Package 'rasterly'

June 8, 2020

Title Easily and Rapidly Generate Raster Image Data with Support for 'Plotly.js'

Version 0.2.0

Description
It aims to easily and rapidly generate raster data in R, even for very large datasets, with an aesthetics-based mapping syntax that should be familiar to users of the 'ggplot2' package. While 'rasterly' does not attempt to reproduce the full functionality of the 'Datashader' graphics pipeline system for Python, the 'rasterly' API has several core elements in common with that software package.

LinkingTo Rcpp

License MIT + file LICENSE

Encoding UTF-8

ByteCompile true

KeepSource true

BugReports `https://github.com/plotly/rasterly/issues`

Depends R (>= 3.4.0), methods, Rcpp

Imports data.table, rlang, plotly, ggplot2, magrittr, grid, stats

Suggests covr, testthat, knitr, rmarkdown, lubridate

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation yes

Author Zehao Xu [aut, cre],
Ryan Patrick Kyle [ctb] (<https://orcid.org/0000-0001-5829-9867>),
Plotly Technologies [cph]

Maintainer Zehao Xu <`z267xu@uwaterloo.ca`>

Repository CRAN

Date/Publication 2020-06-08 13:00:07 UTC

R topics documented:

`add_rasterly` *Add "rasterly" trace to a Plotly visualization*

Description

Add trace to a Plotly visualization.

Usage

```
add_rasterly_heatmap(
  p,
  x = NULL,
  y = NULL,
  z = NULL,
  ...,
  data = NULL,
  inherit = TRUE,
  on = NULL,
  size = NULL,
  scaling = NULL
)

add_rasterly_image(
  p,
  x = NULL,
  y = NULL,
  z = NULL,
    ...,
    data = NULL,
    inherit = TRUE,
    color = NULL,
    on = NULL,
    size = NULL
  )
```

Arguments

`p`	A `plotly` object
`x`	Numeric vector or expression. The x variable, to be passed on to `aes()`.
`y`	Numeric or expression. The y variable, to be passed on to `aes()`.
`z`	Numeric. A numeric matrix (optional), to be processed with `add_heatmap`.
`...`	Arguments (i.e., attributes) passed along to the trace type or `rasterly`.
`data`	A data.frame or SharedData object (optional).
`inherit`	Logical. Inherit attributes from plotly?
`on`	Numeric vector or expression. Provides the data on which to reduce, to be passed on to `aes()`.
`size`	Numeric vector or expression. Pixel size for each observation, to be passed on to `aes()`.
`scaling`	Character string or function. The scaling method to be used for the trace.
`color`	Numeric vector or expression. Pixel color for each observation, to be passed on to `aes()`.

Examples

```
## Not run:
if(requireNamespace("plotly") && requireNamespace("data.table") &&
  requireNamespace("lubridate")) {
 # Load data
 url1 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data1.csv"
 ridesRaw_1 <-  url1 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url2 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data2.csv"
 ridesRaw_2 <-  url2 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url3 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data3.csv"
 ridesRaw_3 <-  url3 %>%
   data.table::fread(stringsAsFactors = FALSE)
 ridesDf <- list(ridesRaw_1, ridesRaw_2, ridesRaw_3) %>%
   data.table::rbindlist()
 time <- lubridate::ymd_hms(ridesDf$`Date/Time`)
 ridesDf <-  ridesDf[, 'Date/Time':=NULL][, list(Lat,
                                                 Lon,
                                                 hour = lubridate::hour(time),
                                                 month = lubridate::month(time),
                                                 day = lubridate::day(time))]
############################# add_rasterly_heatmap #############################
#### quick start
p <- plot_ly(data = ridesDf) %>%
       add_rasterly_heatmap(x = ~Lat, y = ~Lon)
p
#### set artificial scaling function
zeroOneTransform <- function(z) {
  minz <- min(z)
  maxz <- max(z)
  M <- matrix((z - minz)/(maxz - minz), nrow = dim(z)[1])
  return(M)
}
plot_ly(data = ridesDf) %>%
  add_rasterly_heatmap(x = ~Lat,
               y = ~Lon,
               on = ~-Lat,
               reduction_func = "max",
               scaling = zeroOneTransform) %>%
  plotly::layout(
    xaxis = list(
      title = "x"
    ),
    yaxis = list(
      title = "y"
    )
  )
############################# add_rasterly_image #############################
p <- plot_ly(data = ridesDf) %>%
       add_rasterly_image(x = ~Lat, y = ~Lon, color = ~hour,
                          # even `color_map` is deprecated,
                          # it is still a good way to specify the color mapping
                          color_map = hourColors_map,
                          plot_width = 400, plot_height = 400)
p
}

## End(Not run)
```

color_map *Supplemental color maps for rasterly*

Description

Hex codes for the color map. Used in setting argument color in rasterly or rasterly layers.

