Package ‘r3js’

October 14, 2022

Type Package
Title 'WebGL'-Based 3D Plotting using the 'three.js' Library
Version 0.0.1
Date 2022-06-10
Author Sam Wilks
Maintainer Sam Wilks <sw463@cam.ac.uk>
Description Provides R and 'JavaScript' functions to allow 'WebGL'-based 3D plotting using the 'three.js' 'JavaScript' library. Interactivity through roll-over highlighting and toggle buttons is also supported.
License AGPL-3
Encoding UTF-8
LazyData true
Imports htmlwidgets, htmltools, jsonlite, ellipsis, vctrs
Suggests testthat, rmarkdown, colorspace, knitr
RoxygenNote 7.1.2.9000
VignetteBuilder knitr
Depends R (>= 2.10)
NeedsCompilation no
Repository CRAN
Date/Publication 2022-06-14 10:50:02 UTC

R topics documented:

arrows3js .................................................. 2
axis3js ..................................................... 4
background3js .......................................... 5
box3js ...................................................... 5
clippingPlane3js ........................................ 6
grid3js ..................................................... 7
group3js .................................................. 9
arrows3js

Add arrows to a data3js object

Description

Add arrows to a data3js object

Usage

arrows3js(
    data3js,
    from,
    to,
    lwd = 1,
    arrowhead_width = 0.2,
    arrowhead_length = 0.5,
    col = "black",
    mat = "lambert",
    ...
)
arrows3js

Arguments

- **data3js**: The data3js object
- **from**: nx3 matrix of coords for the arrow start points
- **to**: nx3 matrix of coords for the arrow end points
- **lwd**: line width
- **arrowhead_width**: arrowhead width
- **arrowhead_length**: arrowhead length
- **col**: color
- **mat**: material (see `material3js()`)
- **...**: other arguments to pass to `material3js()`

Value

Returns an updated data3js object

See Also

Other plot components: `axis3js()`, `box3js()`, `grid3js()`, `legend3js()`, `light3js()`, `lines3js()`, `mtext3js()`, `points3js()`, `segments3js()`, `shape3js()`, `sphere3js()`, `surface3js()`, `text3js()`, `triangle3js()`

Examples

```r
# Draw a set of arrows
from <- cbind(
  runif(10, 0.2, 0.8),
  runif(10, 0.2, 0.8),
  runif(10, 0.2, 0.8)
)

to <- jitter(from, amount = 0.2)

# Setup base plot
p <- plot3js(label_axes = FALSE)

# Add arrows
p <- arrows3js(
  p, from, to,
  arrowhead_length = 0.06,
  arrowhead_width = 0.04,
  lwd = 0.01
)

# View the plot
r3js(p, translation = c(0, 0, 0.15), zoom = 2)
```
axis3js

Add an axis to an r3js plot

Description

This is used as part of the plot3js() function but can be called separately to add an axis, generally in combination after other lower level functions like plot3js.new() and plot3js.window().

Usage

axis3js(
  data3js,
  side,
  at = NULL,
  labels = NULL,
  cornerside = "f",
  labeloffset = 0.1,
  ...
)

Arguments

data3js                  The data3js object
side                    The axis side, either "x", "y" or "z"
at                      Where to draw labels
labels                  Vector of labels to use
cornerside              See material3js()
labeloffset             Amount of offset of axis labels from the edge of the plot
...                     Other arguments to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), box3js(), grid3js(), legend3js(), light3js(), lines3js(), mtext3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(), text3js(), triangle3js()
background3js

Set the plot background color

Description

Set the plot background color

Usage

background3js(data3js, col)

Arguments

data3js The data3js object
col The background color

Value

Returns an updated data3js object

box3js

Add a box to an r3js plot

Description

Add a box to an r3js plot

Usage

box3js(
  data3js,
  sides = c("x", "y", "z"),
  dynamic = TRUE,
  col = "grey80",
  geometry = FALSE,
  renderOrder = 1,
  ...
)
clippingPlane3js

Arguments

- **data3js**: The data3js object
- **sides**: The axis side to show the box, any combination of "x", "y" or "z"
- **dynamic**: Should edges of the box closest to the viewer hide themselves automatically
- **col**: Box color
- **geometry**: Should the box be rendered as a physical geometry in the scene (see lines3js())
- **renderOrder**: The render order for the box, defaults to 1
- **...**: Other arguments to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), grid3js(), legend3js(), light3js(), lines3js(), mtext3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(), text3js(), triangle3js()

Examples

```r
p <- plot3js.new()
p <- box3js(p)
r3js(p)
```

---

clippingPlane3js **Create a clipping plane object**

Description

This function can be used to create a clipping plane that can then be applied to individual objects in a plot

Usage

```r
clippingPlane3js(coplanarPoints)
```

Arguments

- **coplanarPoints**: A matrix of 3 points coplanar to the plane, each row is a point, cols are coordinates

