

Package ‘deckgl’

November 19, 2018

Title An R Interface to 'deck.gl'

Version 0.1.8

Date 2018-11-10

Maintainer Stefan Kuethe <crazycapivara@gmail.com>

Description Makes 'deck.gl' <<https://deck.gl/>>, a WebGL-powered open-source JavaScript framework for visual exploratory data analysis of large datasets, available within R via the 'htmlwidgets' package.

Furthermore, it supports basemaps from 'mapbox' <<https://www.mapbox.com/>> via 'mapbox-gl-js' <<https://github.com/mapbox/mapbox-gl-js>>.

URL <https://github.com/crazycapivara/deckgl/>,
<https://crazycapivara.github.io/deckgl/>

BugReports <https://github.com/crazycapivara/deckgl/issues/>

Depends R (>= 3.3)

Imports htmlwidgets, htmltools, magrittr, base64enc, yaml, jsonlite,
readr, tibble

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

Suggests knitr, rmarkdown, testthat, rprojroot

VignetteBuilder knitr

NeedsCompilation no

Author Stefan Kuethe [aut, cre]

Repository CRAN

Date/Publication 2018-11-19 18:10:06 UTC

R topics documented:

add_arc_layer	2
add_contour_layer	3
add_data	5
add_geojson_layer	6
add_grid_layer	7
add_hexagon_layer	8
add_icon_layer	9
add_layer	10
add_line_layer	11
add_mapbox_basemap	12
add_path_layer	13
add_point_cloud_layer	14
add_polygon_layer	15
add_scatterplot_layer	16
add_screen_grid_layer	17
add_text_layer	18
bart_segments	19
bart_stations	20
contour_definition	20
deckgl	21
deckgl-shiny	22
default_icon_properties	22
does_it_work	23
encode_icon_atlas	23
get_color_to_rgb_array	24
get_data	24
get_position	25
get_property	25
icon_definition	26
sf_bike_parking	26
Index	28

add_arc_layer	<i>Add an arc layer to the deckgl widget</i>
---------------	--

Description

The ArcLayer renders raised arcs joining pairs of source and target points, specified as latitude/longitude coordinates.

Usage

```
add_arc_layer(deckgl, id = "arc-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a <code>getTooltip</code> property (callback) showing a tooltip when the mouse enters an object, e. g. <code>getTooltip = JS("object => object.name")</code>
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/arc-layer>

Examples

```
## @knitr arc-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/master/",
  "website/bart-segments.json"
)

properties <- list(
  pickable = TRUE,
  getStrokeWidth = 12,
  getSourcePosition = get_property("from.coordinates"),
  getTargetPosition = get_property("to.coordinates"),
  getSourceColor = JS("d => [Math.sqrt(d.inbound), 140, 0]"),
  getTargetColor = JS("d => [Math.sqrt(d.outbound), 140, 0]"),
  getTooltip = JS("object => `${object.from.name} to ${object.to.name}`")
)

deck <- deckgl(zoom = 10, pitch = 35) %>%
  add_arc_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

add_contour_layer *Add a contour layer to the deckgl widget*

Description

The ContourLayer renders contour lines for a given threshold and cell size. Internally it implements [Marching Squares](#) algorithm to generate contour line segments and feeds them into LineLayer to render lines.

Usage

```
add_contour_layer(deckgl, id = "contour-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a <code>getTooltip</code> property (callback) showing a tooltip when the mouse enters an object, e. g. <code>getTooltip = JS("object => object.name")</code>
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/contour-layer>

Examples

```
## @knitr contour-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/sf-bike-parking.json"
)

contours <- list(
  contour_definition(
    threshold = 1,
    color = c(255, 0, 0),
    strokeWidth = 2
  ),
  contour_definition(
    threshold = 5,
    color = c(0, 255, 0),
    strokeWidth = 3
  ),
  contour_definition(
    threshold = 15,
    color = c(0, 0, 255),
    strokeWidth = 5
  )
)

properties <- list(
  contours = contours,
  cellSize = 200,
```

```

    elevationScale = 4,
    getPosition = get_property("COORDINATES")
  )

deck <- deckgl(zoom = 10.5, pitch = 30) %>%
  add_contour_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck

```

add_data

Add JavaScript data file

Description

EXPERIMENTAL

Usage

```
add_data(deckgl, data, var_name = "thanksForAllTheFish")
```

Arguments

deckgl	deckgl widget
data	data object
var_name	JavaScript variable name used to make the data available

Examples