Usage

```
fire_map

viridis_map
```

```
hourColors_map
```

Format

An object of class `character` of length 256.

An object of class `character` of length 256.

An object of class `character` of length 24.

`extract` *Extract or replace parts of a* `rasterly` *object*

Description

The `extract` function provides functionality for updating existing `rasterly` objects.

Usage

```
## S3 method for class 'rasterly'
x[name]

## S3 replacement method for class 'rasterly'
x[name, ...] <- value
```

Arguments

`x`	Object from which to extract element(s) or in which to replace element(s).
`name`	Character. A literal string to be extracted from `x`. See details for more information.
`...`	(missing) or NULL.
`value`	values to replace; typically an array-like R object of a similar class as `x`.

Details

Available names:

- Aggregation: "data", "mapping", "plot_width", "plot_height", "range", "x_range", "y_range", "xlim", "ylim", "aesthetics", "reduction_func", "glyph", "max_size", "group_by_data_table", "drop_data", "variable_check"
- Display: "background", "color", "alpha", "span", "show_raster", "layout"

Set `level` in `...`. `level` is numeric used for specifing level of ‘rasterly‘ object to modify; default is 1 for the parent layer (`rasterly()`).

Examples

```
library(rasterly)
r <- rasterly(
     data = data.frame(x = 1:1e4, y = runif(1e4), category = sample(1:4, 1e4, replace = TRUE)),
       mapping = aes(x = x, y = y)
) %>%
  rasterly_points(xlim = c(1, 5000)) %>%
  rasterly_points(
    mapping = aes(x = x, y = y, color = category),
    xlim = c(5001, 1e4)
  )
r["mapping"]
r["xlim"]

# reassign parent `rasterly()` mapping
r["mapping"] <- aes(x = x, y = y, color = category)
r["mapping"]

# reassign all mapping systems
r["mapping", level = 1:length(r)] <- aes(x = x, y = y)
r["mapping"]
```

---

ggRasterly *ggRasterly*

Description

Display large data set in `ggplot`.

Usage

```
ggRasterly(
  data = NULL,
  mapping = aes(),
  ...,
  plot_width = 600,
  plot_height = 600,
  x_range = NULL,
  y_range = NULL,
  background = "white",
  color = NULL,
  show_raster = TRUE,
  drop_data = FALSE,
  variable_check = FALSE,
  alpha = 0.5,
  shape = 15,
  point_size = 0.5
)
```

Arguments

`data`	Dataset to use for generating the plot. If not provided, data must be supplied in each layer of the plot. For best performance, particularly when processing large datasets, use of data.table is recommended.
`mapping`	Default list of aesthetic mappings to use for plot. The same with `ggplot2` aes. See details.
`...`	Other arguments which will be passed through to layers.
`plot_width`	Integer. The width of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`plot_height`	Integer. The height of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`x_range`	Vector of type numeric. The range of `x`; it can be used to clip the image. For larger datasets, providing `x_range` may result in improved performance.
`y_range`	Vector of type numeric. The range of `y`; it can be used to clip the image. For larger datasets, providing `y_range` may result in improved performance.
`background`	Character. The background color of the image to plot.
`color`	Vector of type character. It will determine this color vector is a `color_map` or `color_key` automatically. • color_map: It has Color(s) used to draw each pixel. The `color_map` is extended by linear interpolation independently for RGB. The darkness of the mapped color depends upon the values of the aggregation matrix. • color_key: Vector of type character. The `color_key` is used for categorical variables; it is passed when the `color` aesthetic is provided.
`show_raster`	Logical. Should the raster be displayed?
`drop_data`	Logical. When working with large datasets, drops the original data once processed according to the provided `aes()` parameters, using the `remove()` function. See details for additional information.
`variable_check`	Logical. If `TRUE`, drops unused columns to save memory; may result in reduced performance.
`alpha`	The transparency of points, from 0 to 1.
`shape`	The shape of points, see pch.
`point_size`	The size of points.