Value

Returns an r3js clipping plane object
Examples

# Set up plot
p <- plot3js(
  xlim = c(-2, 2),
  ylim = c(-2, 2),
  zlim = c(-2, 2)
)

# Add a sphere with clipping planes
p <- sphere3js(
  data3js = p,
  0, 0, 0,
  radius = 2,
  col = "red",
  clippingPlanes = list(
    clippingPlane3js(
      rbind(
        c(1.5,0,1),
        c(1.5,1,1),
        c(1.5,0,0)
      )
    ),
    clippingPlane3js(
      rbind(
        c(1,1.8,1),
        c(0,1.8,1),
        c(1,1.8,0)
      )
    ),
    clippingPlane3js(
      rbind(
        c(0,-1.8,1),
        c(1,-1.8,1),
        c(1,-1.8,0)
      )
    )
  )
)

# View the plot
r3js(p, zoom = 2)

grid3js

Add axis grids to a data3js object

Description

This is used for example by plot3js() to add axis grids to a plot these show along the faces of the plotting box, indicating axis ticks.
Usage

grid3js(
  data3js,
  sides = c("x", "y", "z"),
  axes = c("x", "y", "z"),
  at = NULL,
  dynamic = TRUE,
  col = "grey95",
  lwd = 1,
  geometry = FALSE,
  ...
)

Arguments

data3js The data3js object
sides The axis sides to show the box, any combination of "x", "y" or "z"
axes Axes for which to draw the grid lines
at Where to draw grid lines along the axis
dynamic Should edges of the box closest to the viewer hide themselves automatically
col Grid line color
lwd Grid line width
geometry Should the lines be rendered as a physical geometry in the scene (see lines3js())
... Other arguments to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), legend3js(), light3js(), lines3js(),
  mtext3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(), text3js(),
  triangle3js()

Examples

# Setup blank base plot
p <- plot3js(draw_grid = FALSE, xlab = "X", ylab = "Y", zlab = "Z")

# Add a box
p <- box3js(p)

# Add grid lines but only for the z axis
p <- grid3js(
  p, col = "red",
  axes = "z"
)
group3js

)

r3js(p)

# Add grid lines but only for the z axis and
# only at either end of the x axis
p <- grid3js(
  p, col = "blue",
  axes = "z",
  sides = "x"
)

r3js(p)

start a new r3js object group

Description

This function can be used to link plot objects together into a group in order to apply highlighting
and interactive effects. See details.

Usage

group3js(data3js, objectIDs, groupIDs = objectIDs)

Arguments

data3js The r3js data object
objectIDs IDs for each object you want to apply the group to.
groupIDs IDs for each object you want to include in the group.

Value

Returns an empty r3js group object in the form of a list.
lastID  
*Get the ID of the last object(s) added*

**Description**

Get the ID of the last object(s) added to a data3js object, this is useful when for example wanting to link different objects together into groups, you can use this function after adding each of them to keep a record of their unique plot id.

**Usage**

```
lastID(data3js)
```

**Arguments**

- `data3js` The data3js object

**Value**

Returns a vector of ID(s) for the last object added. After e.g. `sphere3js()`, this will simply be a single id relating to the sphere added, after e.g. `points3js()` this will be a vector of ids relating to each point in turn.

---

legend3js  
*Add a legend to a data3js object*

**Description**

Add a legend to a data3js object

**Usage**

```
legend3js(data3js, legend, fill)
```

**Arguments**

- `data3js` The data3js object
- `legend` Character vector of legend labels
- `fill` If supplied the fill color of a box placed next to each label

**Value**

Returns an updated data3js object
See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), light3js(), lines3js(), mtext3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(), text3js(), triangle3js()

Examples

```r
# Setup plot
p <- plot3js(
  x = iris$Sepal.Length,
  y = iris$Sepal.Width,
  z = iris$Petal.Length,
  col = rainbow(3)[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  zlab = "Petal Length"
)

# Add simple legend
p <- legend3js(
  data3js = p,
  legend = levels(iris$Species),
  fill = rainbow(3)
)

# View plot
r3js(p, zoom = 2)
```

---

### light3js

Add a light source to a data3js object

**Description**

When no light source is provided the 3d scene is lit from the top left, this function allows you to specify different numbers of light sources at different positions - not yet fully implemented.

**Usage**

```r
light3js(
  data3js,
  position = NULL,
  intensity = 1,
  type = "directional",
  col = "white"
)
```
Arguments

- **data3js**  
  The `data3js` object
- **position**  
  Position of the light source in x, y, z coords, see details.
- **intensity**  
  Light intensity
- **type**  
  Type of light, either "point", "directional" or "ambient", see details.
- **col**  
  Light color

Details

If light position is "directional", the default light will appear to come from the direction of the position argument but from an infinite distance. If "point" the light will appear to emanate from that position in coordinate space light a light bulb. If "ambient" any position argument is ignored and the light will light all aspects of the scene evenly from no particular position.

