Package 'threejs'

April 21, 2025

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 (<https://threejs.org>).

Version 0.3.4

Date 2025-04-19

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

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

License MIT + file LICENSE

Depends R (>= 3.0.0), igraph (>= 1.0.0)

Imports htmlwidgets (>= 0.3.2), base64enc, crosstalk, methods, stats

Suggests maps

Enhances knitr, shiny

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2025-04-21 14:10:02 UTC

Contents

`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.

Author(s)

Maintainer: B. W. Lewis <blewis@illposed.net> [copyright holder]

Other contributors:

- Three.js authors (three.js library) [copyright holder]
- jQuery Foundation (jQuery library) [copyright holder]
- Alexey Stukalov <astukalov@gmail.com> [contributor]
- Yihui Xie <xie@yihui.name> [contributor]
- Andreas Briese <ab@edutoolbox.de> [contributor]
- B. Thieurmel <bthieurmel@gmail.com> [contributor]

References

https://threejs.org

See Also

Useful links:

- https://bwlewis.github.io/rthreejs/
- Report bugs at https://github.com/bwlewis/rthreejs/issues

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)
```

---

ego *Facebook social circles*

Description

A facebook social network subgraph obtained from the Stanford SNAP repository.

Usage

```
data(ego)
```

Format

An igraph package undirected graph object with 4039 vertices and 88234 edges. The graph includes a force-directed layout with vertices colored by the `cluster_fast_greedy` algorithm from the igraph package.

Source

Stanford SNAP network repository https://snap.stanford.edu/data/facebook_combined.txt.gz

References

J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

flights *Global flight example data from Callum Prentice.*

Description

Global flight example data from Callum Prentice. Data are dynamically downloaded from GitHub.

Usage

```
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

gcol *A basic internal color format parser*

Description

A basic internal color format parser

Usage

```
gcol(x)
```

Arguments

x	a character-valued color name

Value

a list of 3-hex-digit color values and scalar numeric alpha values

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,
  elementId = 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.
elementId	Use an explicit element ID for the widget (rather than an automatically generated one). Useful if you have other JavaScript that needs to explicitly discover and interact with a specific widget instance.
...	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.

"program" User-supplied JavaScript run on plot initialization

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 = "#000000"`, and `bodycolor = "#ffffff"`. 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 https://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/sites/default/files/2020-04/OverviewCoordinateReferenceSystems.pdf.

Examples

```
## Not run:
# Plot flights to frequent destinations from Callum Prentice's
# global flight data,
# http://callumprentice.github.io/apps/flight_stream/index.html
f <- flights()
# Approximate locations as factors
dest   <- factor(sprintf("%.2f:%.2f", f[,3], f[,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 <- f[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,
        bodycolor="#aaaaff", arcsHeight=0.3, arcsLwd=2,
        arcsColor="#ffff00", arcsOpacity=0.15,
        atmosphere=TRUE, color="#00aaff", pointsize=0.5)

## End(Not run)

## Not run:
# 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", lightcolor="#aaaaaa",
        lat=cities$lat, long=cities$long,
        value=value, color=col)

# 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)

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.

---

graphjs *Interactive 3D Graph Visualization*

Description

Make interactive 3D plots of igraph objects.

Usage