Value

a ‘ggplot‘ object

See Also

`plotRasterly`, `plot.rasterly`

Examples

```
## Not run:
if(requireNamespace("ggplot2") && requireNamespace("data.table") &&
  requireNamespace("lubridate")) {
 # Load data
 url1 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data1.csv"
 ridesRaw_1 <-  url1 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url2 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data2.csv"
 ridesRaw_2 <-  url2 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url3 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data3.csv"
 ridesRaw_3 <-  url3 %>%
   data.table::fread(stringsAsFactors = FALSE)

 ridesDf <- list(ridesRaw_1, ridesRaw_2, ridesRaw_3) %>%
   data.table::rbindlist()

 time <- lubridate::ymd_hms(ridesDf$`Date/Time`)
 ridesDf <-  ridesDf[, 'Date/Time':=NULL][, list(Lat,
                                                 Lon,
                                                 hour = lubridate::hour(time),
                                                 month = lubridate::month(time),
                                                 day = lubridate::day(time))]

 # continuous variable legend
 ggRasterly(data = ridesDf,
            mapping = aes(x = Lat, y = Lon),
            color = fire_map
 )
 # discreate variable legend
 ggRasterly(data = ridesDf,
            mapping = aes(x = Lat, y = Lon, color = hour),
            color = hourColors_map
 ) +
 ggplot2::labs(title = "New York Uber",
               subtitle = "Apr to Sept, 2014",
               caption =
                 "https://raw.githubusercontent.com/plotly/datasets/master")
 }

## End(Not run)
```

`image2data` *Image raster to data frame.*

Description

Transform a image raster to a data frame.

Usage

```
image2data(x, background = "white", x_range = NULL, y_range = NULL)
```

Arguments

x	It could be a rasterly object or a raster image.
background	The background of image raster.
x_range	The range represents image width.
y_range	The range represents image height.

Value

a `data.table` object

See Also

ggRasterly

Examples

```
x <- rnorm(1000, mean = 10)
y <- rnorm(1000, mean = 20)
color <- sample(1:5, 1000, replace = TRUE)
rastObj <- data.frame(x = x, y = y, color = color) %>%
       rasterly(mapping = aes(x = x, y = y, color = color)) %>%
       rasterly_points()
p <- rasterly_build(rastObj)
dt <- image2data(p)
if(requireNamespace("ggplot2")) {
  # Note that each point represents a single pixel in the image
  ggplot2::ggplot(dt, mapping = aes(x = x, y = y)) +
    ggplot2::geom_point(color = dt$color, size = 0.5)
}
```

`is.rasterly` *Is* `rasterly`

Description

Reports whether x is a `rasterly` object.

Usage

```
is.rasterly(x)
```

Arguments

x	a `rasterly` object

is.rasterlyBuild *Is* rasterlyBuild

Description

Reports whether x is a rasterlyBuild object. In other word, it helps to define whether this object has been passed through ‘rasterly_build‘

Usage

```
is.rasterlyBuild(x)
```

Arguments

x	a rasterly object

plotRasterly *plotRasterly*

Description

Display large data set in plotly

Usage

```
plotRasterly(
  data = NULL,
  mapping = aes(),
  ...,
  plot_width = 400,
  plot_height = 400,
  x_range = NULL,
  y_range = NULL,
  background = "white",
  color = NULL,
  show_raster = TRUE,
  drop_data = FALSE,
  variable_check = FALSE,
  alpha = 0.5,
  shape = 19,
  point_size = 0.5,
  as_image = FALSE,
  sizing = c("stretch", "fill", "contain")
)
```

Arguments

`data`	Dataset to use for generating the plot. If not provided, data must be supplied in each layer of the plot. For best performance, particularly when processing large datasets, use of data.table is recommended.
`mapping`	Default list of aesthetic mappings to use for plot. The same with `ggplot2` aes. See details.
`...`	Other arguments which will be passed through to layers.
`plot_width`	Integer. The width of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`plot_height`	Integer. The height of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`x_range`	Vector of type numeric. The range of `x`; it can be used to clip the image. For larger datasets, providing `x_range` may result in improved performance.
`y_range`	Vector of type numeric. The range of `y`; it can be used to clip the image. For larger datasets, providing `y_range` may result in improved performance.
`background`	Character. The background color of the image to plot.
`color`	Vector of type character. It will determine this color vector is a `color_map` or `color_key` automatically. • color_map: It has Color(s) used to draw each pixel. The `color_map` is extended by linear interpolation independently for RGB. The darkness of the mapped color depends upon the values of the aggregation matrix. • color_key: Vector of type character. The `color_key` is used for categorical variables; it is passed when the `color` aesthetic is provided.
`show_raster`	Logical. Should the raster be displayed?
`drop_data`	Logical. When working with large datasets, drops the original data once processed according to the provided `aes()` parameters, using the `remove()` function. See details for additional information.
`variable_check`	Logical. If `TRUE`, drops unused columns to save memory; may result in reduced performance.
`alpha`	The transparency of points, from 0 to 1.
`shape`	The shape of points, see pch.
`point_size`	The size of points.
`as_image`	Logical value. If `FALSE`, image raster will be transformed into a data frame, hence a points layer would be pipped on `plotly`; conversely, a raster layer will be added.
`sizing`	It affects only with `as_image = TRUE`. Specifies which dimension of the image to constrain. One of "stretch" "fill", "contain". see https://plot.ly/r/reference/#Layout_and_layout_style_obje

