Package ‘deckgl’

March 16, 2020

Title  An R Interface to 'deck.gl'
Version 0.2.5
Date 2020-03-03
Maintainer Stefan Kuethe <crazycapivara@gmail.com>


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 6.1.1

Suggests knitr, rmarkdown, testthat, rprojroot, sf, scales

VignetteBuilder knitr

NeedsCompilation no

Author Stefan Kuethe [aut, cre]

Repository CRAN

Date/Publication 2020-03-16 18:20:02 UTC
R topics documented:

add_arc_layer .................................................. 3
add_basemap ..................................................... 4
add_bitmap_layer ............................................... 4
add_column_layer ............................................... 5
add_contour_layer .............................................. 6
add_data ........................................................ 7
add_geojson_layer .............................................. 8
add_great_circle_layer ....................................... 9
add_grid_cell_layer .......................................... 10
add_grid_layer ................................................. 11
add_h3_cluster_layer ....................................... 12
add_h3_hexagon_layer ....................................... 13
add_heatmap_layer ........................................... 14
add_hexagon_layer ............................................ 15
add_icon_layer ................................................ 16
add_layer ......................................................... 17
add_line_layer ................................................. 18
add_mapbox_basemap ......................................... 19
add_path_layer ................................................ 20
add_point_cloud_layer ...................................... 21
add_polygon_layer ........................................... 22
add_raster_tile_layer ........................................ 23
add_scatterplot_layer ....................................... 24
add_screen_grid_layer ...................................... 25
add_text_layer ................................................. 26
bart_segments .................................................. 27
bart_stations ................................................... 28
deckgl .......................................................... 29
deckgl-shiny .................................................... 30
deckgl_proxy ..................................................... 30
does_it_work .................................................... 31
encode_icon_atlas ............................................. 31
color_to_rgb_array ........................................... 32
color_to_rgb_array ........................................... 32
get_first_element ............................................ 33
get_last_element .............................................. 33
get_position .................................................... 34
get_property ..................................................... 34
sf_bike_parking ............................................... 35
update_deckgl .................................................. 35
use_carto_style ............................................... 36
use_contour_definition ..................................... 36
use_default_icon_properties ............................... 37
use_icon_definition .......................................... 37
Index .......................... 38
**add_arc_layer**

*Add an arc layer to the deckgl widget*

**Description**

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

**Usage**

```r
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 `getTooltip` property (callback) showing a tooltip when the mouse enters an object, e.g.:

  ```javascript
  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/arc-layer](https://deck.gl/#/documentation/deckgl-api-reference/layers/arc-layer)

**Examples**

```r
## @knitr arc-layer
data("bart_segments")

properties <- list(
  pickable = TRUE,
  getWidth = 12,
  getSourcePosition = ~from_lng + from_lat,
  getTargetPosition = ~to_lng + to_lat,
  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)""")
)

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

if (interactive()) deck
```
add_basemap

Description
Add a basemap to the deckgl widget

Usage
add_basemap(deckgl, style = use_carto_style(), ...)

Arguments
- deckgl: deckgl widget
- style: The style definition of the map conforming to the Mapbox Style Specification.
- ...: not used

add_bitmap_layer

Description
Add a bitmap layer to the deckgl widget

Usage
add_bitmap_layer(deckgl, id = "h3-hexagon-layer", image = NULL,
properties = list(), ...)

Arguments
- deckgl: deckgl widget
- id: id of the layer
- image: image
- 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
**add_column_layer**

Examples

```r
image <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/master/",
  "website/sf-districts.png"
)
bounds <- c(-122.5190, 37.7045, -122.355, 37.829)

deck <- deckgl() %>%
  add_bitmap_layer(image = image, bounds = bounds) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

---

**add_column_layer**  
*Add a column layer to the deckgl widget*

Description

The ColumnLayer can be used to render a heatmap of vertical cylinders. It renders a tesselated regular polygon centered at each given position (a "disk"), and extrude it in 3d.

Usage

```r
add_column_layer(deckgl, id = "column-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](https://deck.gl/#/documentation/deckgl-api-reference/layers/column-layer) 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

