Package 'threejs'

August 29, 2016

Type Package

Title Interactive 3D Scatter Plots, Networks and Globes

Description Create interactive 3D scatter plots, network plots, and globes using the 'three.js' visualization library (``http://threejs.org'').

Version 0.2.2

Date 2016-03-30

URL http://bwlewis.github.io/rthreejs

BugReports https://github.com/bwlewis/rthreejs/issues

License MIT + file LICENSE

Depends R (>= 3.0.0)

Imports htmlwidgets (>= 0.3.2), base64enc, Matrix, stats, methods, jsonlite

Suggests htmltools (>= 0.2.6), maps, igraph

Enhances knitr, shiny

RoxygenNote 5.0.0

NeedsCompilation no

Author B. W. Lewis [aut, cre],
Three.js authors [cph] (three.js JavaScript library),
Alexey Stukalov [ctb],
Yihui Xie [ctb],
Andreas Briese [ctb],
David Piegza [cph]

Maintainer B. W. Lewis <blewis@illposed.net>

Repository CRAN

Date/Publication 2016-04-01 10:41:46

R topics documented:

`threejs-package` *Interactive 3D graphics including point clouds and globes using three.js and htmlwidgets.*

Description

Interactive 3D graphics including point clouds and globes using three.js and htmlwidgets.

References

http://threejs.org http://www.html5rocks.com/en/tutorials/three/intro/

Examples

```
## Not run:
library("shiny")
runApp(system.file("examples/globe",package="threejs"))
runApp(system.file("examples/scatterplot",package="threejs"))

# See also help for globe.js and scatterplot3.js

## End(Not run)
```

flights *Global flight example data from Callum Prentice.*

Description

Global flight example data from Callum Prentice.

Usage

```
data(flights)
```

Format

A data frame with 34,296 observations of 4 variables, origin_lat, origin_long, dest_lat, and dest_long.

Source

See Callum Prentice https://raw.githubusercontent.com/callumprentice/callumprentice.github.io/master/apps/flight_stream/js/flights_one.js

globejs *Plot Data on 3D Globes*

Description

Plot points, arcs and images on a globe in 3D using Three.js. The globe can be rotated and and zoomed.

Usage

```
globejs(img = system.file("images/world.jpg", package = "threejs"), lat, long,
  value = 40, color = "#00ffff", arcs, arcsColor = "#99aaff",
  arcsHeight = 0.4, arcsLwd = 1, arcsOpacity = 0.2, atmosphere = FALSE,
  bg = "black", height = NULL, width = NULL, ...)
```

Arguments

`img`	A character string representing a file path or URI of an image to plot on the globe surface.
`lat`	Optional data point decimal latitudes, must be of same length as `long` (negative values indicate south, positive north).
`long`	Optional data point decimal longitudes, must be of same length as `lat` (negative values indicate west, positive east).
`value`	Either a single value indicating the height of all data points, or a vector of values of the same length as `lat` indicating height of each point.

color	Either a single color value indicating the color of all data points, or a vector of values of the same length as lat indicating color of each point.
arcs	Optional four-column data frame specifying arcs to plot. The columns of the data frame, in order, must indicate the starting latitude, starting longitude, ending latitude, and ending longitude.
arcsColor	Either a single color value indicating the color of all arcs, or a vector of values of the same length as the number of rows of arcs.
arcsHeight	A single value between 0 and 1 controlling the height above the globe of each arc.
arcsLwd	Either a single value indicating the line width of all arcs, or a vector of values of the same length as the number of rows of arcs.
arcsOpacity	A single value between 0 and 1 indicating the opacity of all arcs.
atmosphere	TRUE enables WebGL atmpsphere effect.
bg	Plot background color.
height	The container div height.
width	The container div width.
...	Additional arguments to pass to the three.js renderer (see below for more information on these options).

Value

An htmlwidget object (displayed using the object's show or print method).

