Package ‘deckgl’

February 19, 2023

Title   An R Interface to 'deck.gl'
Version 0.3.0
Date    2023-02-19
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
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 7.2.0
Suggests knitr, markdown, testthat, rprojroot, sf, scales,
       RColorBrewer, shiny
VignetteBuilder knitr
NeedsCompilation no
Author Stefan Kuethe [aut, cre]
Repository CRAN
Date/Publication 2023-02-19 20:30:02 UTC
R topics documented:

- add_arc_layer
- add_basemap
- add_bitmap_layer
- add_column_layer
- add_contour_layer
- add_control
- add_data
- add_geojson_layer
- add_great_circle_layer
- add_grid_cell_layer
- add_grid_layer
- add_h3_cluster_layer
- add_h3_hexagon_layer
- add_heatmap_layer
- add_hexagon_layer
- add_icon_layer
- add_json_editor
- add_layer
- add_legend
- add_legend_pal
- add_line_layer
- add_mapbox_basemap
- add_path_layer
- add_point_cloud_layer
- add_polygon_layer
- add_raster_tile_layer
- add_scatterplot_layer
- add_screen_grid_layer
- add_source
- add_source_as_dep
- add_text_layer
- bart_segments
- bart_stations
- deckgl
- deckgl-shiny
- deckgl_proxy
- does_it_work
- encode_icon_atlas
- get_color_to_rgb_array
- get_data
- get_first_element
- get_last_element
- get_position
- get_property
- set_view_state
- sf_bike_parking
Description

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

Usage

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

Arguments

- `deckgl`: A deckgl widget object.
- `data`: The url to fetch data from or a data object.
- `properties`: A named list of properties with names corresponding to the properties defined in the `deckgl-api-reference` for the given layer class. The `properties` parameter can also be an empty list. In this case all props must be passed as named arguments.
- `...`: Named arguments that will be added to the `properties` object. Identical parameters are overwritten.
- `id`: The unique id of the 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
data("bart_segments")

properties <- list(
  getWidth = 12,
  getSourcePosition = ~from_lng + from_lat,
  getTargetPosition = ~to_lng + to_lat,
  getSourceColor = "@=[Math.sqrt(inbound), 140, 0]",
  getTargetColor = "@=[Math.sqrt(outbound), 140, 0]",
  tooltip = use_tooltip(
```
add_bitmap_layer

...
add_column_layer

Arguments

decogl A deckgl widget object.
image image
properties A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.

... Named arguments that will be added to the properties object. Identical parameters are overwritten.
id The unique id of the layer.

Examples

decogl <- deckgl() %>%
  add_bitmap_layer(image = image, bounds = bounds) %>%
  add_basemap()

if (interactive()) deck

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

add_column_layer(
  deckgl,
  data = NULL,
  properties = list(),
  ..., .
  id = "column-layer"
)
Arguments

- `deckgl` A deckgl widget object.
- `data` The url to fetch data from or a data object.
- `properties` A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- `...` Named arguments that will be added to the properties object. Identical parameters are overwritten.
- `id` The unique id of the layer.

See Also

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

Examples

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

duck <- deckgl(zoom = 11, pitch = 35) %>%
  add_column_layer(
    data = hexagon_centroids,
    diskResolution = 12,
    getPosition = ~lng + lat,
    getElevation = ~value,
    getFillColor = "@=[48, 128, value * 255, 255]",
    elevationScale = 5000,
    radius = 250,
    extruded = TRUE,
    tooltip = "Value: {{value}}"
  ) %>%
  add_control("Column Layer", "bottom-left") %>%
  add_basemap()

if (interactive()) duck
```

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.
add_contour_layer

Usage

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

Arguments

deckgl  A deckgl widget object.
data    The url to fetch data from or a data object.
properties  A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
...  Named arguments that will be added to the properties object. Identical parameters are overwritten.
id  The unique id of the layer.