Value

Returns an updated `data3js` object

See Also

Other plot components: `arrows3js()`, `axis3js()`, `box3js()`, `grid3js()`, `legend3js()`, `lines3js()`, `mtext3js()`, `points3js()`, `segments3js()`, `shape3js()`, `sphere3js()`, `surface3js()`, `text3js()`, `triangle3js()`

Examples

```r
# Set up a plot
p0 <- plot3js(
  x = 1:4,
  y = c(2,1,3,4),
  z = c(3,2,4,1),
  xlim = c(0, 5),
  ylim = c(0, 5),
  zlim = c(0, 5),
  size = 20,
  col = c("white", "blue", "red", "green"),
  grid_col = "grey40",
  background = "black"
)

# Light scene intensely from above
p <- light3js(
  p0,
  position = c(0, 1, 0)
)

r3js(p, zoom = 2)

# Light scene positionally from the middle of the plot
p <- light3js(
  p0,
  position = c(0, 1, 0)
)```
Add lines to a plot, similarly to the `lines()` function. You have to decide whether you would like lines to physically exist as geometries in the scene (geometry = TRUE), i.e. as cylinders, or rather as webgl lines draw into the scene (geometry = FALSE). Such lines added will be "non-geometric" in the sense that they do not physically exist in the scene, so will not be shaded according to lighting, and their width will remain constant independent of how the plot is zoomed. As with `points3js(geometry = FALSE)` lines drawn in this way are rendered much more efficiently and sometimes the fixed width characteristic is desirable, for example grid lines are drawn in this way.

### Arguments

- **data3js**: The data3js object
- **x**: x coordinates
- **y**: y coordinates
z coordinates

lwd line width

col line color (only a single color is currently supported)

highlight highlight characteristics (see highlight3js() )

geometry logical, should the point be rendered as a physical geometry

... further parameters to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), legend3js(), light3js(), mtext3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(), text3js(), triangle3js()

Examples

# Draw three lines
x <- seq(from = 0, to = 6, length.out = 100)
y <- cos(x*5)
z <- sin(x*5)
linecols <- rainbow(100)

p <- plot3js(
  xlim = c(0, 6),
  ylim = c(0, 6),
  zlim = c(-1, 1),
  aspect = c(1, 1, 1),
  label_axes = FALSE
)

# Add a line using the linegl representation
p <- lines3js(
  data3js = p,
  x, y + 1, z,
  col = linecols
)

# Add a thicker line using the linegl representation
p <- lines3js(
  data3js = p,
  x, y + 3, z,
  lwd = 3,
  col = linecols
)

# Add a line as a physical geometry to the plot
p <- lines3js(
material3js

```
data3js = p, 
x, y + 5, z, 
lwd = 0.2, 
geometry = TRUE, 
col = "blue" # Currently only supports fixed colors
)

# View the plot
r3js(p, rotation = c(0, 0, 0), zoom = 2)
```

---

**material3js**  
*Set material properties of an r3js object*

**Description**

Arguments refer to different material properties for an object, many of which refer directly to properties as described in the `threejs` documentation.

**Usage**

```
materiial3js(  
  mat = "phong",  
  col = "black",  
  fill = "black",  
  opacity = NULL,  
  xpd = TRUE,  
  lwd = 1,  
  dashSize = NULL,  
  gapSize = NULL,  
  interactive = NULL,  
  label = NULL,  
  toggle = NULL,  
  depthWrite = NULL,  
  depthTest = NULL,  
  polygonOffset = NULL,  
  polygonOffsetFactor = NULL,  
  polygonOffsetUnits = NULL,  
  shininess = 30,  
  faces = NULL,  
  corners = NULL,  
  rotation = NULL,  
  normalise = NULL,  
  poffset = NULL,  
  clippingPlanes = NULL,  
  frontSide = TRUE,  
  backSide = TRUE,  
  renderOrder = NULL,
)```
Arguments

mat Material to use for the object, one of "basic", "lambert", "phong" or "line", see e.g. MeshBasicMaterial

col Color

fill Fill color

opacity Opacity

xpd Should parts of the object outside the plot limits be shown

lwd Line width

dashSize Dash size for dashed lines

gapSize Gap size for dashed lines

interactive Is the object interactive

label The label for the object

toggle Toggle button associated with the object

depthWrite See depthWrite

depthTest See depthTest

polygonOffset See polygonOffset

polygonOffsetFactor See polygonOffsetFactor

polygonOffsetUnits See polygonOffsetUnits

shininess Shininess of object surface

faces For dynamically hidden objects, the face with which it is associated, see details.
corners For dynamically hidden objects, the corners with which it is associated, see details.

rotation In place rotation of the object geometry (most relevant for points)

normalise Should coordinates be normalised to be with respect to axis ranges or placed according to the plotting box which has unit coordinates.

offset Positional offset, the offset is relative to the plotting area size rather than axis limits

clippingPlanes Clipping planes to apply to the object

frontSide Logical indicating whether the front side of a mesh should be rendered

backSide Logical indicating whether the back side of a mesh should be rendered

renderOrder See renderOrder

Value

Returns a list of material properties
Description

This is used for example to add axis labels but can also be used for other purposes.