Value

a `plotly` widget

See Also

ggRasterly, plot.rasterly

Examples

```
## Not run:
 library(rasterly)
 if(requireNamespace("plotly") &&
    requireNamespace("data.table") &&
    requireNamespace("lubridate")) {
   # Load data
 url1 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data1.csv"
 ridesRaw_1 <-  url1 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url2 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data2.csv"
 ridesRaw_2 <-  url2 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url3 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data3.csv"
 ridesRaw_3 <-  url3 %>%
   data.table::fread(stringsAsFactors = FALSE)
 ridesDf <- list(ridesRaw_1, ridesRaw_2, ridesRaw_3) %>%
   data.table::rbindlist()

 time <- lubridate::ymd_hms(ridesDf$`Date/Time`)
 ridesDf <-
   ridesDf[, 'Date/Time':=NULL][, list(Lat,
              Lon,
              hour = lubridate::hour(time),
              month = lubridate::month(time),
              day = lubridate::day(time))]
 # A point layer is added
 plotRasterly(data = ridesDf,
              mapping = aes(x = Lat, y = Lon, color = hour),
              color = hourColors_map,
              as_image = FALSE)
  # An image layer is added
  plotRasterly(data = ridesDf,
               mapping = aes(x = Lat, y = Lon, color = hour),
               color = hourColors_map,
               as_image = TRUE)

 }

## End(Not run)
```

`rasterize_points` *rasterize_points*

Description

Points layer for "rasterly". Deprecated now, please use `rasterly_points` instead.

Usage

```
rasterize_points(
  rastObj,
  data = NULL,
  mapping = aes(),
  ...,
  xlim = NULL,
  ylim = NULL,
  max_size = NULL,
  reduction_func = NULL,
  layout = NULL,
  glyph = NULL,
  group_by_data_table = NULL,
  inherit.aes = TRUE
)
```

Arguments

`rastObj`	A `rasterly` object.
`data`	A `data.frame` or `function` with an argument `x`, specifying the dataset to use for plotting. If `data` is `NULL`, the `data` argument provided to `rasterly` may be passed through.
`mapping`	Default list of aesthetic mappings to use for plot. If provided and `inherit.aes = TRUE`, it will be stacked on top of the mappings passed to `rasterly`.
`...`	Pass-through arguments provided by `rasterly`.
`xlim`	Vector of type numeric. X limits in this layer.
`ylim`	Vector of type numeric. Y limits in this layer.
`max_size`	Numeric. When size changes, the upper bound of the number of pixels over which to spread a single observation.
`reduction_func`	Function. A reduction function is used to aggregate data points into their pixel representations. Currently supported reduction operators are `sum`, `any`, `mean`, `m2`, `first`, `last`, `min` and `max`. Default is `sum`. See details.
`layout`	Character. The method used to generate layouts for multiple images. The default is `weighted`. Useful for categorical data (i.e. "color" is provided via `aes()`). `weighted` specifies that the final raster should be a weighted combination of each (categorical) aggregation matrix. Conversely, `cover` indicates that the afterwards objects will be drawn on top of the previous ones.
`glyph`	Character. Currently, only "circle" and "square" are supported; as the `size` of the pixels increases, how should they spread out – should the pattern be circular or square? Other glyphs may be added in the future.
`group_by_data_table`	Logical. Default is `TRUE`; when "color" is provided via `aes()`, the "group by" operation may be perfromed within `data.table` or natively within `rasterly`. Generally, `group_by_data_table = TRUE` is faster, but for very large datasets grouping within `rasterly` may offer better performance.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them.

See Also

rasterly_points

`rasterly` *Easily and rapidly generate raster image data with support for Plotly.js*

Description

Create a rasterly object, to which aggregation layers may be added. This function is the first step in the process to generate raster image data using the `rasterly` package.