See Also

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

Examples

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

contours <- list(
  use_contour_definition(
    threshold = 1,
    color = c(255, 0, 0),
    stroke_width = 2
  ),
  use_contour_definition(
    threshold = 5,
    color = c(0, 255, 0),
    stroke_width = 3
  ),
  use_contour_definition(
    threshold = 15,
    color = c(0, 0, 255),
    stroke_width = 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_control("Contour Layer") %>%
add_basemap()

if (interactive()) deck

---

**add_control**

*Add a control to the widget*

**Description**

Add a control to the widget.

**Usage**

`add_control(deckgl, html, pos = "top-right", style = NULL)`

**Arguments**

- `deckgl` A deckgl widget object.
- `html` The `innerHTML` of the element.
- `pos` The position of the control. Possible values are `top-left`, `top-right`, `bottom-right` and `bottom-left`.
- `style` A `cssText` string that will modify the default style of the element.

**Examples**

```r
deck <- deckgl() %>%
add_basemap() %>%
add_control(
  "<h1>Blank Base Map</h1>",
pos = "top-right",
style = "background: #004080; color: white;"
)

if (interactive()) deck
```
add_data  

Description

EXPERIMENTAL

Usage

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

Arguments

deggl  
deckgl widget

data  
data object

var_name  
JavaScript variable name used to make the data available

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

Arguments

deggl  
A deckgl widget object.

data  
The url to fetch data from or a data object.

properties  
A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.

...  
Named arguments that will be added to the properties object. Identical parameters are overwritten.

id  
The unique id of the layer.
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,
  data = NULL,
  properties = list(),
  ...,
  id = "great-circle-layer"
)
```
add_grid_cell_layer

Arguments

- **deckgl**
  A deckgl widget object.

- **data**
  The url to fetch data from or a data object.

- **properties**
  A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.

- **...**
  Named arguments that will be added to the properties object. Identical parameters are overwritten.

- **id**
  The unique id of the layer.

See Also

https://deck.gl/#/documentation/deckgl-api-reference/layers/great-circle-layer

Examples

```r
## @knitr great-circle-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 = "{{from_name}} to {{to_name}}"
)

deck <- deckgl(zoom = 10, pitch = 35) %>%
  add_great_circle_layer(data = bart_segments, properties = properties) %>%
  add_control("Great Circle Layer") %>%
  add_basemap()

if (interactive()) deck
```

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.
add_grid_cell_layer

Usage

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

Arguments

deckgl     A deckgl widget object.
data       The url to fetch data from or a data object.
properties A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
...       Named arguments that will be added to the properties object. Identical parameters are overwritten.
id       The unique id of the layer.

See Also


Examples

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

deck <- deckgl(zoom = 11, pitch = 35) %>%
  add_grid_cell_layer(
    data = hexagon_centroids,
    getPosition = ~lng + lat,
    getElevation = ~value,
    getFillColor = "@=[48, 128, value * 255, 255]",
    elevationScale = 5000,
    cellSize = 250,
    extruded = TRUE,
    tooltip = "{{value}}"
  ) %>%
  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,
  data = NULL,
  properties = list(),
  ...,
  id = "grid-layer"
)

Arguments

deqkgl  A deckgl widget object.
data  The url to fetch data from or a data object.
properties  A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
...  Named arguments that will be added to the properties object. Identical parameters are overwritten.
id  The unique id of the layer.

See Also

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

Examples

data("sf_bike_parking")

properties <- list(
  filter = "spaces > 4",
  visible = TRUE,
  extruded = TRUE,
  cellSize = 200,
  elevationScale = 4,
  getPosition = "@=[lng, lat]", # = lng + lat,
  colorRange = RColorBrewer::brewer.pal(6, "Y1OrRd"),
deck <- deckgl(zoom = 11, pitch = 45, bearing = 35, element_id = "grid-layer") %>%
  add_source("sf-bike-parking", sf_bike_parking) %>%
  add_grid_layer(
    source = "sf-bike-parking",
    properties = properties
  ) %>%
  add_control("Grid Layer") %>%
  add_basemap() %>%
  add_json_editor(wrap = 50, maxLines = 23)

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,
  data = NULL,
  properties = list(),
  ...,  
  id = "h3-cluster-layer"
)
```

**Arguments**

- **deckgl**: A deckgl widget object.
- **data**: The url to fetch data from or a data object.
- **properties**: A named list of properties with names corresponding to the properties defined in the *deckgl-api-reference* for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- **...**: Named arguments that will be added to the properties object. Identical parameters are overwritten.
- **id**: The unique id of the layer.