Usage

\[
mtext3js(data3js, text, side, line = 0, at = 0.5, cornerside = "f", ...)\]

Arguments

data3js  The data3js object
text  The margin text
side  The axis side, either "x", "y" or "z"
line  The number of lines away from the plot edge
at  Position along the plot edge, defaults to 0.5 (middle)
cornerside  See material3js()
...  Other arguments to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), legend3js(), light3js(), lines3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(), text3js(), triangle3js()

Examples

# Create a blank plot
p <- plot3js.new()
p <- box3js(p)

# Add some margin text
p <- mtext3js(p, "0.5m", side = "x")
p <- mtext3js(p, "0.25m", side = "x", at = 0.25, line = 1)
p <- mtext3js(p, "1m", side = "y", at = 1, line = 2)
r3js(p)
plot3js

3D scatter / line plot

Description

A high level method for generating a 3D scatter or line plot.

Usage

plot3js(
  x,
  y,
  z,
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  xlab = NULL,
  ylab = NULL,
  zlab = NULL,
  label = NULL,
  type = "points",
  geometry = NULL,
  axislabel_line = 3,
  aspect = NULL,
  label_axes = c("x", "y", "z"),
  draw_box = TRUE,
  draw_grid = TRUE,
  grid_lwd = 1,
  grid_col = "grey90",
  axis_lwd = grid_lwd,
  box_lwd = grid_lwd,
  box_col = grid_col,
  background = "#ffffff",
  ...
)

Arguments

x  x coords for points / lines
y  y coords for points / lines
z  z coords for points / lines
xlim  plot x limits
ylim  plot y limits
zlim  plot z limits
xlab  x axis label
plot3js

- ylab: y axis label
- zlab: z axis label
- label: optional vector of interactive point labels
- type: one of "points" or "lines"
- geometry: should points and lines be represented as physical geometries? Default for points is TRUE and for lines is FALSE, see points() and lines() for more information.
- axislabel_line: Distance of axis label from plot
- aspect: Plot axis aspect ratio, see plot3js.window()
- label_axes: Vector of axes to label, any combination of "x", "y" and "z"
- draw_box: Should a box be drawn around the plot
- draw_grid: Should an axis grid be drawn in the background
- grid_lwd: Grid line width
- grid_col: Grid line color
- axis_lwd: Axis line width
- box_lwd: Box line width
- box_col: Box color
- background: Background color for the plot
- ...: Further parameters to pass to material3js()

Value

Returns a data3js object, that can be plotted as a widget using print() or r3js() or further added to with the other plotting functions.

Examples

```r
# Simple plot example
p <- plot3js(
  x = iris$Sepal.Length,
  y = iris$Sepal.Width,
  z = iris$Petal.Length,
  col = rainbow(3)[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  zlab = "Petal Length"
)

r3js(p, zoom = 2)

# Plotting with point rollover info and highlighting
p <- plot3js(
  x = USJudgeRatings$CONT,
  y = USJudgeRatings$INTG,
  z = USJudgeRatings$DMNR,
  label = paste("Judge ", USJudgeRatings$NAME),
  label_axes = "x",
  label_size = 0.8
)

r3js(p, zoom = 2)
```

highlight = list(
    col = "darkgreen",
    size = 2.5
),
  xlab = "CONT",
  ylab = "INTG",
  zlab = "DMNR",
  size = 2,
  col = "green",
  label = rownames(USJudgeRatings)
)

r3js(p, zoom = 2)

---

**plot3js.new**

Setup a new r3js plot

**Description**

This function sets up a new r3js plot and returns an r3js plotting object that can later be added to using other functions such as points3js() and lines3js() etc. It is in many ways equivalent to the plot.new() command.

**Usage**

```r
plot3js.new(background = "ffffff")
```

**Arguments**

- `background` Background color to use

**Value**

Returns a new data3js plotting object

---

**plot3js.window**

Set axis limits for a data3js object

**Description**

This is similar to the plot.window() command except that plot limits can only be set once for each plot.