Usage

```
rasterly(
  data = NULL,
  mapping = aes(),
  ...,
  plot_width = 600,
  plot_height = 600,
  x_range = NULL,
  y_range = NULL,
  background = "white",
  color = NULL,
  show_raster = TRUE,
  drop_data = FALSE,
  variable_check = FALSE
)
```

Arguments

`data`	Dataset to use for generating the plot. If not provided, data must be supplied in each layer of the plot. For best performance, particularly when processing large datasets, use of data.table is recommended.
`mapping`	Default list of aesthetic mappings to use for plot. The same with `ggplot2` aes. See details.
`...`	Other arguments which will be passed through to layers.
`plot_width`	Integer. The width of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`plot_height`	Integer. The height of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`x_range`	Vector of type numeric. The range of `x`; it can be used to clip the image. For larger datasets, providing `x_range` may result in improved performance.
`y_range`	Vector of type numeric. The range of `y`; it can be used to clip the image. For larger datasets, providing `y_range` may result in improved performance.

background	Character. The background color of the image to plot.
color	Vector of type character. It will determine this color vector is a `color_map` or `color_key` automatically. • color_map: It has Color(s) used to draw each pixel. The `color_map` is extended by linear interpolation independently for RGB. The darkness of the mapped color depends upon the values of the aggregation matrix. • color_key: Vector of type character. The `color_key` is used for categorical variables; it is passed when the `color` aesthetic is provided.
show_raster	Logical. Should the raster be displayed?
drop_data	Logical. When working with large datasets, drops the original data once processed according to the provided `aes()` parameters, using the `remove()` function. See details for additional information.
variable_check	Logical. If `TRUE`, drops unused columns to save memory; may result in reduced performance.

Details

- The rasterly package currently supports five aesthetics via `aes()`: `x`, `y`, `on`, `color`, and `size`. The "on" aesthetic specifies the variable upon which the reduction function should be applied to generate the raster data.
- `drop_data` can help save space, particularly when large datasets are used. However, dropping the original dataset may result in errors when attempting to set or update `aes()` parameters within rasterly layers.

Value

An environment wrapped by a list which defines the properties of the raster data to be generated.

Note

Calling `rasterly()` without providing `rasterly_...()` layers has no effect. More info can be found in README.md

See Also

rasterly_points, rasterly_build, [.rasterly, [<-.rasterly ggRasterly, plotRasterly

Examples

```
## Not run:
 if(requireNamespace("data.table")) {
 url1 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data1.csv"
 ridesRaw_1 <-  url1 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url2 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data2.csv"
 ridesRaw_2 <-  url2 %>%
   data.table::fread(stringsAsFactors = FALSE)
 url3 <- "https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data3.csv"
```

```
  ridesRaw_3 <-  url3 %>%
    data.table::fread(stringsAsFactors = FALSE)
  ridesDf <- list(ridesRaw_1, ridesRaw_2, ridesRaw_3) %>%
    data.table::rbindlist()

  ridesDf %>%
        rasterly(mapping = aes(x = Lat, y = Lon)) %>%
        rasterly_points()
  }
## End(Not run)
```

rasterly_build *rasterly_build*

Description

Produce a rasterly object and return the raster information required to produce an image

Usage

```
rasterly_build(rastObj)
```

Arguments

`rastObj`	A rasterly object. It should be a list of environments composed of a `rasterly()` and several `rasterly_...` layers.

Note

A rasterly object will never be produced until `rasterly_build()` is called.

See Also

rasterly, rasterly_points, [.rasterly, [<-.rasterly

Examples

```
r <- data.frame(x = rnorm(1e5), y = rnorm(1e5)) %>%
       rasterly(mapping = aes(x = x, y = y)) %>%
       rasterly_points(color = fire_map)
str(r)
p <- rasterly_build(r)
str(p)
```

`rasterly_guides` *rasterly_guides*

Description

Guides layer for "rasterly".

Usage

```
rasterly_guides(
  rastObj,
  x_pretty = NULL,
  y_pretty = NULL,
  panel_background = "grey92",
  panel_line = "white"
)
```

Arguments

`rastObj`	A "rasterly" object.
`x_pretty`	The pretty on x. Compute a sequence of about n+1 equally spaced 'round' values which cover the range of the values in x. If it is not provided, `x_pretty` will be generated by the x range
`y_pretty`	The pretty on y.
`panel_background`	Panel background.
`panel_line`	Panel line color

Details

When an image has a 'complicated' background, the drawing time increases significantly. So it is not recommended. A suggestion to draw grid guides is to transform image data to a data frame via image2data, then use `ggplot` or `plotly` to display.