## @knitr column-layer

```r
hexagon_centroids <- system.file("sample-data/centroids.csv", package = "deckgl") %>%
  read.csv()

deck <- deckgl(zoom = 11, pitch = 35) %>%
  add_column_layer(  
data = hexagon_centroids,  
diskResolution = 12,  
getPosition = ~lng + lat,  
getElevation = ~value,  
getFillColor = JS("d => [48, 128, d.value * 255, 255]"),  
elevationScale = 5000,  
radius = 250,  
extruded = TRUE,  
getTooltip = JS("object => 'height: $(object.value * 5000)m'")  
)%>%
  add_mapbox_basemap()

if (interactive()) deck
```

### Description

The `ContourLayer` renders contour lines for a given threshold and cell size. Internally it implements the **Marching Squares** algorithm to generate contour line segments and feeds them into `LineLayer` to render lines.

### Usage

```r
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 `getTooltip` property (callback) showing a tooltip when the mouse enters an object, e.g.

  ```js
  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/contour-layer

Examples

```r
## @knitr contour-layer
data("sf_bike_parking")

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

properties <- list(
  contours = contours,
  cellSize = 200,
  elevationScale = 4,
  getPosition = ~lng + lat
)

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

if (interactive()) deck
```

---

**Description**

EXPERIMENTAL

**Usage**

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

Arguments

dekgl 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")
)

dek <- dekgl(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(dekgl, id = "geojson-layer", data = NULL,
  properties = list(), ...)

Arguments

dekgl 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

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

dec <- 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_great_circle_layer**

*Add a great circle layer to the deckgl widget*

**Description**

The `GreatCircleLayer` is a variation of the `ArcLayer`. It renders flat arcs along the great circle joining pairs of source and target points, specified as latitude/longitude coordinates.

**Usage**

```r
add_great_circle_layer(deckgl, id = "great-circle-layer", data = NULL,
                       properties = list(), ...)
```

**Arguments**

- `deckgl` : deckgl widget
- `id` : id of the layer
add_grid_cell_layer

Description

The GridCellLayer can render a grid-based heatmap. It is a variation of the ColumnLayer. It takes the constant width / height of all cells and top-left coordinate of each cell. The grid cells can be given a height using the getElevation accessor.

Usage

add_grid_cell_layer(deckgl, id = "grid-cell-layer", data = NULL, properties = list(), ...)
**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**

```r
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**


**Examples**

```r
hexagon_centroids <- system.file("sample-data/centroids.csv", package = "deckgl") %>% read.csv()

data <- hexagon_centroids,
getPosition = ~lng + lat,
getElevation = ~value,
getFillColor = JS("d => ![48, 128, d.value * 255, 255]"),
elevationScale = 5000,
cellSize = 250,
extruded = TRUE,
getTooltip = JS("object => 'height: ${{object.value * 5000}}m'")
```

```r
if (interactive()) deck
```
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](https://deck.gl/#/documentation/deckgl-api-reference/layers/grid-layer) 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


Examples

```r
## @knitr grid-layer
data("sf_bike_parking")

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

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

if (interactive()) deck
```

---

**add_h3_cluster_layer**  
Add a h3 cluster layer to the deckgl widget

Description

Add a h3 cluster layer to the deckgl widget

Usage

```r
add_h3_cluster_layer(deckgl, id = "h3-cluster-layer", data = NULL, properties = list(), ...)
```
**add_h3_hexagon_layer**

Add a h3 hexagon layer to the deckgl widget

---

**Description**

Add a h3 hexagon layer to the deckgl widget

---

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deckgl</td>
<td>deckgl widget</td>
</tr>
<tr>
<td>id</td>
<td>id of the layer</td>
</tr>
<tr>
<td>data</td>
<td>url to fetch data from or data object</td>
</tr>
<tr>
<td>properties</td>
<td>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. <code>getTooltip = JS(&quot;object =&gt; object.name&quot;)</code></td>
</tr>
</tbody>
</table>

---

**See Also**

https://deck.gl/#/documentation/deckgl-api-reference/layers/h3-cluster-layer

---

**Examples**