**See Also**

https://deck.gl/#/documentation/deckgl-api-reference/layers/h3-cluster-layer
## add_h3_hexagon_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 <- jsonlite::fromJSON(data_url, simplifyDataFrame = FALSE)
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
)

deed <- deckgl(zoom = 10.5, pitch = 20) %>%
  add_h3_cluster_layer(data = sample_data, properties = properties) %>%
  add_basemap()

if (interactive()) deed
```

---

### add_h3_hexagon_layer

Add a h3 hexagon layer to the deckgl widget

#### Description

Add a h3 hexagon layer to the deckgl widget

#### Usage

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

#### Arguments

- **deckgl**: A deckgl widget object.
- **data**: The url to fetch data from or a data object.
add_heatmap_layer

properties A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.

Named arguments that will be added to the properties object. Identical parameters are overwritten.

id The unique id of the 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 = "{{h3_index}}: {{count}}"
)

dek <- deckgl(zoom = 11, pitch = 35) %>%
  add_h3_hexagon_layer(data = h3_cells, properties = properties) %>%
  add_control("H3 Hexagon Layer") %>%
  add_basemap()

if (interactive()) dek
```

Description

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

Usage

```r
add_heatmap_layer(
  deckgl,
  id = "heatmap-layer",
  data = NULL,
  properties = list(),
  ...
)
```
**add_hexagon_layer**

**Arguments**

- **deckgl**
  A deckgl widget object.
- **id**
  The unique id of the layer.
- **data**
  The url to fetch data from or a data object.
- **properties**
  A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- **...**
  Named arguments that will be added to the properties object. Identical parameters are overwritten.

**See Also**

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

**Examples**

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

map <- deckgl() %>%
  add_heatmap_layer(
    data = sf_bike_parking,
    getPosition = ~lng + lat,
    getWeight = ~spaces
  ) %>%
  add_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**

```r
add_hexagon_layer(
  deckgl,
  data = NULL,
  properties = list(),
  ...
)
```
Arguments

- **deckgl**: A deckgl widget object.
- **data**: The url to fetch data from or a data object.
- **properties**: A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- **...**: Named arguments that will be added to the properties object. Identical parameters are overwritten.
- **id**: The unique id of the layer.

See Also

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

Examples

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

properties <- list(
  extruded = TRUE,
  radius = 200,
  elevationScale = 4,
  getPosition = ~lng + lat,
  colorRange = RColorBrewer::brewer.pal(6, "Oranges"),
  tooltip = "
    <p>{{position.0}}, {{position.1}}</p>
    <p>Count: {{points.length}}</p>
    <p>{{#points}}<div>{{address}}</div>{{/points}}</p>
  ",
  onClick = JS("obj => console.log(obj)"),
  autoHighlight = TRUE
)

deqk <- deckgl(zoom = 11, pitch = 45, bearing = 35) %>%
  add_hexagon_layer(data = sf_bike_parking, properties = properties) %>%
  add_control("Hexagon Layer", "top-left") %>%
  add_basemap()