See Also

ggRasterly

`rasterly_points` *rasterly_points*

Description

Points layer for `rasterly`.

Usage

```
rasterly_points(
  rastObj,
  data = NULL,
  mapping = aes(),
  ...,
  xlim = NULL,
  ylim = NULL,
  max_size = NULL,
  reduction_func = NULL,
  layout = NULL,
  glyph = NULL,
  group_by_data_table = NULL,
  inherit.aes = TRUE
)
```

Arguments

`rastObj`	A `rasterly` object.
`data`	A `data.frame` or `function` with an argument `x`, specifying the dataset to use for plotting. If `data` is `NULL`, the `data` argument provided to `rasterly` may be passed through.
`mapping`	Default list of aesthetic mappings to use for plot. If provided and `inherit.aes = TRUE`, it will be stacked on top of the mappings passed to `rasterly`.
`...`	Pass-through arguments provided by `rasterly`.
`xlim`	Vector of type numeric. X limits in this layer.
`ylim`	Vector of type numeric. Y limits in this layer.
`max_size`	Numeric. When size changes, the upper bound of the number of pixels over which to spread a single observation.
`reduction_func`	Function. A reduction function is used to aggregate data points into their pixel representations. Currently supported reduction operators are `sum`, `any`, `mean`, `m2`, `first`, `last`, `min` and `max`. Default is `sum`. See details.
`layout`	Character. The method used to generate layouts for multiple images. The default is `weighted`. Useful for categorical data (i.e. "color" is provided via `aes()`). `weighted` specifies that the final raster should be a weighted combination of each (categorical) aggregation matrix. Conversely, `cover` indicates that the afterwards objects will be drawn on top of the previous ones.

`glyph`	Character. Currently, only "circle" and "square" are supported; as the `size` of the pixels increases, how should they spread out – should the pattern be circular or square? Other glyphs may be added in the future.
`group_by_data_table`	Logical. Default is `TRUE`; when "color" is provided via `aes()`, the "group by" operation may be perfromed within `data.table` or natively within `rasterly`. Generally, `group_by_data_table = TRUE` is faster, but for very large datasets grouping within `rasterly` may offer better performance.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them.

Details

Reduction functions

- `sum`: If `on` is not provided within `aes()`, the default is to take the sum within each bin. When `on` is specified, the function reduces by taking the sum of all elements within the variable named in `on`.
- `any`: When `on` is provided within `aes()`, the `any` reduction function specifies whether any elements in `on` should be mapped to each bin.
- `mean`: If `on` is not provided in mapping `aes()`, `on` would be set as variable "`y`" by default. When `on` is given, the `mean` reduction function takes the mean of all elements within the variable specified by `on`.
- `m2`: Requires that `on` is specified within `aes()`. The `m2` function computes the sum of square differences from the mean of all elements in the variable specified by `on`.
- `var`: Requires that `on` is specified within `aes()`. The `var` function computes the variance over all elements in the vector specified by `on`.
- `sd`: Requires that `on` is specified within `aes()`. The `sd` function computes the standard deviation over all elements in the vector specified by `on`.
- `first`: Requires that `on` is specified within `aes()`. The `first` function returns the first element in the vector specified by `on`.
- `last`: Requires that `on` is specified within `aes()`. The `last` function returns the last element in the vector specified by `on`.
- `min`: Requires that `on` is specified within `aes()`. The `min` function returns the minimum value in the vector specified by `on`.
- `max`: Requires that `on` is specified within `aes()`. The `min` function returns the maximum value in the vector specified by `on`.

Value

A list of environments.

See Also

rasterly, rasterly_build, [.rasterly, [<-.rasterly

Examples

```
## Not run:
   library(rasterly)
   if(requireNamespace("grid") && requireNamespace("gridExtra")) {
     x <- rnorm(1e7)
     y <- rnorm(1e7)
     category <- sample(1:5, 1e7, replace = TRUE)
     data.frame(x = x, y = y, category = category) %>%
       rasterly(mapping = aes(x = x, y = y, color = category)) %>%
       rasterly_points(layout = "weighted") -> ds1
     ds1
     # layout with cover
     data.frame(x = x, y = y, category = category) %>%
       rasterly(mapping = aes(x = x, y = y, color = category)) %>%
       rasterly_points(layout = "cover") -> ds2
     ds2
     # display side by side
     grid::grid.newpage()
     gridExtra::grid.arrange(
        grobs = list(rasterlyGrob(ds1), rasterlyGrob(ds2)),
        ncol = 2,
        top = "'weighted' layout versus 'cover' layout"
     )
   }

## End(Not run)
```

`rplot` *Rasterly plot*

Description

`rplot` is created to generate `rasterly` plot quickly but with base plot design. It is convenient but lacks flexibility and rasterly is highly recommended for a more versatile method.

