_name = NULL,
  yin_colors = c("gray100", "gray85", "gray50", "gray35", "gray0"),
  yang_colors = c("#FED7D8", "#FE8C91", "#F5636B", "#E72D3F", "#C20824"),
  yin_scale = NULL,
  yang_scale = NULL,
  angle = NULL,
  eyes = FALSE,
  yin_eye_size = 0.15,
  yang_eye_size = 0.15,
  yin_eye_colour = "white",
  yang_eye_colour = "black",
  shared_limits = FALSE,
  shared_legend = FALSE,
  width = NULL,
  height = NULL,
  alpha = NA,
  na.rm = FALSE,
  colour = NA,
  linewidth = 0.1,
  linetype = 1,
  show.legend = NA,
  ...
)
```

Arguments

`yin`	The unquoted column name (or a literal string naming a column) for the yin (dark) fish of the taichi symbol. To pass a name held in a variable, use `.data[[nm]]`

or `!!rlang::sym(nm)` — a bare variable would be mapped as a constant fill, exactly as it would be inside `aes()`.

`yang` The unquoted column name (or a literal string naming a column) for the yang (light) fish of the taichi symbol, as `yin`.

`yin_name` The label name (in quotes) for the legend of the yin rendering. Default is `NULL` (uses the column name).

`yang_name` The label name (in quotes) for the legend of the yang rendering. Default is `NULL` (uses the column name).

`yin_colors` A color vector, usually as hex codes, for the yin fish fill. Used as a gradient for continuous data and as a discrete palette for factor/character data. Ignored if `yin_scale` is provided.

`yang_colors` A color vector, usually as hex codes, for the yang fish fill. Used as a gradient for continuous data and as a discrete palette for factor/character data. Ignored if `yang_scale` is provided.

`yin_scale` An optional fill scale for the yin fish: either a ready scale object or a scale constructor function (e.g. `ggplot2::scale_fill_viridis_d`). Overrides auto-detection. It must govern a *fill* aesthetic; a scale for another aesthetic (say `scale_colour_viridis_c`) is rejected with an error rather than quietly leaving the fish on the default gradient.

`yang_scale` An optional fill scale for the yang fish, as `yin_scale`.

`angle` Rotation of each glyph in degrees, counter-clockwise: either a single number or an unquoted column name (one angle per cell). A mapped column must be numeric.

`eyes` Logical. If `TRUE`, draws the classic taichi eyes (dots), each centred in its fish's head. Default `FALSE`, preserving the plain v0.1.0 look.

`yin_eye_size, yang_eye_size`
Size of each eye as a proportion of the glyph radius: a constant (default 0.15) or an unquoted data column to encode a variable (see the Eyes section for the rescaling rule).

`yin_eye_colour, yang_eye_colour`
Colour of each eye dot: a constant (defaults "white" and "black") or an unquoted data column containing colour strings.

`shared_limits` If `TRUE` and both sources are of the same type (both continuous, or both discrete), the two auto-built fill scales share common limits — the union range (or union of levels) of `yin` and `yang` — so equal values read as equal ink. Explicit `limits` passed through `...` take precedence. Default `FALSE`.

`shared_legend` If `TRUE`, treats the two sources as directly comparable: implies `shared_limits = TRUE`, paints both fish with `yin_colors`, and shows a single legend (the yang guide is dropped). Unless `yin_name` is supplied, the legend is titled "`yin / yang`". Ignored when custom `yin_scale` / `yang_scale` are given. Default `FALSE`.

`width, height` Width and height of each cell. Typically omitted.

`alpha` Alpha transparency for the fish fills. A single value for the whole layer (see the Styling section).

na.rm	If TRUE, silently removes rows with missing values.
colour	Outline colour of the fish. A single value for the whole layer (see the Styling section).
linewidth	Outline width of the fish (in mm). Replaces the deprecated size aesthetic of ggtaichi 0.1.0. A single value for the whole layer (see the Styling section).
linetype	Outline linetype of the fish. A single value for the whole layer (see the Styling section).
show.legend	Logical. Should the layer be included in the legend?
...	Additional arguments passed to *both* auto-built fill scales (e.g., shared limits or na.value). Because they go to both, an argument that suits only one kind of scale will be rejected by the other when yin and yang are of different types — for instance a numeric limits draws ggplot2's "Continuous limits supplied to discrete scale" warning from the discrete fish. For per-fish scale options, supply yin_scale / yang_scale instead. The scale arguments geom_taichi() fills in itself — name, values and colors / colours — are not accepted here; use yin_name / yang_name and yin_colors / yang_colors.

Value

A ggtaichi_plot object: the two fish layers plus the fill scales they need, ready to be added to a ggplot with +. It is not a plot on its own.

Discrete and continuous fills

geom_taichi() inspects the plot data at + time. A numeric yin / yang column gets a continuous scale_fill_gradientn built from yin_colors / yang_colors; a factor, character, or logical column (including computed expressions such as factor(week)) gets a discrete scale_fill_manual whose palette is interpolated from the same color vectors. With the default vectors the discrete palette skips the palest end of the ramp so that no category is invisible on a white panel; an explicitly supplied color vector is used as-is. Supply yin_scale / yang_scale to override the automatic choice entirely.

Because the choice is made when the layer is added, replacing the plot's data afterwards keeps the scales picked for the original data. Swapping in data of the same types is fine; if the new yin / yang columns are of the *other* kind, ggplot2 reports a "Discrete value supplied to a continuous scale" (or the reverse) at draw time — rebuild the plot rather than substituting its data.

Eyes

eyes = TRUE draws the classic taichi dots, each sitting in its own fish's head: the yin eye in the top bulb, the yang eye in the bottom bulb. The size and colour arguments accept either a constant or an (unquoted) data column, so the eyes can encode up to two further variables. A mapped eye-size column is rescaled to radii between 0.05 and 0.3 of the glyph radius, unless all its non-zero values already lie in (0, 0.5], in which case they are used directly as radius proportions. Cells whose eye size is NA or 0 are drawn without an eye, so a column may mix proportions with zeros to suppress individual eyes. A column whose values are all equal gets the midpoint radius, 0.175.

Styling

alpha, colour, linewidth and linetype are layer-wide constants here. Each has a concrete default, so geom_taichi() always passes it to both fish layers as a parameter, and a parameter takes precedence over an inherited mapping: a plot-level aes(linewidth = ...) (or alpha, colour, linetype) has no effect on the glyphs. To drive one of those from a column, build the layers yourself with geom_yin_fish() / geom_yang_fish(), which take all four as ordinary aesthetics.

width and height behave differently, because they default to NULL and are forwarded only when you actually supply them: a plot-level aes(width = ...) *does* size the cells per row. So the data-driven channels of geom_taichi() are yin, yang, angle, the two eyes, and width / height via aes().

Missing values

A fish whose fill value is NA is painted in the scale's na.value colour (pass e.g. na.value = "transparent" through ... to change it), while na.rm = TRUE silently drops rows with missing positions.

Examples