```

properties <- list(
  extruded = TRUE,
  cellSize = 200,
  elevationScale = 4,
  getPosition = get_position("lat", "lng")
)

deck <- deckgl(pitch = 45) %>%
  add_data(sf_bike_parking, "sfBikeParking") %>%
  add_grid_layer(
    data = get_data("sfBikeParking"),
    properties = properties
  )

if (interactive()) deck

```

add_geojson_layer *Add a geojson layer to the deckgl widget*

Description

The GeoJsonLayer takes in **GeoJson** formatted data and renders it as interactive polygons, lines and points.

Usage

```
add_geojson_layer(deckgl, id = "geojson-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/geojson-layer>

Examples

```
## @knitr geojson-layer
geojson <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/bart.geo.json"
)

deck <- deckgl(zoom = 10, pickingRadius = 5) %>%
  add_geojson_layer(
    data = geojson,
    filled = TRUE,
    extruded = TRUE,
    getRadius = 100,
    lineWidthScale = 20,
    lineWidthMinPixels = 2,
    getLineWidth = 1,
```

```

    getLineColor = get_color_to_rgb_array("properties.color || 'black'"),
    getFillColor = c(160, 160, 180, 200),
    getElevation = 30,
    getTooltip = JS("object => object.properties.name || object.properties.station")
  ) %>%
  add_mapbox_basemap()

if (interactive()) deck

```

add_grid_layer

Add a grid layer to the deckgl widget

Description

The GridLayer renders a grid heatmap based on an array of points. It takes the constant size all each cell, projects points into cells. The color and height of the cell is scaled by number of points it contains.

Usage

```
add_grid_layer(deckgl, id = "grid-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/grid-layer>

Examples

```
## @knitr grid-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/sf-bike-parking.json"
)
```

```

properties <- list(
  extruded = TRUE,
  cellSize = 200,
  elevationScale = 4,
  getPosition = get_property("COORDINATES"),
  getTooltip = JS("object => `${object.position.join(', ')}<br/>Count: ${object.count}`"),
  fixedTooltip = TRUE
)

deck <- deckgl(zoom = 11, pitch = 45, bearing = 35) %>%
  add_grid_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck

```

add_hexagon_layer *Add a hexagon layer to the deckgl widget*

Description

The HexagonLayer renders a hexagon heatmap based on an array of points. It takes the radius of hexagon bin, projects points into hexagon bins. The color and height of the hexagon is scaled by number of points it contains.

Usage

```
add_hexagon_layer(deckgl, id = "hexagon-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/hexagon-layer>

Examples

```
## @knitr hexagon-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/sf-bike-parking.json"
)

properties <- list(
  extruded = TRUE,
  radius = 200,
  elevationScale = 4,
  getPosition = get_property("COORDINATES"),
  getTooltip = JS("object => `${object.centroid.join(', ')}<br/>Count: ${object.points.length}`"),
  fixedTooltip = TRUE
)

deck <- deckgl(zoom = 11, pitch = 45, bearing = 35) %>%
  add_hexagon_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

 add_icon_layer

 Add an icon layer to the deckgl widget

Description

The IconLayer renders raster icons at given coordinates.

Usage

```
add_icon_layer(deckgl, id = "icon-layer", data = NULL,
  properties = default_icon_properties(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/icon-layer>

Examples

```
## @knitr icon-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/bart-stations.json"
)

properties <- list(
  pickable = TRUE,
  iconAtlas = encode_icon_atlas(),
  iconMapping = list(marker = icon_definition()),
  sizeScale = 10,
  getPosition = get_property("coordinates"),
  getIcon = JS("d => 'marker'"),
  getSize = 5,
  getColor = JS("d => [Math.sqrt(d.exits), 140, 0]"),
  getTooltip = JS("object => `${object.name}<br/>${object.address}`")
)

deck <- deckgl(zoom = 10, pitch = 45) %>%
  add_icon_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

add_layer

Add any kind of layer to the deckgl widget

Description

Generic function to add any kind of layer to the deckgl widget. Usually you will not use this one but any of the add_*_layer functions instead.

Usage

```
add_layer(deckgl, class_name, id, data, properties = list(), ...)
```

Arguments

deckgl	deckgl widget
class_name	name of the js layer class, e. g. ScatterplotLayer
id	id of the layer
data	url to fetch data from or data object

properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

Value

deckgl widget

add_line_layer	<i>Add a line layer to the deckgl widget</i>
----------------	--

Description

The LineLayer renders flat lines joining pairs of source and target points, specified as latitude/longitude coordinates.

Usage