Usage

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

## Default S3 method:
rplot(
  x,
  y = NULL,
  ...,
  plot_width = 600,
  plot_height = 600,
  x_range = NULL,
  y_range = NULL,
    background = "white",
    reduction_func = NULL,
    layout = NULL,
    glyph = NULL
  )
```

Arguments

`x, y`	Coordinates x, y for the plot.
`...`	Other `rasterly` arguments to pass through.
`plot_width`	Integer. The width of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`plot_height`	Integer. The height of the image to plot; must be a positive integer. A higher value indicates a higher resolution.
`x_range`	Vector of type numeric. The range of `x`; it can be used to clip the image. For larger datasets, providing `x_range` may result in improved performance.
`y_range`	Vector of type numeric. The range of `y`; it can be used to clip the image. For larger datasets, providing `y_range` may result in improved performance.
`background`	Character. The background color of the image to plot.
`reduction_func`	Function. A reduction function is used to aggregate data points into their pixel representations. Currently supported reduction operators are `sum`, `any`, `mean`, `m2`, `first`, `last`, `min` and `max`. Default is `sum`. See details.
`layout`	Character. The method used to generate layouts for multiple images. The default is `weighted`. Useful for categorical data (i.e. "color" is provided via `aes()`). `weighted` specifies that the final raster should be a weighted combination of each (categorical) aggregation matrix. Conversely, `cover` indicates that the afterwards objects will be drawn on top of the previous ones.
`glyph`	Character. Currently, only "circle" and "square" are supported; as the `size` of the pixels increases, how should they spread out – should the pattern be circular or square? Other glyphs may be added in the future.

Details

`rasterly` arguments are passed through via `...`. But some of them are noticeable.

- `size`: Size can be either a specified size (1, 2, 3, etc) or a mapping variable. Since `rasterly` does not provide point to point display, if the length of input `size` is the same with the length of `x` (or `y`). It will be treated as a mapping variable.
- `color`: Color can be either a color map vector or a mapping variable. If the length of `color` is equal to the length of `x` (or `y`). It will be treated as a mapping variable.
- on: On is always treated as a mapping variable.

See Also

rasterly rasterly_points

Examples

```
if(requireNamespace("ggplot2")) {
  library(ggplot2)
  # `color` represents a variable here
  with(diamonds,
       rplot(x = carat, y = price, color = color)
  )
  # `color` represents an actual color vector
  with(diamonds,
       rplot(x = carat, y = price, color = fire_map)
  )
}
```

`static` *Annotate and customize rasterly figures*

Description

Create a static plot based on `rasterly` object. This function allows users to add axes, legends and other descriptive details when generating ‘rasterly‘ objects.

Usage

```
rasterlyGrob(
  rasterlyObj,
  xlim = NULL,
  ylim = NULL,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  sub = NULL,
  interpolate = FALSE,
  axes = TRUE,
  legend = TRUE,
  legend_label = NULL,
  legend_layer = 1,
  legend_main = NULL,
  axes_gpar = grid::gpar(col = "black", cex = 1),
  label_gpar = grid::gpar(col = "black", cex = 1),
  main_gpar = grid::gpar(col = "black", cex = 1.5),
  legend_gpar = grid::gpar(col = "black", cex = 1.5),
  name = NULL,
  gp = NULL,
  vp = NULL
)

grid.rasterly(
    rasterlyObj,
    interpolate = FALSE,
    axes = TRUE,
    xlim = NULL,
    ylim = NULL,
    xlab = NULL,
    ylab = NULL,
    main = NULL,
    sub = NULL,
    legend = TRUE,
    legend_label = NULL,
    legend_layer = 1,
    legend_main = NULL,
    axes_gpar = grid::gpar(col = "black", cex = 1),
    label_gpar = grid::gpar(col = "black", cex = 1),
    main_gpar = grid::gpar(col = "black", cex = 1.5),
    legend_gpar = grid::gpar(col = "black", cex = 1.5),
    name = NULL,
    gp = NULL,
    vp = NULL,
    ...
  )

  ## S3 method for class 'rasterly'
  plot(
    x,
    y = NULL,
    xlim = NULL,
    ylim = NULL,
    xlab = NULL,
    ylab = NULL,
    main = NULL,
    legend_main = NULL,
    sub = NULL,
    interpolate = FALSE,
    axes = TRUE,
    legend = TRUE,
    legend_label = NULL,
    legend_layer = 1,
    new.page = TRUE,
    ...
  )