if (interactive()) deck
```

add_icon_layer

*Add an icon layer to the deckgl widget*

Description

The IconLayer renders raster icons at given coordinates.
add_icon_layer

Usage

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

Arguments

deckgl A deckgl widget object.
data The url to fetch data from or a data object.
properties A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
... Named arguments that will be added to the properties object. Identical parameters are overwritten.
id The unique id of the layer.

See Also

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

Examples

## @knitr icon-layer

data("bart_stations")

properties <- list(
  iconAtlas = encode_icon_atlas(),
  iconMapping = list(marker = use_icon_definition()),
  sizeScale = 10,
  getPosition = ~lng + lat,
  getIcon = JS("d => 'marker'"),
  getSize = 5,
  getColor = JS("d => [Math.sqrt(d.exits), 140, 0]"),
  getTooltip = "{{name}}<br/>{{address}}"
)

deck <- deckgl(zoom = 10, pitch = 45) %>%
  add_icon_layer(data = bart_stations, properties = properties) %>%
  add_control("Icon Layer") %>%
  add_basemap()

if (interactive()) deck
add_json_editor  
Add a JSON-editor to the deckgl widget

Description

Adds a Ace-editor in JSON mode to the map to interact with the layers of your deck instance.

Usage

add_json_editor(deckgl, ..., style = "width: 40%;", theme = "idle_fingers")

Arguments

deqgl  A deckgl widget object.
...  Optional args that are passed to the editor. See https://github.com/ajaxorg/ace/wiki/Configuring-Ace for a list of available options.
style  A cssText string that will modify the default style of the container that holds the editor.
theme  The name of the theme used by the editor.

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,
  data = NULL,
  properties = list(),
  ...,
  id = "hopeful-hopper",
  tooltip = NULL,
  source = NULL,
  filter = NULL
)

add_legend

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deckgl</td>
<td>A deckgl widget object.</td>
</tr>
<tr>
<td>class_name</td>
<td>The name of the JavaScript layer class, e. g. ScatterplotLayer.</td>
</tr>
<tr>
<td>data</td>
<td>The url to fetch data from or a data object.</td>
</tr>
<tr>
<td>properties</td>
<td>A named list of properties with names corresponding to the properties defined</td>
</tr>
<tr>
<td></td>
<td>in the <code>deckgl-api-reference</code> for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.</td>
</tr>
<tr>
<td>...</td>
<td>Named arguments that will be added to the <code>properties</code> object. Identical parameters are overwritten.</td>
</tr>
<tr>
<td>id</td>
<td>The unique id of the layer.</td>
</tr>
<tr>
<td>tooltip</td>
<td>A tooltip template that defines what should be displayed when the mouse enters an object. You can also pass a list with the properties <code>html</code> and <code>style</code>. See also <code>use_tooltip</code>.</td>
</tr>
<tr>
<td>source</td>
<td>The ID of the data source. See <code>add_source</code>.</td>
</tr>
<tr>
<td>filter</td>
<td>A filter expression that is applied to the data object.</td>
</tr>
</tbody>
</table>

Value

A deckgl widget object.

Description

Add a legend to the deckgl widget

Usage

```r
add_legend(
  deckgl, colors, labels,
  title = NULL,
  pos = "top-right",
  style = NULL,
  ...
)
```
Arguments

- **deckgl**: A deckgl widget object.
- **colors**: The colors of the legend items.
- **labels**: The labels corresponding to the colors of the legend items.
- **title**: The title of the legend.
- **pos**: The position of the control. Possible values are `top-left`, `top-right`, `bottom-right` and `bottom-left`.
- **style**: A `cssText` string that will modify the default style of the element.
- **...**: not used

---

**add_legend_pal**

Add a legend to the deckgl widget using a palette func

**Description**

Add a legend to the deckgl widget using a palette func

**Usage**

```r
add_legend_pal(deckgl, pal, ...)
```

**Arguments**

- **deckgl**: A deckgl widget object.
- **pal**: A palette function that is used to create the legend elements (colors and labels) automatically.
- **...**: Parameters that are passed to `add_legend`.

**See Also**

`col_numeric` et cetera for how to create a palette function.
**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**

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

**Arguments**

- **deckgl**  
  A deckgl widget object.
- **data**  
  The url to fetch data from or a data object.
- **properties**  
  A named list of properties with names corresponding to the properties defined in the [deckgl-api-reference](https://deck.gl/#/documentation/deckgl-api-reference/layers/line-layer) for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- **...**  
  Named arguments that will be added to the properties object. Identical parameters are overwritten.
- **id**  
  The unique id of the layer.

**See Also**


**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]"),  
  tooltip = "{{from_name}} to {{to_name}}"
)
```
add_path_layer

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

add_mapbox_basemap

Add a basemap from mapbox to the deckgl widget

Description
Add a basemap from mapbox to the deckgl widget

Usage
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
Arguments

- **deckgl**: A deckgl widget object.
- **data**: The url to fetch data from or a data object.
- **properties**: A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- **id**: The unique id of the layer.