```
add_line_layer(deckgl, id = "line-layer", data = NULL,
               properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/line-layer>

Examples

```
## @knitr line-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/bart-segments.json"
)

properties <- list(
  pickable = TRUE,
  getStrokeWidth = 12,
  getSourcePosition = get_property("from.coordinates"),
  getTargetPosition = get_property("to.coordinates"),
  getColor = JS("d => [Math.sqrt(d.inbound + d.outbound), 140, 0]"),
  getTooltip = JS("object => `${object.from.name} to ${object.to.name}`")
)

deck <- deckgl(zoom = 10, pitch = 20) %>%
  add_line_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

add_mapbox_basemap *Add a base map from mapbox to the deckgl widget*

Description

Add a base map from mapbox to the deckgl widget

Usage

```
add_mapbox_basemap(deckgl, token = Sys.getenv("MAPBOX_API_TOKEN"),
  style = "mapbox://styles/mapbox/light-v9")
```

Arguments

deckgl	deckgl widget
token	mapbox api access token
style	map style

Value

deckgl widget

add_path_layer	<i>Add a path layer to the deckgl widget</i>
----------------	--

Description

The PathLayer takes in lists of coordinate points and renders them as extruded lines with mitering.

Usage

```
add_path_layer(deckgl, id = "path-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/path-layer>

Examples

```
## @knitr path-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/bart-lines.json"
)

properties <- list(
  pickable = TRUE,
  widthScale = 20,
  widthMinPixels = 2,
  getPath = get_property("path"),
  getColor = get_color_to_rgb_array("color"),
  getWidth = 5,
  getTooltip = get_property("name")
)

deck <- deckgl(pitch = 25, zoom = 10.5) %>%
```

```

add_path_layer(data = sample_data, properties = properties) %>%
add_mapbox_basemap()

if (interactive()) deck

```

add_point_cloud_layer *Add a point cloud layer to the deckgl widget*

Description

The PointCloudLayer takes in points with 3d positions, normals and colors and renders them as spheres with a certain radius.

Usage

```

add_point_cloud_layer(deckgl, id = "point-cloud-layer", data = NULL,
properties = list(), ...)

```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/point-cloud-layer>

Examples

```

## @knitr point-cloud-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/pointcloud.json"
)

properties <- list(
  pickable = TRUE,
  coordinateSystem = JS("COORDINATE_SYSTEM.METER_OFFSETS"),
  coordinateOrigin = c(-122.4, 37.74),
  radiusPixels = 4,

```

```

    getPosition = get_property("position"),
    getNormal = get_property("normal"),
    getColor = get_property("color"),
    lightSettings = list(),
    getTooltip = JS("object => object.position.join(', ')")
  )

deck <- deckgl(pitch = 45, zoom = 10.5) %>%
  add_point_cloud_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck

```

add_polygon_layer *Add a polygon layer to the deckgl widget*

Description

The PolygonLayer renders filled and/or stroked polygons.

Usage

```
add_polygon_layer(deckgl, id = "polygon-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/polygon-layer>

Examples

```
## @knitr polygon-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/sf-zipcodes.json"
```

```

)

properties <- list(
  pickable = TRUE,
  stroked = TRUE,
  filled = TRUE,
  wireframe = TRUE,
  lineWidthMinPixels = 1,
  getPolygon = get_property("contour"),
  getElevation = JS("d => d.population / d.area / 10"),
  getFillColor = JS("d => [d.population / d.area / 60, 140, 0]"),
  getLineColor = c(80, 80, 80),
  getLineWidth = 1,
  getTooltip = JS("object => `${object.zipcode}<br/>Population: ${object.population}`")
)

deck <- deckgl(zoom = 11, pitch = 25) %>%
  add_polygon_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck

```

add_scatterplot_layer *Add a scatterplot layer to the deckgl widget*

Description

The ScatterplotLayer takes in paired latitude and longitude coordinated points and renders them as circles with a certain radius.

Usage

```
add_scatterplot_layer(deckgl, id = "scatterplot-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/scatterplot-layer>

Examples

```
## @knitr scatterplot-layer
bart_stations <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/bart-stations.json"
)

properties <- list(
  getPosition = get_property("coordinates"),
  getRadius = JS("data => Math.sqrt(data.exits)"),
  radiusScale = 6,
  getColor = c(255, 140, 20)
)

deck <- deckgl(zoom = 10.5, pitch = 35) %>%
  add_scatterplot_layer(data = bart_stations, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

add_screen_grid_layer *Add a screen grid layer to the deckgl widget*

Description

The ScreenGridLayer takes in an array of latitude and longitude coordinated points, aggregates them into histogram bins and renders as a grid.

Usage