  ## S3 method for class 'rasterly'
  print(x, ...)
```

Arguments

`rasterlyObj`	A rasterly object.

xlim	Numeric; the x limits (x1, x2) of the plot. Default is NULL.
ylim	Numeric; the y limits (y1, y2) of the plot. Default is NULL.
xlab	Character; the label to be used for the x axis. Default is NULL.
ylab	Character; the label to be used for the y axis. Default is NULL.
main	Character; the title to be used for the plot. Default is NULL.
sub	sub Character; a subtitle for the plot. Default is NULL.
interpolate	Logical. Linearly interpolates the image if TRUE. Default is FALSE.
axes	Logical; should axes be drawn? Default is TRUE, set to FALSE to hide axes.
legend	Logical. Show a figure legend? Default is TRUE; set to FALSE to hide the legend.
legend_label	Character. The label to apply to the figure legend. Default is NULL, which omits the figure legend label.
legend_layer	Numeric. Specify the layer level within the rasterly object. The default layer level is ‘1‘, which represents the uppermost layer.
legend_main	Character. The main title to use within the figure legend. The default is NULL, which omits the figure legend title.
axes_gpar	Object of class gpar. This graphical parameter (gpar) controls axis color, size, and other aesthetics.
label_gpar	Object of class gpar. This graphical parameter (gpar) controls label color, size, and other aesthetics.
main_gpar	Object of class gpar. This graphical parameter (gpar) controls the main title's color, size, and other aesthetics.
legend_gpar	Object of class gpar. This graphical parameter (gpar) controls the legend's color, size, and other aesthetics.
name	Character. An identifier used to locate the grob within the display list and/or as a child of another grob.
gp	A gpar object, typically the output from a call to the function grid::gpar. This argument represents a list of graphical parameter settings.
vp	Object of class viewport. If provided, rasterlyGrob will pass this argument through to grob. Default is NULL.
...	Other arguments to modify the display.
x	A rasterly object
y	NULL, will be ignored.
new.page	display on a new page or not.

Details

We provide three functions to produce static graphics, which is based on the API of grid, plot and print.

- grid: The rasterlyGrob and grid.rasterly are the most flexible data structure. These functions produce a **grob** object. Users can modify the existing display by the functions provided by grid.

- plot.rasterly: The usage of this S3 method is very similar to the classic plot function. Users can set axis limits via xlim and ylim, as well as the corresponding labels using xlab and ylab, among other attributes.
- print.rasterly: This S3 method returns only a basic image raster.

See Also

plotRasterly, ggRasterly

Examples

```
if(requireNamespace("grid")) {
  data <- data.frame(x = rnorm(1e6),
                     y = rexp(1e6, 10))
  # a rasterly object
  rasterlyObj <- data %>%
                   rasterly(mapping = aes(x = x, y = y)) %>%
                   rasterly_points()
  # Generate a grob
  rg <- rasterlyGrob(rasterlyObj)
  ## get the raster grob by `grid::getGrob()`
  grid::getGrob(rg, "raster")
  grid::grid.newpage()
  grid::grid.draw(rg)
  # or
  grid::grid.newpage()
  grid.rasterly(rasterlyObj)
  # or `plot`
  plot(rasterlyObj, xlab = "rnorm(1e6)",
       ylab = "rexp(1e6, 10)",
       main = "This is an arbitrary plot")
  # or simply print
  rasterlyObj
  ## it is equivalent to `print(rasterlyObj)`
}
```

%<-% *Merge operator*

Description

Merge two objects from right to left.

Usage

```
x %<-% y
```

Arguments

x	A named list or vector
y	A named list or vector. Any duplicated names are detected in x will be covered by y

Value

a list

Examples

```
# two lists
x <- list(a = 1, b = "foo", c = 3)
y <- list(b = 2, d = 4)
x %<-% y
y %<-% x

# one list and one vector
x <- c(foo = 1, bar = 2)
y <- list(foo = "foo")
x %<-% y
y %<-% x

# two vectors
x <- c(a = 1, b = "foo", c = 3)
y <- c(b = 2, d = 4)
x %<-% y
y %<-% x

# duplicated names in x
x <- list(a = 1, b = "foo", b = 3)
y <- list(b = 2, d = 4)
x %<-% y
y %<-% x # be careful, since "3" will cover on "foo" in x, then on "2" in y
```

Index