**Usage**

```r
plot3js.window(data3js, xlim, ylim, zlim, aspect = NULL)
```
**points3js**

**Arguments**

- `data3js` The data3js object
- `xlim` x axis limits
- `ylim` y axis limits
- `zlim` z axis limits
- `aspect` vector of length 3 giving the aspect ratio, or null to automatically set the aspect ratio such that axes have the same visual length

**Value**

Returns an updated data3js object

---

**points3js**  
*Add points to a data3js object*

**Description**

This is the base function for adding points to a plot. Alongside other parameters you will need to decide whether you want the points plotted as physical geometries (geometry = TRUE) or webgl points rendered with a shader (geometry = FALSE). Points rendered as geometries use geopoint3js() and will respect lighting and intersect properly, also more point types are supported but come at a larger computational cost of rendering. webgl points use glpoints3js() and are rendered orders of magnitude faster but have less flexible appearances and ignore lighting.

**Usage**

```r
points3js(  
data3js,  
x,  
y,  
z,  
size = 1,  
col = "black",  
fill = col,  
shape = "sphere",  
highlight,  
geometry = TRUE,  
label = NULL,  
toggle = NULL,  
...  
)
```
Arguments

- `data3js` The data3js object
- `x` point x coords
- `y` point y coords
- `z` point z coords
- `size` point sizes
- `col` point colors
- `fill` point fill color
- `shape` point shapes, see the examples below for a list of different types.
- `highlight` highlight characteristics (see `highlight3js()`)
- `geometry` logical, should the point be rendered as a physical geometry
- `label` optional vector of interactive labels to apply to the points (see `highlight3js()`)  
- `toggle` optional vector of interactive toggles associate to each point (see `highlight3js()`)  
- ... further parameters to pass to `material3js()`

Value

Returns an updated data3js object

See Also

Other plot components: `arrows3js()`, `axis3js()`, `box3js()`, `grid3js()`, `legend3js()`, `light3js()`, `lines3js()`, `mtext3js()`, `segments3js()`, `shape3js()`, `sphere3js()`, `surface3js()`, `text3js()`, `triangle3js()`

Examples

geo_shapes <- c(
  "circle", "square", "triangle",
  "circle open", "square open", "triangle open",
  "circle filled", "square filled", "triangle filled",
  "sphere", "cube", "tetrahedron",
  "cube open",
  "cube filled"
)

gl_shapes <- c(
  "circle", "square", "triangle",
  "circle open", "square open", "triangle open",
  "circle filled", "square filled", "triangle filled",
  "sphere"
)

# Setup base plot
p <- plot3js(
  xlim = c(0, length(geo_shapes) + 1),
points3js

```r
ylim = c(-4, 4),
zlim = c(-4, 4),
label_axes = FALSE
)

# Plot the different point geometries
p <- points3js(
data3js = p,
x = seq_along(geo_shapes),
y = rep(0, length(geo_shapes)),
z = rep(0, length(geo_shapes)),
size = 2,
shape = geo_shapes,
col = rainbow(length(geo_shapes)),
fill = "grey70"
)

r3js(p, rotation = c(0, 0, 0), zoom = 2)

# Setup base plot
p <- plot3js(
xlim = c(0, length(gl_shapes) + 1),
ylim = c(-4, 4),
zlim = c(-4, 4),
label_axes = FALSE
)

# Plot the different gl points
p <- points3js(
data3js = p,
x = seq_along(gl_shapes),
y = rep(0, length(gl_shapes)),
z = rep(0, length(gl_shapes)),
size = 2,
shape = gl_shapes,
col = rainbow(length(gl_shapes)),
fill = "grey50",
geometry = FALSE
)

r3js(p, rotation = c(0, 0, 0), zoom = 2)

# Plot a 10,000 points using the much more efficient gl_point representation

# Setup base plot
p <- plot3js(
xlim = c(-4, 4),
ylim = c(-4, 4),
zlim = c(-4, 4),
label_axes = FALSE
)

p <- points3js(
```
data3js = p,
x = rnorm(10000, 0),
y = rnorm(10000, 0),
z = rnorm(10000, 0),
size = 0.6,
col = rainbow(10000),
shape = "sphere",
geometry = FALSE
)

r3js(p, rotation = c(0, 0, 0), zoom = 2)

---

**r3js**

*Plot a data3js object*

**Description**

This function takes the assembled data3js object and plots it as an htmlwidget.