```
graphjs(
  g,
  layout,
  vertex.color,
  vertex.size,
  vertex.shape,
  vertex.label,
  edge.color,
  edge.width,
  edge.alpha,
  main = "",
  bg = "white",
  width = NULL,
  height = NULL,
  elementId = NULL,
  ...
)
```

Arguments

g	an igraph graph object or a list of igraph objects (see notes)
layout	optional graph layout or list of layouts (see notes)
vertex.color	optional vertex color or vector of colors as long as the number of vertices in g
vertex.size	optional vertex size or vector of sizes
vertex.shape	optional vertex shape or vector of shapes
vertex.label	optional mouse-over vertex label or vector of labels

edge.color	optional edge color or vector of colors as long as the number of edges in g
edge.width	optional edge width (single scalar value, see notes)
edge.alpha	optional single numeric edge transparency value
main	plot title text
bg	plot background color
width	the widget container div width in pixels
height	the widget container div height in pixels
elementId	Use an explicit element ID for the widget (rather than an automatically generated one). Useful if you have other JavaScript that needs to explicitly discover and interact with a specific widget instance.
...	optional additional arguments passed to scatterplot3js

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.)

Interacting with the plot

Press and hold the left mouse button, or touch or trackpad equivalent, and move the mouse to rotate the plot. Press and hold the right mouse button to pan. Use the mouse scroll wheel to zoom. If vertex.labels are specified (see below), moving the mouse pointer over a point will display the label. Altenatively use vertex.shape to plot character names as shown in the examples below. Set the optional experimental use.orbitcontrols=TRUE argument to use a more CPU-efficient but somewhat less fluid mouse/touch interface.

Layout options

Use the layout parameter to control the visualization layout by supplying either a three-column matrix of vertex x, y, z coordinates, or a function that returns such a layout. The igraph layout_with_fr force-directed layout is used by default (note that only 3D layouts are supported). Also see the animation section below.

Vertex options

Optional parameters beginning with vertex. represent a subset of the igraph package vertex visualization options and work similarly, see igraph.plotting. Vertex shapes in graphjs act somewhat differently, and are mapped to the pch option in scatterplot3js. In particular, pch character symbols or even short text strings may be specified. The vertex.label option enables a mouse-over label display instead of plotting lables directly near the vertices. (Consider using the text pch options for that instead.)

Edge options

Optional parameters beginning with edge. represent a subset of the igraph edge visualization options and work similarly as the vertex. options above. The current version of the package only supports uniform edge widths specified by a single scalar value. This choice was made for performance reasons to support large visualizations.

Graph animation

Specifying a list of three-column layout matrices in `layout` displays a linear interpolation from one layout to the next, providing a simple mechanism for graph animation. Each layout must have the same number of rows as the number of vertices in the graph.

Specify the optional `fpl` (frames per layout) parameter to control the number of interpolating animation frames between layouts. See the examples.

Optionally specify a list of graph objects in `g` to vary the displayed edges and edge colors from one layout to the next, with the restriction that each graph object must refer to a uniform number of vertices.

The lists of graphs may optionally include varying vertex and edge colors. Alternatively, specify a list of `vertex.color` vectors (one for each layout) to animate vertex colors. Similarly, optionally specify a list of `edge.color` vectors to animate edge colors.

Optionally provide a list of `main` title text strings to vary the title with each animation layout.

None of the other plot parameters may be animated.

Click animation

Specify the option `click=list` to animate the graph when specified vertices are clicked interactively, where `list` is a named list of animation entries. Each entry must itself be a list with the following entries

g optional a single igraph object with the same number of vertices as `g` above (if specified this must be the first entry)

layout - optional a single igraph layout, or differential layout if `cumulative=TRUE`

vertex.color - optional single vector of vertex colors

edge.color - optional single vector of edge colors

cumulative - optional boolean entry, if `TRUE` then vertex positions are added to current plot, default is `FALSE`

At least one of `g` or `layout` must be specified in each animation list entry. The layouts and colors may be alternatively imbedded in the igraph object itself. Each animation list entry must be named by a number corresponding to the vertex enumeration in `g`. An animation sequence is triggered when a corresponding vertex is clicked. For instance, to trigger animations when vertices number 1 or 5 are clicked, include list entries labeled `"1"` and `"5"`. See the demos in `demo(package="threejs")` for detailed examples.

Other interactions

Specify the argument `brush=TRUE` to highlight a clicked vertex and its directly connected edges (click off of a vertex to reset the display). Optionally set the `highlight=<hex color>` and `lowlight=<hex color>` to manually control the brushing display colors.

Crosstalk

`graphjs()` works with crosstalk selection (but not filtering yet); see https://rstudio.github.io/crosstalk/. Enable crosstalk by supplying the optional agrument `crosstalk=df`, where `df` is a crosstalk-SharedData data.frame-like object with the same number of rows as graph vertices (see the examples).

User-defined JavaScript

Use the optional `program` argument (see `scatterplot3js`) to supply JavaScript code as a character string value. The code will be run during plot initialization. See the examples.

Note

Edge transparency values specified as part of `edge.color` are ignored, however you can set an overall transparency for edges with `edge.alpha`.

References

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

See Also

`igraph.plotting`, `scatterplot3js`

Examples