Available rendering options

- "bodycolor" The diffuse reflective color of the globe.
- "emissive" The emissive color of the globe object.
- "lightcolor" The color of the ambient light in the scene.
- "fov" The initial field of view, default is 35.
- "rotationlat" The initial globe latitudinal rotation in radians, default is 0.
- "rotationlong" The initial globe longitudinal rotation in radians, default is 0.
- "pointsize" The numeric size of the points/bars, default is 1.
- "renderer" Manually set the three.js renderer to one of 'auto' or 'canvas'. The canvas renderer works across a greater variety of viewers and browsers. The default setting of 'auto' automatically chooses WebGL rendering if it's available.

Specify colors with standard color names or hex color representations. The default values (well-suited to many earth-like map images) are `lightcolor = "#aaeeff"`, `emissive = "#0000ff"`, and `bodycolor = "#0000ff"`. Larger `fov` values result in a smaller (zoomed out) globe. The latitude and longitude rotation values are relative to the center of the map image. Their default values of zero radians result in the front of the globe corresponding to the center of the flat map image.

Note

The `img` argument specifies the WebGL texture image to wrap on a sphere. If you plan to plot points using `lat` and `lon` the image must be a plate carree (aka lat/long) equirectangular map projection; see https://en.wikipedia.org/wiki/Equirectangular_projection for details. Lat/long maps are commonly found for most planetary bodies in the solar system, and are also easily generated directly in R (see the references and examples below).

References

The three.js project http://threejs.org. (The corresponding three.js javascript file is in `system.file("htmlwidgets/gl`

An excellent overview of available map coordinate reference systems (PDF): https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/OverviewCoordinateReferenceSystems.pdf

Includes images adapted from the NASA Earth Observatory and NASA Jet Propulsion Laboratory. World image link: http://goo.gl/GVjxJ.

Examples

```
# Plot flights to frequent destinations from Callum Prentice's
# global flight data set,
# http://callumprentice.github.io/apps/flight_stream/index.html
data(flights)
# Approximate locations as factors
dest   <- factor(sprintf("%.2f:%.2f",flights[,3], flights[,4]))
# A table of destination frequencies
freq <- sort(table(dest), decreasing=TRUE)
# The most frequent destinations in these data, possibly hub airports?
frequent_destinations <- names(freq)[1:10]
# Subset the flight data by destination frequency
idx <- dest %in% frequent_destinations
frequent_flights <- flights[idx, ]
# Lat/long and counts of frequent flights
ll <- unique(frequent_flights[,3:4])
# Plot frequent destinations as bars, and the flights to and from
# them as arcs. Adjust arc width and color by frequency.
globejs(lat=ll[,1], long=ll[,2], arcs=frequent_flights,
        arcsHeight=0.3, arcsLwd=2, arcsColor="#ffff00", arcsOpacity=0.15,
        atmosphere=TRUE, color="#00aaff", pointsize=0.5)

# Plot populous world cities from the maps package.
library(threejs)
library(maps)
data(world.cities, package="maps")
cities <- world.cities[order(world.cities$pop, decreasing=TRUE)[1:1000],]
value  <- 100 * cities$pop / max(cities$pop)
col <- colorRampPalette(c("cyan", "lightgreen"))(10)[floor(10 * value/100) + 1]
globejs(lat=cities$lat, long=cities$long, value=value, color=col, atmosphere=TRUE)

# Plot the data on the moon:
moon <- system.file("images/moon.jpg", package="threejs")
globejs(img=moon, bodycolor="#555555", emissive="#444444",
         lightcolor="#555555", lat=cities$lat, long=cities$long,
```

```
         value=value, color=col)

## Not run:
# Plot a high-resolution NASA MODIS globe, setting colors to more closely reproduce
# the natural image colors. Note that this example can can take a while to download!
globejs("http://goo.gl/GVjxJ",
        emmisive="#000000", bodycolor="#000000", lightcolor="#aaaa44")

# Using global plots from the maptools, rworldmap, or sp packages.

# Instead of using ready-made images of the earth, we can use
# many R spatial imaging packages to produce globe images
# dynamically. With a little extra effort you can build globes with total
# control over how they are plotted.

library(maptools)
library(threejs)
data(wrld_simpl)

bgcolor <- "#000025"
earth <- tempfile(fileext=".jpg")

# NOTE: Use antialiasing to smooth border boundary lines. But! Set the jpeg
# background color to the globe background color to avoid a visible aliasing
# effect at the the plot edges.

jpeg(earth, width=2048, height=1024, quality=100, bg=bgcolor, antialias="default")
par(mar = c(0,0,0,0), pin = c(4,2), pty = "m",  xaxs = "i",
    xaxt = "n",       xpd = FALSE,  yaxs = "i", bty = "n", yaxt = "n")
plot(wrld_simpl, col="black", bg=bgcolor, border="cyan", ann=FALSE,
     setParUsrBB=TRUE)
dev.off()
globejs(earth)

# A shiny example:
shiny::runApp(system.file("examples/globe",package="threejs"))

## End(Not run)

# See http://bwlewis.github.io/rthreejs for additional examples.
```