**Usage**

```r
r3js(
  data3js,
  rotation = c(-1.45, 0, -2.35),
  zoom = 2,
  translation = c(0, 0, 0),
  styles = list(),
  title = "R3JS viewer",
  ...)
```

**Arguments**

- **data3js**: The data3js object
- **rotation**: Plot starting rotation as an XYZ Euler rotation
- **zoom**: Plot starting zoom factor
- **translation**: Plot starting translation
- **styles**: List of styles controlling elements of the plot, see examples
- **title**: Title for the viewer
- **...**: Additional arguments to pass to htmlwidgets::createWidget()

**Value**

Returns an html widget of the plot
Examples

```r
# Control toggle button appearance
r3js(
  plot3js(
    x = iris$Sepal.Length,
    y = iris$Sepal.Width,
    z = iris$Petal.Length,
    col = rainbow(3)[iris$Species],
    xlab = "Sepal Length",
    ylab = "Sepal Width",
    zlab = "Petal Length",
    toggle = iris$Species
  ),
  styles = list(
    togglediv = list(
      bottom = "4px",
      right = "4px"
    ),
    toggles = list(
      setosa = list(
        on = list(backgroundColor = colorspace::darken(rainbow(3)[1], 0.1), color = "white"),
        off = list(backgroundColor = colorspace::lighten(rainbow(3)[1], 0.8), color = "white")
      ),
      versicolor = list(
        on = list(backgroundColor = colorspace::darken(rainbow(3)[2], 0.1), color = "white"),
        off = list(backgroundColor = colorspace::lighten(rainbow(3)[2], 0.8), color = "white")
      ),
      virginica = list(
        on = list(backgroundColor = colorspace::darken(rainbow(3)[3], 0.1), color = "white"),
        off = list(backgroundColor = colorspace::lighten(rainbow(3)[3], 0.8), color = "white")
      )
    ),
    zoom = 1.5
  )
)
```

---

**r3js-shiny**  
*Shiny bindings for r3js*

**Description**

Output and render functions for using r3js within Shiny applications and interactive Rmd documents.

**Usage**

```r
r3jsOutput(outputId, width = "100\%", height = "400px")
```

```r
renderR3js(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 a r3js
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.

Value

An output or render function that enables the use of the widget within Shiny applications.

Description

Converts r3js plot data to a widget and saves it to an HTML file (e.g. for sharing with others)

Usage

```r
save3js(
  data3js,
  file,
  title = "r3js plot",
  selfcontained = TRUE,
  libdir = NULL,
  ...
)
```

Arguments

- data3js  The r3js data object to be saved
- file  File to save HTML into
- title  Text to use as the title of the generated page
- selfcontained  Whether to save the HTML as a single self-contained file (with external resources base64 encoded) or a file with external resources placed in an adjacent directory.
- libdir  Directory to copy HTML dependencies into (defaults to filename_files)
- ...  Further arguments to pass to r3js()

Value

No return value, called for the side-effect of saving the plot.
save3jsWidget  
_Save an r3js widget to an HTML file_

Description

Save a rendered r3js widget to an HTML file (e.g. for sharing with others). This is mostly a wrapper for `saveWidget`.

Usage

```r
save3jsWidget(
  widget,  # Widget to save
  file,    # File to save HTML into
  title = "r3js plot",  # Text to use as the title of the generated page
  selfcontained = TRUE,  # Whether to save the HTML as a single self-contained file (with external resources base64 encoded) or a file with external resources placed in an adjacent directory
  libdir = NULL,  # Directory to copy HTML dependencies into (defaults to filename_files)
  ...  # Further arguments to pass to saveWidget
)
```

Arguments

- **widget**: Widget to save
- **file**: File to save HTML into
- **title**: Text to use as the title of the generated page
- **selfcontained**: Whether to save the HTML as a single self-contained file (with external resources base64 encoded) or a file with external resources placed in an adjacent directory
- **libdir**: Directory to copy HTML dependencies into (defaults to filename_files)
- **...**: Further arguments to pass to `saveWidget`

Value

No return value, called for the side-effect of saving the plot.

segments3js  
__Add lines segments a 3js object__

Description

Add lines segments a 3js object
Usage

segments3js(  
data3js,  
x,  
y,  
z,  
lwd = 1,  
col = "black",  
highlight,  
geometry = FALSE,  
...  
)

Arguments

data3js The data3js object
x x coords
y y coords
z z coords
lwd line width
col line color
highlight highlight characteristics (see highlight3ks())
geometry logical, should the lines be rendered as a physical geometries
...

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), legend3js(), light3js(), lines3js(), mtext3js(), points3js(), shape3js(), sphere3js(), surface3js(), text3js(), triangle3js()

Examples

# Draw three lines
x <- seq(from = 0, to = 6, length.out = 100)
y <- cos(x*5)
z <- sin(x*5)
linecols <- rainbow(100)

p <- plot3js(  
xlim = c(0, 6),  
ylim = c(0, 6),  
zlim = c(-1, 1),  
)
shape3js

Add a generic shape to an 3js plot

Description

Add a generic shape to an 3js plot

Usage

shape3js(
  data3js, vertices, faces, normals = NULL, col = "black", highlight, ...
)
Arguments

- **data3js**: The data3js object
- **vertices**: An nx3 matrix of 3d vertex coordinates
- **faces**: An nx3 matrix of indices relating to vertices that make up each triangular face
- **normals**: Optional nx3 matrix of normals to each vertex
- **col**: Shape color
- **highlight**: highlight attributes (see highlight3js())
- ... Additional attributes to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), legend3js(), light3js(), lines3js(), mtext3js(), points3js(), segments3js(), sphere3js(), surface3js(), text3js(), triangle3js()

Examples