```r
## @knitr h3-cluster-layer
data_url <- paste0("https://raw.githubusercontent.com/uber-common/deck.gl-data/master/website/sf.h3clusters.json")
sample_data <- data_url
properties <- list(stroked = TRUE,
                    filled = TRUE, 
                    extruded = FALSE,
                    getHexagons = ~hexIds,
                    getFillColor = JS("d => [255, (1 - d.mean / 500) * 255, 0]"),
                    getLineColor = c(255, 255, 255),
                    lineWidthMinPixels = 2,
                    getTooltip = ~mean)
deck <- deckgl(zoom = 10.5, pitch = 20) %>%
  add_h3_cluster_layer(data = sample_data, properties = properties) %>%
  add_mapbox_basemap()
if (interactive()) deck
```
add_heatmap_layer

Usage

add_h3_hexagon_layer(deckgl, id = "h3-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/h3-hexagon-layer

Examples

```r
## @knitr h3-hexagon-layer-layer
h3_cells <- system.file("sample-data/h3-cells.csv", package = "deckgl") %>%
  read.csv()

properties <- list(
  getHexagon = ~h3_index,
  getFillColor = JS("d => [255, (1 - d.count / 500) * 255, 0]"),
  getElevation = ~count,
  elevationScale = 20,
  getTooltip = JS("object => '/grave.Var
  ${object.h3_index}: ${object.count}'/grave.Var")
)

dec <- deckgl(zoom = 11, pitch = 35) %>%
  add_h3_hexagon_layer(data = h3_cells, properties = properties) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

Description

The HeatmapLayer can be used to visualize spatial distribution of data. It internally implements Gaussian Kernel Density Estimation to render heatmaps.
add_hexagon_layer

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


Examples

```r
## @knitr heatmap-layer
data("sf_bike_parking")

map <- deckgl() %>%
  add_hexagon_layer(
    data = sf_bike_parking,
    getPosition = ~lng + lat,
    getWeight = ~spaces
  ) %>%
  add_mapbox_basemap()

if (interactive()) map
```

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(), ...)
```
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 = use_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**

```r
## @knitr icon-layer
data("bart_stations")

properties <- list(
  pickable = TRUE,
  iconAtlas = encode_icon_atlas(),
  iconMapping = list(marker = use_icon_definition()),
  sizeScale = 10,
  getPosition = ~lng + lat,
  getIcon = JS("d => 'marker'")
)

dock <- deckgl(zoom = 10, pitch = 45) %>%
  add_icon_layer(data = bart_stations, properties = properties) %>%
  add_basemap()

if (interactive()) dock
```

---

**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.
add_line_layer

Usage

add_line_layer(deckgl, id = "line-layer", data = NULL,
               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  Add a line layer to the deckgl widget

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

```r
## @knitr line-layer
data("bart_segments")

properties <- list(
pickable = TRUE,
getWidth = 12,
getSourcePosition = ~from_lng + from_lat,
getTargetPosition = ~to_lng + to_lat,
getColor = JS("d => [Math.sqrt(d.inbound + d.outbound), 140, 0]")
   
getTooltip = JS("object => '${object.from_name} to ${object.to_name}'")
   
)

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

if (interactive()) deck
```

Description

Add a base map from mapbox to the deckgl widget

Usage

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

Arguments

- **deckgl**: deckgl widget
- **style**: map style
- **token**: mapbox API access token

Value

deckgl widget
**add_path_layer**

*Add a path layer to the deckgl widget*

### Description

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

### Usage

```r
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


### Examples

```r
## @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")
)

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

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

decogl  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

```r
sample_data <- paste0("https://raw.githubusercontent.com/uber-common/deck.gl-data/master/website/pointcloud.json")

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

```r
getPosition = ~position,
getNormal = ~normal,
getColor = ~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
```

## Description

The **PolygonLayer** renders filled and/or stroked polygons.