`globeOutput` *Shiny bindings for threejs widgets*

Description

Output and render functions for using threejs widgets within Shiny applications and interactive Rmd documents.

Usage

```
globeOutput(outputId, width = "100%", height = "600px")

renderGlobe(expr, env = parent.frame(), quoted = FALSE)

graphOutput(outputId, width = "100%", height = "500px")

renderGraph(expr, env = parent.frame(), quoted = FALSE)

scatterplotThreeOutput(outputId, width = "100%", height = "500px")

renderScatterplotThree(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like "100%", "400px", "auto") or a number, which will be coerced to a string and have "px" appended.
expr	An expression that generates threejs graphics.
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

---

graph2Matrix *Convert from node and edge graph representation to a sparse adjacency matrix representation*

Description

Convert from node and edge graph representation to a sparse adjacency matrix representation

Usage

```
graph2Matrix(edges, nodes, symmetric = TRUE)
```

Arguments

edges	A data frame with at least the columns "from" and "to" referring to edges between ids in the nodes data frame. If the data frame includes a numeric "size" variable then graph is assumed to be weighted and the corresponding matrix entries are set to the size values.
nodes	Optional data frame with at least a column named "id" corresponding to the from and to node ids in the edges argument. The size of the matrix is determined by number of rows in the data frame. If nodes is missing it will be inferred from the edges. If nodes has a "label" column, the matrix row and column names will be set to the corresponding node labels.
symmetric	Set to FALSE for directed graphs, or leave as TRUE for undirected graphs.

Value

A sparse matrix

See Also

graphjs, graph2Matrix

Examples

```
data(LeMis)
M <- graph2Matrix(LeMis$edges, LeMis$nodes)
G <- matrix2graph(M)
```

---

graphjs *Interactive 3D Force-directed Graphs*

Description

Plot interactive force-directed graphs.

Usage