```
set.seed(1)
g <- sample_islands(3, 10, 5/10, 1)
i <- membership(cluster_louvain(g))
(graphjs(g, vertex.color=c("orange", "green", "blue")[i],
         vertex.shape="sphere"))

# similar example with user-defined directional lighting
l1 <- light_directional(color="red", position=c(0, -0.8, 0.5))
l2 <- light_directional(color="yellow", position=c(0, 0.8, -0.5))
l3 <- light_ambient(color="#555555")
(graphjs(g, vertex.color="gray", vertex.shape="sphere",
         lights=list(l1, l2, l3)))

# Les Miserables Character Co-appearance Data
data("LeMis")
(graphjs(LeMis))

# Use HTML and CSS directly in each vertex label to customize
# and align the legend:
(graphjs(LeMis,
         vertex.label=sprintf("<h2 style='text-align:left;'>%s</h2>",
           V(LeMis)$label)))

# The plot legend 'div' element is of CSS class 'infobox'. Use JavaScript
# to customize labels to hover near the mouse pointer:
program <- "document.addEventListener('mousemove', function(e) {
  e.preventDefault();
  let x = document.getElementsByClassName('infobox')[0];
  x.style['background'] = '#00c9c2';
  x.style['border-radius'] = '5px';
  x.style['color'] = '#222';
  x.style['font-family'] = 'sans-serif';
  x.style['position'] = 'absolute';
```

```
  x.style['top'] = e.pageY + 'px';
  x.style['left'] = e.pageX + 'px';
})"
(graphjs(LeMis, program = program))

# ...plot Character names
(graphjs(LeMis, vertex.shape=V(LeMis)$label, vertex.size=0.3))

# SNAP Facebook ego network dataset
data("ego")
(graphjs(ego, bg="black"))

## Not run:
# A shiny example
shiny::runApp(system.file("examples/graph", package="threejs"))

# A graph amination that shows several layouts
data("LeMis")
graphjs(LeMis,
  layout=list(
    layout_randomly(LeMis, dim=3),
    layout_on_sphere(LeMis),
    layout_with_drl(LeMis, dim=3),  # note! somewhat slow...
    layout_with_fr(LeMis, dim=3, niter=30)),
  main=list("random layout", "sphere layout", "drl layout", "fr layout"),
  fpl=300)

# A simple graph animation illustrating edge modification
g <- make_ring(5) - edges(1:5)
graph_list <- list(
 g + edge(1, 2),
 g + edge(1, 2) + edge(2, 3),
 g + edge(1, 2) + edge(2, 3) + edge(3, 4),
 g + edge(1, 2) + edge(2, 3) + edge(3, 4) + edge(4, 5),
 g + edge(1, 2) + edge(2, 3) + edge(3, 4) + edge(4, 5) + edge(5, 1))
 graphjs(graph_list, main=paste(1:5),
   vertex.color=rainbow(5), vertex.shape="sphere", edge.width=3)

# see `demo(package="threejs") for more animation demos.

# A crosstalk example
library(crosstalk)
library(DT)
data(LeMis)
sd = SharedData$new(data.frame(Name = V(LeMis)$label))
print(bscols(
  graphjs(LeMis, brush=TRUE, crosstalk=sd),
  datatable(sd, rownames=FALSE, options=list(dom='tp'))
))

## End(Not run)
```

LeMis *Les Miserables Character Coappearance Data*

Description

Les Miserables Character Coappearance Data

Usage

```
data(LeMis)
```

Format

An igraph package graph object.

Source

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

light_ambient *Plot illumination*

Description

Plot illumination

Usage

```
light_ambient(color = "#eeeeee")
```

Arguments

color	the ambient light color

Value

An object for use with the lights argument in scatterplot3js and graphjs.

light_directional *Plot illumination*

Description

Plot illumination

Usage

```
light_directional(color = "#eeeeee", position = c(0, 0, 0))
```

Arguments

color	the light color
position	the light position as an (x, y, z) coordinate vector with entries in [-1, 1]

Value

An object for use with the lights argument in scatterplot3js and graphjs.

lines3d *Add lines to a 3D scatterplot*

Description

Add lines to a 3D scatterplot

Usage