```r
# Draw a teapot
data(teapot)
p <- plot3js(  
  xlim = range(teapot$vertices[,1]),
  ylim = range(teapot$vertices[,2]),
  zlim = range(teapot$vertices[,3]),
  label_axes = FALSE,
  aspect = c(1, 1, 1)
)
p <- shape3js(  
p,  
  vertices = teapot$vertices,
  faces = teapot$edges,
  col = "lightblue"
)
r3js(p, rotation = c(-2.8, 0, 3.14), zoom = 1.2)
```
sphere3js

Add a sphere of defined radius to a data3js object

Description

Unlike points3js, where geometric points can also be represented as spheres, this adds sphere that is sized with respect to the actual dimensions of the plotting space (and so if aspect ratios differ for each axis may not actually appear sphere-like).

Usage

sphere3js(data3js, x, y, z, radius, col = "black", highlight, ...)

Arguments

data3js The data3js object
x x coordinate of the sphere center
y y coordinate of the sphere center
z z coordinate of the sphere center
radius sphere radius
col color
highlight highlight attributes (see highlight3js())
... other arguments to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), legend3js(), light3js(), lines3js(), mtext3js(), points3js(), segments3js(), shape3js(), surface3js(), text3js(), triangle3js()

Examples

# Setup base plot
p <- plot3js(
  xlim = c(-10, 10),
  ylim = c(-5, 5),
  zlim = c(-8, 8)
)

# Add sphere (this will look distorted because of axis scaling)
p <- sphere3js(
  data3js = p,
### surface3js

Add a surface to a data3js object

#### Description

This function behaves very similarly to the surface3d function in the rgl package, although the handling of NA values are handled differently.

#### Usage

```r
surface3js(
  data3js,
  x,
  y,
  z,
  col = "black",
  mat,
  wireframe = FALSE,
  highlight,
  ...
)
```
Arguments

- `data3js` The data3js object
- `x` Values corresponding to rows of `z`, or matrix of `x` coordinates
- `y` Values corresponding to the columns of `z`, or matrix of `y` coordinates
- `z` Matrix of heights
- `col` The color of the surface as either a single value, vector or matrix.
- `mat` The material to use when drawing the matrix, for a solid surface the default is "phong", for a wireframe the default is "line".
- `wireframe` Logical value for if the surface should be displayed as a mesh
- `highlight` highlight attributes (see `highlight3js()`)
- `...` Material and texture properties. See `material3js()`

Value

Returns an updated data3js object

See Also

Other plot components: `arrows3js()`, `axis3js()`, `box3js()`, `grid3js()`, `legend3js()`, `light3js()`, `lines3js()`, `mtext3js()`, `points3js()`, `segments3js()`, `shape3js()`, `sphere3js()`, `text3js()`, `triangle3js()`

Examples

```r
# volcano example taken from "persp"
z <- 2 * volcano # Exaggerate the relief
x <- 10 * (1:nrow(z)) # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z)) # 10 meter spacing (E to W)

zlim <- range(z)

colorlut <- terrain.colors(zlen) # height color lookup table
col <- colorlut[ z - zlim[1] + 1 ] # assign colors to heights for each point

p <- plot3js(
xlim = range(x),
ylim = range(y),
zi = range(z),
label_axes = FALSE,
aspect = c(1, 1, 1) # Maintain a constant aspect ratio
)

p <- surface3js(
data3js = p,
x, y, z,
col = col
)
```
r3js(
  data3js = p,
  rotation = c(-1.15, 0, -0.65),
  zoom = 1.5
)

teapot

Utah Teapot

Description

The Utah teapot is a classic computer graphics example. This data set contains a representation in terms of triangles. This is taken from the misc3d package.

Usage

teapot

Format

A list with components vertices and edges. vertices is a 1976 by 3 numeric matrix of the coordinates of the vertices. edges is a 3751 by 3 integer matrix of the indices of the triangles.

Source

Taken from the misc3d package

text3js

Add text to a data3js object

Description

The text added can either be as an html text object, superimposed on the scene but moving relative to appear relative to the specified coordinates, or an actual geometry, which will appear in the scene, zoom and rotate with it etc.
text3js

Usage

text3js(
  data3js,
  x,
  y,
  z,
  text,
  size = NULL,
  col = "inherit",
  toggle = NULL,
  type = "geometry",
  alignment = "center",
  offset = c(0, 0),
  style = list(fontFamily = "sans-serif"),
  ...
)

Arguments

data3js The data3js object
x x coords
y y coords
z z coords
text character vector of text
size text size, if type is "geometry" this is interpreted in terms of text height within
the plotting space (default 1), if type is "html" then this is interpreted as size in
pts (default 16).
col text color
toggle associated text toggle button
type text type, either "geometry" or "html"
alignment text alignment, i.e. "left" "top" "topright"
offset onscreen text offset for html text, x then y
style named list of css style attributes to apply to the html text
... Additional attributes to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), legend3js(), light3js(),
lines3js(), mtext3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(),
triangle3js()
Examples