See Also

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

Examples

```r
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 = ~path,
  getColor = ~color,
  getWidth = 5,
  tooltip = ~name
)

deck <- deckgl(pitch = 25, zoom = 10.5) %>%
  add_path_layer(data = sample_data, properties = properties) %>%
  add_basemap() %>%
  add_control("Path Layer")

if (interactive()) deck
```

Description

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

Usage

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

Arguments

- `deckgl` A deckgl widget object.
- `data` The url to fetch data from or a data object.
- `properties` A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- `...` Named arguments that will be added to the properties object. Identical parameters are overwritten.
- `id` The unique id of the layer.

See Also

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

Examples

```r
## @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("deck.COORDINATE_SYSTEM.METER_OFFSETS"),
  coordinateOrigin = c(-122.4, 37.74),
  pointSize = 4,
  getPosition = ~position,
  getNormal = ~normal,
  getColor = ~color,
  lightSettings = list(),
  tooltip = "{{position.0}}, {{position.1}}"
)

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

add_control("Point Cloud Layer")

if (interactive()) deck

---

**add_polygon_layer**  
Add a polygon layer to the deckgl widget

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

**Usage**

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

**Arguments**

- **deckgl**: A deckgl widget object.
- **data**: The url to fetch data from or a data object.
- **properties**: A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
- **...**: Named arguments that will be added to the properties object. Identical parameters are overwritten.
- **id**: The unique id of the layer.

**See Also**


**Examples**

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

properties <- list(
```
pickleable = 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,
tooltip = "{{zipcode}}<br/>Population: {{population}}"
)

deck <- deckgl(zoom = 11, pitch = 25) %>%
  add_polygon_layer(data = sample_data, properties = properties) %>%
  add_basemap() %>%
  add_control("Polygon Layer")

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](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**  
  A deckgl widget object.

- **id**  
  The unique id of the layer.

- **tileServer**  
  base url of the tile server

- **properties**  
  A named list of properties with names corresponding to the properties defined in the [deckgl-api-reference](https://deck.gl/#/examples/core-layers/tile-layer) for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.

- **...**  
  Named arguments that will be added to the properties object. Identical parameters are overwritten.
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/
)

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

if (interactive()) deck
```

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,  # A deckgl widget object.
  data = NULL,  # The url to fetch data from or a data object.
  properties = list(),  # A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.
  ...,  # Named arguments that will be added to the properties object. Identical parameters are overwritten.
  id = "scatterplot-layer"  # The unique id of the layer.
)
```
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

```r
add_screen_grid_layer(
  deckgl,
  data = NULL,
  properties = list(),
  ...
)
```

Arguments

- **deckgl**: A deckgl widget object.
- **data**: The url to fetch data from or a data object.
- **properties**: A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The properties parameter can also be an empty list. In this case all props must be passed as named arguments.

Examples

```r
data("bart_stations")

properties <- list(
  getPosition = ~lng + lat,
  getRadius = "@=Math.sqrt(exits)", #JS("data => Math.sqrt(data.exits)"),
  radiusScale = 6,
  getFillColor = "@=code === 'LF' ? 'white': 'red'", #c(255, 140, 20),
  tooltip = "{{name}}"
)

deed <- deckgl(zoom = 10.5, pitch = 35) %>%
  add_scatterplot_layer(data = bart_stations, properties = properties) %>%
  add_basemap() %>%
  add_control("Scatterplot Layer")

if (interactive()) deck
```
add_source

... Named arguments that will be added to the properties object. Identical parameters are overwritten.

id The unique id of the layer.