```
add_screen_grid_layer(deckgl, id = "screen-grid-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")
...	more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/screen-grid-layer>

Examples

```
## @knitr screen-grid-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/uber-common/",
  "deck.gl-data/master/",
  "website/sf-bike-parking.json"
)

properties <- list(
  pickable = FALSE,
  opacity = 0.8,
  cellSizePixels = 50,
  minColor = c(0, 0, 0, 0),
  maxColor = c(0, 180, 0, 255),
  getPosition = get_property("COORDINATES"),
  getWeight = get_property("SPACES")
)

deck <- deckgl() %>%
  add_screen_grid_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

add_text_layer	<i>Add a text layer to the deckgl widget</i>
----------------	--

Description

The TextLayer renders text labels on the map using texture mapping.

Usage

```
add_text_layer(deckgl, id = "text-layer", data = NULL,
  properties = list(), ...)
```

Arguments

deckgl	deckgl widget
id	id of the layer
data	url to fetch data from or data object
properties	named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class, additionally there is a getTooltip property (callback) showing a tooltip when the mouse enters an object, e. g. getTooltip = JS("object => object.name")

... more properties (will be added to the properties object), useful if you want to use a properties object for more than one layer

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/layers/text-layer>

Examples

```
## @knitr text-layer
deck <- deckgl(zoom = 10, pitch = 35) %>%
  add_text_layer(
    data = bart_stations,
    pickable = TRUE,
    getPosition = get_position("lat", "lng"),
    getText = get_property("name"),
    getSize = 15,
    getAngle = 0,
    getTextAnchor = "middle",
    getAlignmentBaseline = "center",
    getTooltip = JS("object =>`${object.name}<br/>${object.address}`")
  ) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

bart_segments

bart segments

Description

bart segments

Usage

bart_segments

Format

tibble with 45 rows and 8 variables:

inbound number of inbound trips
outbound number of outbound trips
from_name name of source station
from_lng longitude of source station
from_lat latitude of source station
to_name name of target station
to_lng longitude of target station
to_lat latitude of target station

Source

<https://raw.githubusercontent.com/uber-common/deck.gl-data/master/website/bart-segments.json>

bart_stations	<i>bart stations</i>
---------------	----------------------

Description

bart stations

Usage

bart_stations

Format

tibble with 44 rows and 7 variables:

name station name
code two-letter station code
address address
entries number of entries
exits number of exits
lng longitude
lat latitude

Source

<https://raw.githubusercontent.com/uber-common/deck.gl-data/master/website/bart-stations.json>

contour_definition	<i>Contour definition</i>
--------------------	---------------------------

Description

Contour definition

Usage

```
contour_definition(threshold = 1, color = c(255, 255, 255),  
  strokeWidth = 1)
```

Arguments

threshold	threshold value to be used in contour generation
color	RGB color array to be used to render contour lines
strokeWidth	width of the contour lines in pixels

deckgl	<i>Create a deckgl widget</i>
--------	-------------------------------

Description

Create a deckgl widget

Usage

```
deckgl(latitude = 37.8, longitude = -122.45, zoom = 12, pitch = 0,
        bearing = 0, initialState = NULL, views = NULL, width = NULL,
        height = NULL, elementId = NULL, ...)
```

Arguments

latitude	latitude of the initial view state
longitude	longitude of the initial view state
zoom	zoom of the initial view state
pitch	pitch of the initial view state
bearing	bearing of the initial view state
initialViewState	initial view state, if set, other view state arguments (longitude, latitude etc.) are ignored
views	a single View, or an array of View instances, if not supplied, a single MapView will be created
width	width of the widget
height	height of the widget
elementId	explicit element id (usually not needed)
...	optional properties passed to the deck instance

Value

deckgl widget

See Also

<https://deck.gl/#/documentation/deckgl-api-reference/deck>

 deckgl-shiny

Shiny bindings for deckgl

Description

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

Usage

```
deckglOutput(outputId, width = "100%", height = "400px")
```

```
renderDeckgl(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 deckgl
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.

default_icon_properties

Default icon properties

Description

Returns icon properties with default values for iconAtlas, iconMapping and getIcon, so that the default icon is used.

Usage