```
lines3d(s, from, to, lwd = 1, alpha = 1, color)
```

Arguments

s	A scatterplot object returned by scatterplot3js.
from	A vector of integer indices of starting points.
to	A vector of integer indices of ending points of the same length as from.
lwd	A single numeric value of line width (applies to all lines).
alpha	A single numeric value of line alpha (applies to all lines).
color	Either a single color value or vector of values as long as from of line colors; line colors default to interpolating their vertex point colors.

Value

A new scatterplot htmlwidget object.

Note

This function replaces the old `points3d` approach used by `scatterplot3d`.

Examples

```
## Not run:
 x <- rnorm(5)
 y <- rnorm(5)
 z <- rnorm(5)
 scatterplot3js(x, y, z, pch="@", color=rainbow(5)) %>%
   lines3d(c(1, 2), c(3, 4), lwd=2)

## End(Not run)
```

`points3d` *Add points to a 3D scatterplot*

Description

Add points to a 3D scatterplot

Usage

```
points3d(s, x, y, z, color = "orange", pch = "@", size = 1, labels = "")
```

Arguments

`s`	A non-animated scatterplot object returned by `scatterplot3js`.
`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.
`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 points in `x`.
`pch`	Optional point glyphs or text strings, see `scatterplot3js`.
`size`	The plot point radius, either as a single number or a vector of sizes of length `nrow(x)`.
`labels`	Character vector of length `x` of point labels displayed when the mouse moves over the points.

Value

A new scatterplot htmlwidget object.

Note

This function replaces the old `points3d` approach used by `scatterplot3d`.

Examples

```
## Not run:
 # Adding point labels to a scatterplot:
 x <- rnorm(5)
 y <- rnorm(5)
 z <- rnorm(5)
 scatterplot3js(x, y, z, pch="o") %>%
   points3d(x + 0.1, y + 0.1, z, color="red", pch=paste("point", 1:5))

# Adding point labels to a graph, obtaining the graph vertex coordinates
# with the `vertices()` function:
data(LeMis)
graphjs(LeMis) %>% points3d(vertices(.), color="red", pch=V(LeMis)$label)


## End(Not run)
```

`scatterplot3js` *Interactive 3D Scatterplots*

Description

A 3D scatterplot widget using three.js. Many options follow the `scatterplot3d` 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 = cex.symbols,
  stroke = "black",
  flip.y = TRUE,
  grid = TRUE,
  renderer = c("auto", "canvas"),
  signif = 8,
```

```
  bg = "#ffffff",
  cex.symbols = 1,
  xlim,
  ylim,
  zlim,
  axis.scale = c(1, 1, 1),
  pch = "@",
  elementId = NULL,
  ...
)
```

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 or one-element vector with the suggested number of ticks to display per axis. If a one-element vector, this number of ticks will be used for the axis with the smallest axis.scale, and the number of ticks on the remaining axes will be increased proportionally to the axis.scale values. 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).
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' or 'canvas'. Set to 'canvas' to explicitly use non-accelerated Canvas rendering, otherwise WebGL is used if available.
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.

cex.symbols	Equivalent to the size parameter.
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.
axis.scale	Three-element vector to scale each axis as displayed on the plot, after first scaling them all to a unit length. Default c(1,1,1) thus results in the axes of equal length. If NA, the displayed axes will be scaled to the ratios determined from c(xlim,ylim,zlim).
pch	Optional point glyphs, see notes.
elementId	Use an explicit element ID for the widget (rather than an automatically generated one). Useful if you have other JavaScript that needs to explicitly discover and interact with a specific widget instance.
...	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.)

Scaling the axes

With the default values, the displayed axes are scaled to equal one-unit length. If you instead need to maintain the relative distances between points in the original data, and the same distance between the tick labels, pass num.ticks=6 (or any other single number) and axis.scale=NA

Interacting with the plot

Press and hold the left mouse button (or touch or trackpad equivalent) and move the mouse to rotate the plot. Press and hold the right mouse button (or touch equivalent) to pan. Use the mouse scroll wheel or touch equivalent to zoom. If labels are specified (see below), moving the mouse pointer over a point will display the label.