See Also


Examples

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

properties <- list(
opacity = 0.8,
 cellSizePixels = 50,
 colorRange = RColorBrewer::brewer.pal(6, "Blues"),
 getPosition = ~lng + lat,
 getWeight = ~spaces
)

deck <- deckgl() %>%
  add_screen_grid_layer(data = sf_bike_parking, properties = properties) %>%
  add_basemap() %>%
  add_control("Screen Grid Layer")

if (interactive()) deck
```

---

add_source Add a data source to the deckgl widget

Description

Add a data source to the deckgl widget

Usage

```r
add_source(deckgl, id, data)
```

Arguments

dechgl A deckgl widget object.

id The unique id of the source.

data The url to fetch data from or a data object.
Examples

data("bart_stations")

dekgl() %>%
  add_source("bart-stations", bart_stations) %>%
  add_scatterplot_layer(
    source = "bart-stations",
    getPosition = ~lng + lat,
    getFillColor = "steelblue",
    getRadius = 50,
    radiusScale = 6
  ) %>%
  add_text_layer(
    source = "bart-stations",
    getPosition = ~lng + lat,
    getText = ~name,
    getSize = 15,
    sizeScale = 1.5,
    getColor = "white"
  ) %>%
  add_basemap()

---

**add_source_as_dep** Add source as JavaScript dep

Description

Add source as JavaScript dep

Usage

add_source_as_dep(deckgl, id, data)

Arguments

dekgl A deckgl widget object.
id The unique id of the source.
data The url to fetch data from or a data object.
Description

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

Usage

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

Arguments

- `deckgl`: A deckgl widget object.
- `data`: The url to fetch data from or a data object.
- `properties`: A named list of properties with names corresponding to the properties defined in the deckgl-api-reference for the given layer class. The `properties` parameter can also be an empty list. In this case all props must be passed as named arguments.
- `...`: Named arguments that will be added to the `properties` object. Identical parameters are overwritten.
- `id`: The unique id of the layer.

See Also

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

Examples

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

dek <- deckgl(zoom = 10, pitch = 35) %>%
  add_text_layer(
    data = bart_stations,
    pickable = TRUE,
    getPosition = ~lng + lat,
    getText = ~name,
    getSize = 15,
    getAngle = 0,
    getTextAnchor = "middle",
  )
```
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

bart_stations

<table>
<thead>
<tr>
<th>bart_stations</th>
<th>bart stations</th>
</tr>
</thead>
</table>

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

deskgl

Create a deckgl widget

Description
Create a deckgl widget

Usage
deskgl(
    latitude = 37.8,
    longitude = -122.45,
    zoom = 12,
    pitch = 0,
    bearing = 0,
    initial_view_state = NULL,
views = NULL,
width = NULL,
height = NULL,
 element_id = NULL,
...)

Arguments

latitude The latitude of the initial view state.
longitude The longitude of the initial view state.
zoom The zoom level of the initial view state.
pitch The pitch of the initial view state.
bearing The bearing of the initial view state.
initial_view_state
  The initial view state. If set, other view state arguments (longitude, latitude et cetera) are ignored.
views A single View, or an array of View instances. If not supplied, a single MapView will be created.
width The width of the widget.
height The height of the widget.
element_id The explicit id of the widget (usually not needed).
... Optional properties that are passed to the deck instance.

Value
dekgl widget

See Also

https://deck.gl/#/documentation/deckgl-api-reference/deck for optional properties that can be passed to the deck instance.
deckgl_proxy

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**

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

**Arguments**
- `filename` The filename of the 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**
```r
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

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
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
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**

Create a data accessor

**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

---

**set_view_state**  
Set the view state of the map

**Description**

Set the view state of the map

**Usage**