```
graphjs(edges, nodes, main = "", curvature = 0, bg = "white",
  fg = "black", showLabels = FALSE, attraction = 1, repulsion = 1,
  max_iterations = 1500, opacity = 1, stroke = TRUE, width = NULL,
  height = NULL)
```

Arguments

edges	Either a list with edges and nodes data frames as described below, or a graph object produced from the igraph package (see igraph2graphjs), or an edge data frame with at least columns: • from Integer node id identifying edge 'from' node • to Integer node id identifying the edge 'to' node • size Nonnegative numeric edge line width • color Edge color specified like node color above Each row of the data frame identifies a graph edge.
nodes	Optional node (vertex) data frame with at least columns: • label Node character labels • id Unique integer node ids (corresponding to node ids used by edges) • size Positive numeric node plot size • color A character color value, either color names ("blue", "red", ...) or 3-digit hexadecimal values ("#0000FF", "#EE0011") Each row of the data frame defines a graph node. If the nodes argument is missing it will be inferred from the edges argument.

main	Plot title
curvature	Zero implies that edges are straight lines. Specify a positive number to curve the edges, useful to distinguish multiple edges in directed graphs (the z-axis of the curve depends on the sign of edge$from - edge$to). Larger numbers = more curvature, with 1 a usually reasonable value.
bg	Plot background color specified similarly to the node colors described above
fg	Plot foreground text color
showLabels	If TRUE then display text labels near each node
attraction	Numeric value specifying attraction of connected nodes to each other, larger values indicate more attraction
repulsion	Numeric value specifying repulsion of all nodes to each other, larger values indicate greater repulsion
max_iterations	Integer value specifying the maximum number of rendering iterations before stopping
opacity	Node transparency, 0 <= opacity <= 1
stroke	If TRUE, stroke each node with a black circle
width	optional widget width
height	optional widget height

Value

An htmlwidget object that is displayed using the object's show or print method. (If you don't see your widget plot, try printing it with the print) function.

Note

All colors must be specified as color names like "red", "blue", etc. or as hexadecimal color values without opacity channel, for example "#FF0000", "#0a3e55" (upper or lower case hex digits are allowed).

The plot responds to the following mouse controls (touch analogs may also be supported on some systems):

- scrollwheel zoom
- left-mouse button + move rotate
- right-mouse button + move pan
- mouse over identify node by appending its label to the title

Double-click or tap on the plot to reset the view.

Basic support for plotting igraph objects is provided by the igraph2graphjs function.

References

Original code by David Piegza: https://github.com/davidpiegza/Graph-Visualization.

The three.js project http://threejs.org.

See Also

LeMis

Examples

```
data(LeMis)
g <- graphjs(LeMis, main="Les Mis&eacute;rables", showLabels=TRUE)
print(g)

## Not run:
# The next example uses the `igraph` package.
library(igraph)
set.seed(1)
g <- sample_islands(3, 10, 5/10, 1)
i <- cluster_optimal(g)
g <- set_vertex_attr(g, "color", value=c("yellow", "green", "blue")[i$membership])
print(graphjs(g))

## End(Not run)
```

---

igraph2graphjs *Convert* igraph *graph objects to a simpler form used by* graphjs

---

Description

Convert igraph graph objects to a simpler form used by graphjs

Usage

```
igraph2graphjs(ig)
```

Arguments

ig	A graph object from igraph

Value

A list with node and edges data frame entries used by graphjs.

Examples

```
## Not run:
  library(igraph)
  g <- make_ring(10) %>%
       set_edge_attr("weight", value = 1:10) %>%
       set_edge_attr("color", value = "red") %>%
       set_vertex_attr("name", value = letters[1:10])
  (G <- igraph2graphjs(g))
```

```
  # Can also directly run:
  graphjs(g)

## End(Not run)
```

`LeMis` *Les Miserables Character Coappearance Data*

Description

Les Miserables Character Coappearance Data

Usage

```
data(LeMis)
```

Format

A list of two data frames: nodes with 77 observations (listing characters in the novel) of 4 variables (`id` a numeric identifier, `label` the character's name, `size` a node plot size, and `color` the node plot color); edges: with 254 observations of 4 variables (`from` and `to` listing connections between node ids, `size` the edge plot size, and `color` the edge plot color.

Source

Mike Bostock's D3.js force-directed graph example http://bl.ocks.org/mbostock/4062045. Data based on character coappearence in Victor Hugo's Les Miserables, compiled by Donald Knuth (http://www-cs-faculty.stanford.edu/~uno/sgb.html).

`matrix2graph` *Convert a matrix or column-sparse matrix to a list of edges and nodes for use by* `graphjs`.

Description

Convert a matrix or column-sparse matrix to a list of edges and nodes for use by `graphjs`.

Usage