Detailed plot options

Use the optional ... argument to explicitly supply axisLabels as a three-element character vector, see the examples below. A few additional plot options are also supported:

"lights" a list of light_ambient and light_directional objects

"cex.lab" font size scale factor for the axis labels

"cex.axis" font size scale factor for the axis tick labels

"font.axis" CSS font string used for all axis labels

"font.symbols" CSS font string used for plot symbols

"font.main" CSS font string used for main title text box

"labels" character vector of length x of point labels displayed when the mouse moves over the points

"main" Plot title text

"top" Top location in pixels from top of the plot title text

"left" Left location in pixels from center of the plot title text

"program" User-supplied JavaScript run on plot initialization

The default CSS font string is "48px Arial". Note that the format of this font string differs from, for instance, the usual 'par(font.axis)'.

Use the `pch` option to specify points styles in WebGL-rendered plots. `pch` may either be a single character value that applies to all points, or a vector of character values of the same length as `x`. All character values are used literally ('+', 'x', '*', etc.) except for the following special cases:

"o" Plotted points appear as 3-d spheres.

"@" Plotted points appear as stroked disks.

"." Points appear as tiny squares.

Character strings of more than one character are supported–see the examples. The `"@"` and `"."` options exhibit the best performance, consider using one of those to plot large numbers of points.

Set the optional experimental `use.orbitcontrols=TRUE` argument to use a more CPU-efficient but somewhat less fluid mouse/touch interface.

Plotting lines

See `lines3d` for an alternative interface. Lines are optionally drawn between points specified in `x, y, z` using the following new plot options.

"from" A numeric vector of indices of line starting vertices corresponding to entries in `x`.

"to" A numeric vector exactly as long as `from` of indices of line ending vertices corresponding to entries in `x`.

"lcol" Either a single color value or vector of values as long as from; line colors default to interpolating their vertex point colors.

"lwd" A single numeric value of line width (for all lines), defaults to 1.

"linealpha" A single numeric value between 0 and 1 inclusive setting the transparency of all plot lines, defaulting to 1.

Highlighting selected points

Specify the argument `brush=TRUE` to highlight a clicked point (currently limited to single-point selection). Optionally set the `highlight=<color>` and `lowlight=<color>` to manually control the brushing display colors. This feature works with crosstalk.

Crosstalk

The `scatterplot3js()` and `graphjs()` functions work with crosstalk selection (but not filtering yet); see https://rstudio.github.io/crosstalk/. Enable crosstalk with the optional agrument `crosstalk=df`, where `df` is a crosstalk-SharedData data.frame-like object with the same number of rows as points (`scatterplot3js()`) or graph vertices (`graphjs()`) (see the examples).

Note

Points with missing values are omitted from the plot, please try to avoid missing values in `x`, `y`, `z`.

References

The three.js project: https://threejs.org. The HTML Widgets project:

See Also

scatterplot3d, rgl, points3d, lines3d, light_ambient, light_directional

Examples

```
# 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)))

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

# Same example showing multiple point styles with pch
scatterplot3js(x, y, z, color=rainbow(length(z)),
               pch=sample(c(".", "o", letters), length(x), replace=TRUE))

# 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=0.5, color=col, bg="black", pch=".")

# Plot generic text using 'pch' (we label some points in this example)
set.seed(1)
x <- rnorm(5); y <- rnorm(5); z <- rnorm(5)
scatterplot3js(x, y, z, pch="@") %>%
```

```
   points3d(x + 0.1, y + 0.1, z, color="red", pch=paste("point", 1:5))

## Not run:
  # A shiny example
  shiny::runApp(system.file("examples/scatterplot", package="threejs"))

## End(Not run)

## Not run:
  # A crosstalk example
  library(crosstalk)
  library(d3scatter) # devtools::install_github("jcheng5/d3scatter")
  z <- seq(-10, 10, 0.1)
  x <- cos(z)
  y <- sin(z)
  sd <- SharedData$new(data.frame(x=x, y=y, z=z))
  print(bscols(
    scatterplot3js(x, y, z, color=rainbow(length(z)), brush=TRUE, crosstalk=sd),
    d3scatter(sd, ~x, ~y, width="100%", height=300)
  ))

## 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 https://threejs.org. https://github.com/mrdoob/three.js/wiki/How-to-run-things-locall

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)
```

`vertices,scatterplotThree-method` *Extract a matrix of vertex coordinates from a threejs widget*

Description

Extract a matrix of vertex coordinates from a threejs widget

Usage

```
## S4 method for signature 'scatterplotThree'
vertices(...)
```

Arguments

`...`	a `scatterplotThree` object from the threejs package.

See Also

points3d

Index