```
library(ggplot2)

# taichi with numeric fills

data <- data.frame(x = rep(c(1, 2, 3), 3),
                   y = rep(c(1, 2, 3), each = 3),
                   yin_values = 1:9,
                   yang_values = 9:1)

ggplot(data, aes(x, y)) +
  geom_taichi(yin = yin_values,
              yang = yang_values)

# categorical (discrete) fills are detected automatically

data$yin_class <- rep(c("low", "mid", "high"), 3)

ggplot(data, aes(x, y)) +
  geom_taichi(yin = yin_class,
              yang = yang_values)

# classic eyes, rotation, and data-driven eye sizes

ggplot(data, aes(x, y)) +
  geom_taichi(yin = yin_values,
              yang = yang_values,
              eyes = TRUE,
              yin_eye_size = yang_values,
              angle = 45)
```

geom_yin_fish *The individual taichi fish layers*

Description

‘geom_yin_fish()‘ and ‘geom_yang_fish()‘ each draw one of the two interlocking fish of a taichi symbol per ‘(x, y)‘ cell. They are the building blocks that [geom_taichi()] assembles (together with two fill scales and a [ggnewscale::new_scale_fill()] break); use them directly when you want full control — e.g. to bring your own fill scale for a single fish, to stack scales differently, or to draw only one source.

Usage

```
geom_yin_fish(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  width = NULL,
  height = NULL,
  eyes = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

geom_yang_fish(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  width = NULL,
  height = NULL,
  eyes = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping, data, stat, position, inherit.aes	See [ggplot2::layer()].
width, height	Cell size; defaults to the resolution of the data.
eyes	Logical. Draw the classic eye dot inside this fish’s head?

na.rm	If 'TRUE', silently removes rows with missing values.
show.legend	Logical. Should this layer be included in the legends?
...	Other arguments passed to [ggplot2::layer()]: either aesthetics used as constant parameters (e.g. 'eye_size = 0.2') or geom parameters.