## Usage

```r
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


## Examples

```r
## @knitr polygon-layer
sample_data <- paste0(
  "https://raw.githubusercontent.com/",
  "uber-common/deck.gl-data/",
  "master/website/sf-2016-01-01.json"
)
```
add_raster_tile_layer

```
properties <- list(
  pickable = TRUE,
  stroked = TRUE,
  filled = TRUE,
  wireframe = TRUE,
  lineWidthMinPixels = 1,
  getPolygon = ~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_raster_tile_layer**  Add a raster tile layer to the deckgl widget

**Description**

EXPERIMENTAL, see https://deck.gl/#/examples/core-layers/tile-layer

**Usage**

```
add_raster_tile_layer(deckgl, id = "raster-tiles",
  tileServer = "https://c.tile.openstreetmap.org/",
  properties = list(), ...)
```

**Arguments**

- **deckgl**  deckgl widget
- **id**  id of the layer
- **tileServer**  base url of the tile server
- **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
add_scatterplot_layer

Examples

```r
### @knitr raster-tile-layer
tile_servers <- list(
  osm = "https://a.tile.openstreetmap.org/",
  carto_light = "https://cartodb-basemaps-a.global.ssl.fastly.net/light_all/",
  carto_dark = "https://cartodb-basemaps-a.global.ssl.fastly.net/dark_all/",
  stamen_toner = "http://a.tile.stamen.com/toner/"
)

dock <- deckgl() %>%
  add_raster_tile_layer(
    tileServer = tile_servers$osm,
    pickable = TRUE,
    autoHighlight = TRUE,
    highlightColor = c(60, 60, 60, 40)
  )

if (interactive()) deck
```

**add_scatterplot_layer**  
*Add a scatter 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

```r
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](https://deck.gl/#/documentation/deckgl-api-reference/layers/scatterplot-layer) 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")`

See Also

add_screen_grid_layer

Examples

```r
## @knitr scatterplot-layer
data("bart_stations")

properties <- list(
  getPosition = ~lng + lat,
  getRadius = JS("data => Math.sqrt(data.exits)"),
  radiusScale = 6,
  getFillColor = c(255, 140, 20),
  getTooltip = ~name
)

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

if (interactive()) deck
```

---

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

```r
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

Examples

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

color_to_rgb <- function(color) col2rgb(color) %>% as.vector()
color_range <- c("#0198BD", "#49E3CE", "#D8FEB5", "#FEEDB1", "#FEAD54", "#D1374E") %>%
  lapply(color_to_rgb)

properties <- list(
  opacity = 0.8,
  cellSizePixels = 50,
  colorRange = color_range,
  getPosition = ~COORDINATES,
  getWeight = ~SPACES
)

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

# Use a data object instead of the data url above
data("sf_bike_parking")

deck <- deckgl() %>%
  add_screen_grid_layer(
    data = sf_bike_parking,
    properties = properties,
    # overwrite properties
    getPosition = ~lng + lat,
    getWeight = ~spaces
  ) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

---

### add_text_layer

Add a text layer to the deckgl widget

**Description**

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

**Usage**

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

#### bart segments

**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**

```r
## @knitr text-layer
data("bart_stations")

dez <- deckgl(zoom = 10, pitch = 35) %>%
  add_text_layer(
    data = bart_stations,
    pickable = TRUE,
    getPosition = ~lng + lat,
    getText = ~name,
    getSize = 15,
    getAngle = 0,
    getTextAnchor = "middle",
    getAlignmentBaseline = "center",
    getTooltip = JS("object => object.name<br/>object.address")
  ) %>%
  add_mapbox_basemap()

if (interactive()) deck
```

---

**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


bart_stations  bart stations

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

Create a deckgl widget

Description
Create a deckgl widget

Usage
deckgl(latitude = 37.8, longitude = -122.45, zoom = 12, pitch = 0, bearing = 0, initialViewState = 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
dekgl widget

See Also
https://deck.gl/#/documentation/deckgl-api-reference/deck
deckgl-proxy

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.

deckgl-proxy
Create a deckgl proxy object

Description
Creates a deckgl-like object that can be used to update a deckgl object that has already been rendered.