```r
# Set text parameters
x <- 1:4
y <- rep(0, 4)
z <- rep(0, 4)
labels <- LETTERS[1:4]
sizes <- c(0.4, 0.6, 0.8, 1)

# Create empty plot
p0 <- plot3js(
    xlim = c(0, 5),
    ylim = c(-1, 1),
    zlim = c(-1, 1),
    aspect = c(1, 1, 1),
    label_axes = FALSE
)

# Add text as a geometry
p <- text3js(
    data3js = p0,
    x = x,
    y = y,
    z = z,
    size = sizes,
    text = labels
)

r3js(p, rotation = c(0, 0, 0), zoom = 1)

# Add text as a html labels
p <- text3js(
    data3js = p0,
    x = x,
    y = y,
    z = z,
    size = sizes*40,
    text = labels,
    type = "html"
)

r3js(p, rotation = c(0, 0, 0), zoom = 1)
```

---

triangle3js

Add a triangle to a data3js object

Description

Add a triangle to a data3js object
triangle3js

Usage

triangle3js(data3js, vertices, col = "black", highlight, ...)

Arguments

- **data3js**: The data3js object
- **vertices**: An nx3 matrix of triangle vertices
- **col**: Single color for the triangles or vector of vertex colors
- **highlight**: highlight attributes (see highlight3js())
- **...**: Additional attributes to pass to material3js()

Value

Returns an updated data3js object

See Also

Other plot components: arrows3js(), axis3js(), box3js(), grid3js(), legend3js(), light3js(), lines3js(), mtext3js(), points3js(), segments3js(), shape3js(), sphere3js(), surface3js(), text3js()

Examples

```r
# Draw some random triangles
M <- matrix(
  data = rnorm(36),
  ncol = 3,
  nrow = 12
)

p <- plot3js(
  xlim = range(M[,1]),
  ylim = range(M[,2]),
  zlim = range(M[,3]),
  label_axes = FALSE
)

p <- triangle3js(
  p,
  vertices = M,
  col = rainbow(nrow(M))
)

r3js(p, zoom = 2)
```
Index

* datasets
  teapot, 34

* plot components
  arrows3js, 2
  axis3js, 4
  box3js, 5
  grid3js, 7
  legend3js, 10
  light3js, 11
  lines3js, 13
  mtext3js, 17
  points3js, 21
  segments3js, 27
  shape3js, 29
  sphere3js, 31
  surface3js, 32
  text3js, 34
  triangle3js, 36

arrows3js, 2, 4, 6, 8, 11, 12, 14, 17, 22, 28, 30, 31, 33, 35, 37
axis3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 28, 30, 31, 33, 35, 37
background3js, 5
box3js, 3, 4, 5, 8, 11, 12, 14, 17, 22, 28, 30, 31, 33, 35, 37
clippingPlane3js, 6
grid3js, 3, 4, 6, 7, 11, 12, 14, 17, 22, 28, 30, 31, 33, 35, 37
group3js, 9
lastID, 10
legend3js, 3, 4, 6, 8, 10, 12, 14, 17, 22, 28, 30, 31, 33, 35, 37
light3js, 3, 4, 6, 8, 11, 11, 14, 17, 22, 28, 30, 31, 33, 35, 37
lines3js, 3, 4, 6, 8, 11, 12, 13, 17, 22, 28, 30, 31, 33, 35, 37
material3js, 15
mtext3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 28, 30, 31, 33, 35, 37
plot3js, 18
plot3js.new, 20
plot3js.window, 20
points3js, 3, 4, 6, 8, 11, 12, 14, 17, 21, 28, 30, 31, 33, 35, 37
r3js, 24
r3js-shiny, 25
r3jsOutput (r3js-shiny), 25
renderR3js (r3js-shiny), 25
save3js, 26
save3jsWidget, 27
saveWidget, 27
segments3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 27, 30, 31, 33, 35, 37
shape3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 28, 29, 31, 33, 35, 37
sphere3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 28, 30, 31, 33, 35, 37
surface3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 28, 30, 31, 32, 35, 37
teatop, 34
text3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 28, 30, 31, 33, 34, 37
triangle3js, 3, 4, 6, 8, 11, 12, 14, 17, 22, 28, 30, 31, 33, 35, 36