```
set_view_state(
    deckgl,
    latitude = 37.8,
    longitude = -122.45,
    zoom = 12,
    pitch = 0,
    bearing = 0
)
```

**Arguments**

- `deckgl`  A deckgl widget object.
- `latitude`  The latitude of the view state.
- `longitude`  The longitude of the view state.
- `zoom`  The zoom level of the view state.
- `pitch`  The pitch of the view state.
- `bearing`  The bearing of the view state.
**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**

dekgl_proxy
use_carto_style

Use a Carto style

Description

Use a Carto style

Usage

use_carto_style(theme = "dark-matter")

Arguments

tHEME

theme

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

use_contour_definition

Create a contour definition

Description

Create a contour definition

Usage

use_contour_definition(
    threshold = 1,
    color = c(255, 255, 255),
    stroke_width = 1
)

Arguments

threshold

The threshold value used in contour generation.

color

The RGB color array used to render contour lines.

stroke_width

The 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

*Create an icon definition on an atlas image*

**Description**

Create an icon definition on an atlas image

**Usage**

```r
use_icon_definition(
  x = 0,
  y = 0,
  width = 128,
  height = 128,
  anchor_x = (width/2),
  anchor_y = 128,
  mask = TRUE
)
```
use_tooltip

Arguments

x
The x position of the icon on the atlas image.

y
The y position of the icon on the atlas image.

width
The width of the icon on the atlas image.

height
The height of the icon on the atlas image.

anchor_x
The horizontal position of the icon anchor.

anchor_y
the vertical position of the 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

use_tooltip
Create a tooltip property

Description

Create a tooltip property

Usage

use_tooltip(html, style, ...)

Arguments

html The innerHTML of the element.

style A cssText string that will modify the default style of the element.

... not used

Tooltip template Syntax

The tooltip string is a mustache template in which variable names are identified by the double curly brackets (mustache tags) that surround them. The variable names available to the template are given by deck.gl's pickingInfo.object and vary by layer.

See Also

mustache for a complete syntax overview.
Examples

data("bart_segments")

props <- list(
  tooltip = use_tooltip(
    html = "{{from_name}} to {{to_name}}",
    style = "background: steelBlue; border-radius: 5px;"
  )
)

# The picking object of the hexagon layer offers a property that contains the list of points of the hexagon.
# You can iterate over this list as shown below.
data("sf_bike_parking")

html = "
  <p>{{position.0}}, {{position.1}}</p>
  <p>Count: {{points.length}}</p>
  <p>{{#points}}<div>{{address}}</div>{{/points}}</p>
Index

* datasets
  - bart_segments, 34
  - bart_stations, 35
  - sf_bike_parking, 42

add_arc_layer, 3
add_basemap, 4
add_bitmap_layer, 4
add_column_layer, 5
add_contour_layer, 6
add_control, 8
add_data, 9, 39
add_geojson_layer, 9
add_great_circle_layer, 10
add_grid_cell_layer, 11
add_grid_layer, 13
add_h3_cluster_layer, 14
add_h3_hexagon_layer, 15
add_heatmap_layer, 16
add_hexagon_layer, 17
add_icon_layer, 18
add_json_editor, 20
add_layer, 20
add_legend, 21, 22
add_legend_pal, 22
add_line_layer, 23
add_mapbox_basemap, 24
add_path_layer, 24
add_point_cloud_layer, 25
add_polygon_layer, 27
add_raster_tile_layer, 28
add_scatterplot_layer, 29
add_screen_grid_layer, 30
add_source, 21, 31
add_source_as_dep, 32
add_text_layer, 33

bart_segments, 34
bart_stations, 35

col_numeric, 22

dechgl, 35
dechgl-shiny, 36
dechgl_proxy, 37, 42
dechglOutput (dechgl-shiny), 36
does_it_work, 37
code_icon_atlas, 38
getcolor_to_rgb_array, 38
get_data, 39
get_first_element, 39
get_last_element, 40
get_position, 40
get_property, 41
renderDeckgl (dechgl-shiny), 36
set_view_state, 41
sf_bike_parking, 42
update_deckgl, 42
use_carto_style, 43
use_contour_definition, 43
use_default_icon_properties, 44
use_icon_definition, 44
use_tooltip, 21, 45

47