Details

Both geoms understand the aesthetics 'x', 'y', 'fill', 'colour', 'linewidth', 'linetype', 'alpha', 'width', 'height', 'angle' (degrees, counter-clockwise), 'eye_size', and 'eye_colour' (the latter two only matter when 'eyes = TRUE'). At 'angle = 0' the yin fish is the left half of the circle plus the top bulb (its head); the yang fish is the right half plus the bottom bulb.

Value

A ggplot2 layer drawing one fish per cell.

Examples

```
library(ggplot2)
d <- data.frame(x = 1:3, y = 1, value = 1:3)

# a yin-only plot with an ordinary fill scale
ggplot(d, aes(x, y)) +
  geom_yin_fish(aes(fill = value)) +
  scale_fill_viridis_c()

# both fish, manually stacked with ggnewscale
ggplot(d, aes(x, y)) +
  geom_yin_fish(aes(fill = value)) +
  scale_fill_viridis_c(name = "yin") +
  ggnewscale::new_scale_fill() +
  geom_yang_fish(aes(fill = rev(value))) +
  scale_fill_viridis_c(name = "yang", option = "magma")
```

---

pitts_emojis *Popular Emojis*

Description

The most popular emoji of a given week in a given category from the Meltwater Tweet sample, as HTML <img> tags. The vector is aligned row-for-row with pitts_tg, so pitts_emojis[i] is the emoji for the week / category combination in row i of that data set. The tags are meant to be drawn as rich text, e.g. with ggtext::geom_richtext() or annotate("richtext", ...).

Usage

```
pitts_emojis
```

Format

A character vector of 270 HTML `<img>` tags (90 distinct emoji), one per row of `pitts_tg`.

Note on the image URLs

Each tag points at a remotely hosted PNG on the Emojipedia asset host used when the data was collected in 2020. Those objects are no longer served publicly, so the tags no longer render as pictures on their own. The emoji each tag refers to is still readable from its file name, which ends in the Unicode code point (for example `thinking-face_1f914.png` is U+1F914); substitute your own image paths or the literal emoji characters to draw them.

Source

The most frequent emoji per week and category in the Meltwater Twitter sample described in `pitts_tg`, processed by the package author.

`pitts_tg` *Pittsburgh COVID-related Google & Twitter incidence rates*

Description

A data set containing the 30-week incidence rates of COVID-related categories in the Pittsburgh Metropolitan Statistical Area (MSA), from week 1 beginning June 1, 2020 to week 30, which ended on the last Sunday of the year. The data columns are introduced below. One quick note about the columns of the data set: `week_start` is present for illustration purposes, as a reminder of what the `week` column counts. In other words, it does not participate in any visualization.

Usage

```
pitts_tg
```

Format

A data frame with 270 rows and 6 columns:

msa Metropolitan statistical area (Pittsburgh only).

week week 1 to week 30.

week_start The Monday date of the week started.

category One of 9 COVID-related categories: Covid, General Virus, Masks, Sanitizing, Social Distancing, Symptoms, Tests, Treatment, Working.

Twitter weekly tweets percentage (%) in the MSA falling into each category.

Google weekly Google search percentage (%) in the MSA falling into each category.

Source

Just like `states_tg`, Google is processed from Google Health API, and Twitter from Meltwater, a Twitter vendor. Both data sources are processed by the author of the package.

remove_padding *Remove ggplot2 default padding*

Description

ggplot2 pads both continuous and discrete axes with a little expansion, which can make a taichi grid look like it is floating. ‘remove_padding()‘ trims that space. Called with no arguments it inspects the plot it is added to and figures out for itself whether each axis is continuous or discrete; pass ‘"c"‘ (continuous) or ‘"d"‘ (discrete) explicitly to override the detection, e.g. when the axis mapping is a computed expression the plot data cannot answer for.