Usage
- `deckgl_proxy(shinyId, session = shiny::getDefaultReactiveDomain())`

Arguments
- `shinyId`: single-element character vector indicating the output ID of the deck to modify
- `session`: the Shiny session object to which the deckgl widget belongs; usually the default value will suffice.
**does_it_work**

*Check if everything works fine*

**Description**

Check if everything works fine

**Usage**

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

**Arguments**

- **token**: mapbox API access token

**encode_icon_atlas**

*Encode atlas image to base64*

**Description**

Encode atlas image to base64

**Usage**

```r
code_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_first_element**  
*Create a data accessor retrieving the first element of an array*

**Description**
Create a data accessor retrieving the first element of an array

**Usage**

```javascript
get_first_element(property_name)
```

**Arguments**

- `property_name` property name of data object

**Value**
JavaScript code evaluated on the client-side

---

**get_last_element**  
*Create a data accessor retrieving the last element of an array*

**Description**
Create a data accessor retrieving the last element of an array

**Usage**

```javascript
get_last_element(property_name)
```

**Arguments**

- `property_name` property name of data object

**Value**
JavaScript code evaluated on the client-side
get_position

Create a getPosition data accessor

Description

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

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>latitude</td>
<td>latitude property of data object</td>
</tr>
<tr>
<td>longitude</td>
<td>longitude property of data object</td>
</tr>
<tr>
<td>coordinates</td>
<td>coordinates property of data object (in this case latitude and longitude parameters are ignored)</td>
</tr>
</tbody>
</table>

Value

JavaScript code evaluated on the client-side

get_property

Create a data accessor

Description

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

Usage

get_property(property_name)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property_name</td>
<td>property name of data object</td>
</tr>
</tbody>
</table>

Value

JavaScript code evaluated on the client-side
sf_bike_parking

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


update_deckgl

Send commands to a deckgl instance in a Shiny app

Description

Send commands to a deckgl instance in a Shiny app

Usage

update_deckgl(proxy, ...)

Arguments

- **proxy** deckgl proxy object
- **...** unused

See Also

deskgl_proxy
use_carto_style  
*Use a Carto style*

---

### Description

Use a Carto style

### Usage

```r
use_carto_style(theme = "dark-matter")
```

### Arguments

- **theme**  
The theme of the style, `dark-matter`, `positron` or `voyager`.

---

use_contour_definition

*Use contour definition*

---

### Description

Use contour definition

### Usage

```r
use_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
**use_default_icon_properties**

*Use default icon properties*

**Description**

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

**Usage**

```r
use_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

**use_icon_definition**

*Icon definition on an atlas image*

**Description**

Icon definition on an atlas image

**Usage**

```r
use_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
Index

**Topic datasets**

- bart_segments, 27
- bart_stations, 28
- sf_bike_parking, 35

add_arc_layer, 3
add_basemap, 4
add_bitmap_layer, 4
add_column_layer, 5
add_contour_layer, 6
add_data, 7, 32
add_geojson_layer, 8
add_great_circle_layer, 9
add_grid_cell_layer, 10
add_grid_layer, 11
add_h3_cluster_layer, 12
add_h3_hexagon_layer, 13
add_heatmap_layer, 14
add_hexagon_layer, 15
add_icon_layer, 16
add_layer, 17
add_line_layer, 18
add_mapbox_basemap, 19
add_path_layer, 20
add_point_cloud_layer, 21
add_polygon_layer, 22
add_raster_tile_layer, 23
add_scatterplot_layer, 24
add_screen_grid_layer, 25
add_text_layer, 26

bart_segments, 27
bart_stations, 28

derendDeckgl(deckgl-shiny), 30

get_color_to_rgb_array, 32
get_data, 32
get_first_element, 33
get_last_element, 33
get_position, 34
get_property, 34

renderDeckgl(deckgl-shiny), 30

sf_bike_parking, 35

update_deckgl, 35

use_carto_style, 36

use_contour_definition, 36

use_default_icon_properties, 37

use_icon_definition, 37

encode_icon_atlas, 31