```
default_icon_properties(sizeScale = 15, getSize = 5,  
  getColor = c(240, 140, 0))
```

Arguments

sizeScale	icon size multiplier
getSize	height of each object (in pixels), if a number is provided, it is used as the size for all objects, if a function is provided, it is called on each object to retrieve its size
getColor	rgba color of each object, if an array is provided, it is used as the color for all objects if a function is provided, it is called on each object to retrieve its color

does_it_work	<i>Check if everything works fine</i>
--------------	---------------------------------------

Description

Check if everything works fine

Usage

```
does_it_work(token = NULL)
```

Arguments

token	mapbox api access token
-------	-------------------------

encode_icon_atlas	<i>Encode atlas image to base64</i>
-------------------	-------------------------------------

Description

Encode atlas image to base64

Usage

```
encode_icon_atlas(filename = NULL)
```

Arguments

filename	filename of atlas image
----------	-------------------------

Value

base64 encoded atlas image

`get_color_to_rgb_array`*Create a getColor data accessor*

Description

Creates a JS method to retrieve the color of each object. The method parses the HEX color property of the data object to an rgb color array.

Usage

```
get_color_to_rgb_array(color_property)
```

Arguments

`color_property` property name of data object containing the HEX color

Value

JavaScript code evaluated on the client-side

`get_data`*Get data*

Description

EXPERIMENTAL, usually used in conjunction with [add_data](#)

Usage

```
get_data(var_name = "thanksForAllTheFish")
```

Arguments

`var_name` JavaScript variable name

get_position	<i>Create a getPosition data accessor</i>
--------------	---

Description

Creates a JS method to retrieve the position of each object.

Usage

```
get_position(latitude = NULL, longitude = NULL, coordinates = NULL)
```

Arguments

latitude	latitude property of data object
longitude	longitude property of data object
coordinates	coordinates property of data object (in this case latitude and longitude parameters are ignored)

Value

JavaScript code evaluated on the client-side

get_property	<i>Create a data accessor</i>
--------------	-------------------------------

Description

Creates a JS method to retrieve a given property of each object.

Usage

```
get_property(property_name)
```

Arguments

property_name	property name of data object
---------------	------------------------------

Value

JavaScript code evaluated on the client-side

icon_definition	<i>Icon definition on an atlas image</i>
-----------------	--

Description

Icon definition on an atlas image

Usage

```
icon_definition(x = 0, y = 0, width = 128, height = 128,
  anchorX = (width/2), anchorY = 128, mask = TRUE)
```

Arguments

x	x position of icon on the atlas image
y	y position of icon on the atlas image
width	width of icon on the atlas image
height	height of icon on the atlas image
anchorX	horizontal position of icon anchor
anchorY	vertical position of icon anchor
mask	whether icon is treated as a transparency mask, if TRUE, user defined color is applied, if FALSE, pixel color from the image is applied

sf_bike_parking	<i>sf bike parking</i>
-----------------	------------------------

Description

sf bike parking

Usage

```
sf_bike_parking
```

Format

tibble with 2520 rows and 5 variables:

address address
racks number of racks
spaces number of spaces
lng longitude
lat latitude

Source

<https://raw.githubusercontent.com/uber-common/deck.gl-data/master/website/sf-bike-parking.json>

Index

*Topic **datasets**

- bart_segments, [19](#)
- bart_stations, [20](#)
- sf_bike_parking, [26](#)

- add_arc_layer, [2](#)
- add_contour_layer, [3](#)
- add_data, [5](#), [24](#)
- add_geojson_layer, [6](#)
- add_grid_layer, [7](#)
- add_hexagon_layer, [8](#)
- add_icon_layer, [9](#)
- add_layer, [10](#)
- add_line_layer, [11](#)
- add_mapbox_basemap, [12](#)
- add_path_layer, [13](#)
- add_point_cloud_layer, [14](#)
- add_polygon_layer, [15](#)
- add_scatterplot_layer, [16](#)
- add_screen_grid_layer, [17](#)
- add_text_layer, [18](#)

- bart_segments, [19](#)
- bart_stations, [20](#)

- contour_definition, [20](#)

- deckgl, [21](#)
- deckgl-shiny, [22](#)
- deckglOutput (deckgl-shiny), [22](#)
- default_icon_properties, [22](#)
- does_it_work, [23](#)

- encode_icon_atlas, [23](#)

- get_color_to_rgb_array, [24](#)
- get_data, [24](#)
- get_position, [25](#)
- get_property, [25](#)

- icon_definition, [26](#)

- renderDeckgl (deckgl-shiny), [22](#)

- sf_bike_parking, [26](#)