Usage

```
remove_padding(x = NULL, y = NULL, ...)
```

Arguments

x, y	‘NULL‘ (the default) to auto-detect the scale type of that axis from the plot’s data and mapping, ‘"c"‘ for a continuous axis, or ‘"d"‘ for a discrete one. Auto-detection reads the *plot’s* mapping, so name the type explicitly when ‘x‘ / ‘y‘ are mapped in a layer rather than in ‘ggplot()‘, or when the mapping is a computed expression the plot data cannot answer for.
...	Additional arguments passed on to the underlying [ggplot2::scale_x_continuous()] / [ggplot2::scale_x_discrete()] (and y) calls. They go to *both* scales, so when the two axes are of different types only arguments that continuous and discrete scales share (‘name‘, ‘breaks‘, ‘labels‘, ‘guide‘, ...) can be used here — a continuous-only argument such as ‘n.breaks‘ would be rejected by the discrete scale with an "unused argument" error. For per-axis options, add your own ‘scale_x_*(expand = c(0, 0))‘ call instead.

Value

An object that, added to a ggplot, replaces both position scales with padding-free ones.

Examples

```
library(ggplot2)
d <- data.frame(x = 1:3, y = c("a", "b", "c"), yin = 1:3, yang = 3:1)

# auto-detects x as continuous and y as discrete
ggplot(d, aes(x, y)) +
  geom_taichi(yin = yin, yang = yang) +
  remove_padding()

# explicit override, identical result here
ggplot(d, aes(x, y)) +
  geom_taichi(yin = yin, yang = yang) +
  remove_padding(x = "c", y = "d")
```

states_tg *States' COVID-related Google & Twitter incidence rates*

Description

A data set containing the 31-week incidence rates of COVID-related categories in 4 states (Florida, Missouri, New York, and Texas), from week 1 beginning June 1, 2020 to week 31, which begins December 28, 2020 and so runs a few days past the end of the year. The data columns are introduced below. One quick note about the columns of the data set: `week_start` is present for illustration purposes, as a reminder of what the `week` column counts. In other words, it does not participate in any visualization.

Usage

```
states_tg
```

Format

A data frame with 1116 rows and 6 columns:

state One of the four states: Florida, Missouri, New York, Texas.

week week 1 to week 31.

week_start The Monday date of the week started.

category One of 9 COVID-related categories: Covid, General Virus, Masks, Sanitizing, Social Distancing, Symptoms, Tests, Treatment, Working.

Twitter weekly tweets percentage (%) in state falling into each category.

Google weekly Google search percentage (%) in state falling into each category.

Source

Just like `pitts_tg`, Google is processed from Google Health API, and Twitter from Meltwater, a Twitter vendor. Both data sources are processed by the author of the package.

theme_taichi *Plot Themes*

Description

A light theme tuned for the taichi grid: it bottoms the legends, drops the panel grid and axis ticks, and gives the canvas a soft off-white background reminiscent of rice paper.

Usage

```
theme_taichi(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)
```

Arguments

`base_size`	base font size
`base_family`	base font family
`base_line_size`	base size for line elements
`base_rect_size`	base size for rect elements

Value

A `theme` object that can be added to any ggplot, in the same way as `theme_bw()`.

Opinionated choices

Two of the theme's settings surprise people often enough to be worth spelling out. The y axis *title* is blanked, on the assumption that the y axis of a taichi grid is a list of category names that already reads as a label — so `labs(y = "...")` has no visible effect under this theme. Legend text is rotated 90 degrees, which keeps a wide continuous legend from running off the bottom of the plot. Both are ordinary theme elements, so add a `theme()` call afterwards to put them back:

```
  + theme_taichi() + theme(axis.title.y = element_text(),
                           legend.text = element_text(angle = 0))
```

Examples

```
library(ggplot2)
d <- data.frame(x = 1:3, y = 1:3, yin = 1:3, yang = 3:1)

ggplot(d, aes(x, y)) +
  geom_taichi(yin = yin, yang = yang) +
  theme_taichi()
```

Index