Package 'rasterly'

October 28, 2019

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

Version 0.1.0

Description 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

Suggests grid, stats, compiler, loon, purrr, covr, testthat, knitr, rmarkdown, lubridate

LazyData true

RoxygenNote 6.1.1

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 2019-10-28 15:50:02 UTC

R topics documented:

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

Description

rasterly makes it easy to rapidly generate rasters 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.

A raster may be described as a matrix of cells or pixels arranged in grid-like fashion, in which each pixel represents a value in the source data.

When combined with the plotly package and Plotly.js, rasterly enables analysts to generate interactive figures with very large datasets which are responsive enough to embed into Dash for R applications.

The rasterly function creates a rasterly object, to which aggregation layers may be added. This function is the first step in the process of generating raster image data using the package. The 'rasterly' function is not intended to be used in isolation, since aggregation layers are required for full functionality.

Usage

```
rasterly(data = NULL, mapping = aes(), ..., plot_width = 600,
  plot_height = 600, x_range = NULL, y_range = NULL,
  background = "white", color_map = c("lightblue", "darkblue"),
  color_key = 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_map`	Vector of type character. 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. Use of this option 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

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

https://github.com/plotly/rasterly

- https://github.com/plotly/dashR
- Please report bugs at https://github.com/plotly/rasterly/issues

Examples

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

ridesDf %>%
  rasterly(mapping = aes(x = Lat, y = Lon)) %>%
  rasterize_points() -> p
p

## End(Not run)
```

---

add_rasterly_heatmap *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)
```

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.

Examples

```
## Not run:
library(rasterly)
if(requireNamespace("plotly") && requireNamespace("data.table")) {
 # 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()

 #### 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"
     )
```

```
  )
}

## End(Not run)
```

`fire_map` *Supplemental color maps for rasterly*

Description

Hex codes for the "fire" color map. Used in setting argument ‘color_map‘ or ‘color_key‘ in `rasterly` or `rasterly` layers. This may be deprecated in the future.

Usage

```
fire_map
```

Format

An object of class `character` of length 256.

`hourColors_map` *Supplemental color maps for rasterly*

Description

Hex codes for the "hourColors" color map. Used in setting argument ‘color_map‘ or ‘color_key‘ in `rasterly` or `rasterly` layers. This may be deprecated in the future.

Usage

```
hourColors_map
```

Format

An object of class `character` of length 24.

is.rasterly *Is* rasterly

Description

Reports whether x is a rasterly object.

Usage

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

Arguments

x	a rasterly object

plotly.rasterly *'rasterly' to 'plotly'*

Description

Display raster image via 'plotly'

Usage

```
plotly.rasterly(rastObj, as_heatmap = FALSE, scaling = NULL,
  sizing = c("stretch", "fill", "contain"), ...)
```

Arguments

rastObj	A rasterly object
as_heatmap	Draw 'plotly' by adding heatmap layer. See add_rasterly_heatmap
scaling	It could be an artificial function or a scaling way ("log", "origin")
sizing	Specifies which dimension of the image to constrain. One of "stretch" "fill", "contain"
...	Arguments to the layout object. For documentation, see https://plot.ly/r/reference/#Layout_and_layout_st

Examples

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

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

     max_x <- max(ridesDf$Lat)
     min_x <- min(ridesDf$Lat)
     max_y <- max(ridesDf$Lon)
     min_y <- min(ridesDf$Lon)
     ridesDf %>%
       rasterly(background = "black") %>%
       rasterize_points(xlim = c(min_x, (min_x+max_x)/2),
                        ylim = c(min_y, max_y),
                        mapping = aes(x = Lat, y = Lon),
                        color_map = fire_map) %>%
       rasterize_points(xlim = c((min_x+max_x)/2, max_x),
         ylim = c(min_y, max_y),
         mapping = aes(x = Lat, y = Lon, color = hour),
         color_key = hourColours) %>%
    rasterize_build() %>%
    plotly.rasterly(title = "New York Uber Rides")
   }

## End(Not run)
```

---

rasterize_points *rasterize_points*

Description

Points layer for "rasterly".

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.
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)) %>%
       rasterize_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)) %>%
       rasterize_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)
```

---

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, rasterize_points, [.rasterly, [<-.rasterly

Examples

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

viridis_map *Supplemental color maps for rasterly*

Description

Hex codes for the "viridis" color map. Used in setting argument ‘color_map‘ or ‘color_key‘ in `rasterly` or `rasterly` layers. This may be deprecated in the future.

Usage

```
viridis_map
```

Format

An object of class `character` of length 256.

[.rasterly *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]
```

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.

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_map", "color_key", "alpha", "span", "show_raster", "layout"

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)
) %>%
  rasterize_points(xlim = c(1, 5000)) %>%
  rasterize_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"]
```

`[<-.rasterly` *Extract or replace parts of a 'rasterly' object*

Description

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

Usage

```
## 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. See `help('[<-')`
`value`	values to replace; typically an array-like R object of a similar class as x.

Details

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)
) %>%
  rasterize_points(xlim = c(1, 5000)) %>%
  rasterize_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"]
```

%<-% *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