```
matrix2graph(M)
```

Arguments

`M`	either a matrix or any of the possible column sparse matrix objects from the Matrix package.

Value

A list with node and edges data frame entries used by graphjs.

Note

Numeric graphs are assumed to be weighted and the edge "size" values are set to the corresponding matrix entries.

See Also

graphjs, graph2Matrix

Examples

```
data(LeMis)
M <- graph2Matrix(LeMis$edges, LeMis$nodes)
G <- matrix2graph(M)
```

---

scatterplot3js *Interactive 3D Scatterplots*

Description

A 3D scatterplot widget using three.js. Many options follow the scatterplot3d function from the eponymous package.

Usage

```
scatterplot3js(x, y, z, height = NULL, width = NULL, axis = TRUE,
  num.ticks = c(6, 6, 6), x.ticklabs = NULL, y.ticklabs = NULL,
  z.ticklabs = NULL, color = "steelblue", size = 1, labels = NULL,
  label.margin = "10px", stroke = "black", flip.y = TRUE, grid = TRUE,
  renderer = c("auto", "canvas", "webgl"), signif = 8, bg = "#ffffff",
  xlim, ylim, zlim, pch, ...)
```

Arguments

x	Either a vector of x-coordinate values or a three-column data matrix with columns corresponding to the x,y,z coordinate axes. Column labels, if present, are used as axis labels.
y	(Optional) vector of y-coordinate values, not required if x is a matrix.
z	(Optional) vector of z-coordinate values, not required if x is a matrix.
height	The container div height.
width	The container div width.
axis	A logical value that when TRUE indicates that the axes will be displayed.

num.ticks	A three-element vector with the suggested number of ticks to display per axis. Set to NULL to not display ticks. The number of ticks may be adjusted by the program.
x.ticklabs	A vector of tick labels of length num.ticks[1], or NULL to show numeric labels.
y.ticklabs	A vector of tick labels of length num.ticks[2], or NULL to show numeric labels.
z.ticklabs	A vector of tick labels of length num.ticks[3], or NULL to show numeric labels.
color	Either a single hex or named color name (all points same color), or a vector of #' hex or named color names as long as the number of data points to plot.
size	The plot point radius, either as a single number or a vector of sizes of length nrow(x). A vector of sizes is only supported by the canvas renderer. The webgl renderer accepts a single size value for all points.
labels	Either NULL (no labels), or a vector of labels as long as the number of data points displayed when the mouse hovers over each point.
label.margin	A CSS-style margin string used to display the point labels.
stroke	A single color stroke value (surrounding each point). Set to null to omit stroke (only available in the canvas renderer).
flip.y	Reverse the direction of the y-axis (the default value of TRUE produces plots similar to those rendered by the R scatterplot3d package).
grid	Set FALSE to disable display of a grid.
renderer	Select from available plot rendering techniques of 'auto', 'canvas', or 'webgl'.
signif	Number of significant digits used to represent point coordinates. Larger numbers increase accuracy but slow plot generation down.
bg	The color to be used for the background of the device region.
xlim	Optional two-element vector of x-axis limits. Default auto-scales to data.
ylim	Optional two-element vector of y-axis limits. Default auto-scales to data.
zlim	Optional two-element vector of z-axis limits. Default auto-scales to data.
pch	Not yet used but one day will support changing the point glyph.
...	Additional options (see note).

Value

An htmlwidget object that is displayed using the object's show or print method. (If you don't see your widget plot, try printing it with the print) function. The returned object includes a special points3d function for adding points to the plot similar to scatterplot3d. See the note below and examples for details.

Note

Points with missing values are omitted.

Use the renderer option to manually select from the available rendering options. The canvas renderer is the fallback rendering option when webgl is not available. Select auto to automatically choose between the two. The two renderers produce slightly different-looking output and have different available options (see above). Use the webgl renderer for plotting large numbers of points

(if available). Use the `canvas` renderer to excercise finer control of plotting of smaller numbers of points. See the examples.

Use the optional `...` argument to explicitly supply `axisLabels` as a three-element character vector, see the examples below.

The returned object includes a `points3d` function that can add points to a plot, returning a new htmlwidget plot object. The function signature is a subset of the full `scatterplot3js` function:

```
points3d (x, y, z, color="steelblue", size=1, labels="")
```

It allows you to add points to a plot using the same syntax as `scatterplot3js` with optionally specified color, size, and labels. New points are plotted in the same scale as the existing plot. See the examples section below for an example.

References

The three.js project http://threejs.org.

See Also

scatterplot3d, rgl

Examples

```
# Gumball machine
N <- 100
i <- sample(3, N, replace=TRUE)
x <- matrix(rnorm(N*3),ncol=3)
lab <- c("small", "bigger", "biggest")
scatterplot3js(x, color=rainbow(N), labels=lab[i], size=i, renderer="canvas")

# Example 1 from the scatterplot3d package (cf.)
z <- seq(-10, 10, 0.1)
x <- cos(z)
y <- sin(z)
scatterplot3js(x,y,z, color=rainbow(length(z)),
      labels=sprintf("x=%.2f, y=%.2f, z=%.2f", x, y, z))

# Same example with explicit axis labels
scatterplot3js(x,y,z, color=rainbow(length(z)), axisLabels=c("a","b","c"))

# Pretty point cloud example, should run this with WebGL!
N     <- 20000
theta <- runif(N)*2*pi
phi   <- runif(N)*2*pi
R     <- 1.5
r     <- 1.0
x <- (R + r*cos(theta))*cos(phi)
y <- (R + r*cos(theta))*sin(phi)
z <- r*sin(theta)
d <- 6
h <- 6
t <- 2*runif(N) - 1
```

```
w <- t^2*sqrt(1-t^2)
x1 <- d*cos(theta)*sin(phi)*w
y1 <- d*sin(theta)*sin(phi)*w
i <- order(phi)
j <- order(t)
col <- c( rainbow(length(phi))[order(i)],
         rainbow(length(t),start=0, end=2/6)[order(j)])
M <- cbind(x=c(x,x1),y=c(y,y1),z=c(z,h*t))
scatterplot3js(M,size=0.25,color=col,bg="black")

# Adding points to a plot with points3d
set.seed(1)
lim <- c(-3,3)
x <- scatterplot3js(rnorm(5),rnorm(5),rnorm(5), xlim=lim, ylim=lim, zlim=lim)
a <- x$points3d(rnorm(3),rnorm(3),rnorm(3)/2, color="red", labels="NEW")
## Not run:
  # A shiny example
  shiny::runApp(system.file("examples/scatterplot",package="threejs"))

## End(Not run)
```

`texture` *Convert an image file or uri to a three.js texture*

Description

Convert file image representations in R into JSON-formatted arrays suitable for use as three.js textures. This function is automatically invoked for images used in the `globejs` function.

Usage

```
texture(data)
```

Arguments

`data`	A character string file name referring to an image file, or referring to an image uri (see the examples).

Value

JSON-formatted list with image, width, and height fields suitable for use as a three.js texture created with the base64texture function. The image field contains a base64 dataURI encoding of the image.

Note

Due to browser "same origin policy" security restrictions, loading textures from a file system in three.js may lead to a security exception, see https://github.com/mrdoob/three.js/wiki/How-to-run-things-locally. References to file locations work in Shiny apps, but not in standalone examples. The `texture` function facilitates transfer of image texture data from R into three.js textures. Binary image data are encoded and inserted into three.js without using files as dataURIs.

References

The threejs project http://threejs.org. https://github.com/mrdoob/three.js/wiki/How-to-run-things-locally

Examples

```
## Not run:
# A big image (may take a while to download):
img <- paste("http://eoimages.gsfc.nasa.gov/",
             "images/imagerecords/73000/73909/",
             "world.topo.bathy.200412.3x5400x2700.jpg", sep="")
t <- texture(img)

## End(Not run)
```

Index