Package ‘mapdeck’

January 22, 2019

Type Package
Title Interactive Maps Using 'Mapbox GL JS' and 'Deck.gl'
Version 0.2.1
Date 2019-01-22
Description Provides a mechanism to plot an interactive map using 'Mapbox GL'  
(<https://www.mapbox.com/mapbox-gl-js/api/>), a javascript library for interactive maps,  
and 'Deck.gl' (<http://deck.gl/#/>), a javascript library which uses 'WebGL' for  
visualising large data sets.
License GPL-3
URL https://symbolixau.github.io/mapdeck/articles/mapdeck.html
BugReports https://github.com/SymbolixAU/mapdeck/issues
Encoding UTF-8
LazyData true
Depends R (>= 3.3.0)
Imports colourvalues (>= 0.2.2), geojsonsf (>= 1.3.0), googlePolylines  
(>= 0.7.2), htmltools, htmlwidgets, jsonify (>= 0.2.0),  
magrittr, Rcpp, shiny
RoxygenNote 6.1.1
Suggests covr, jsonlite, knitr, rmarkdown, sf, spatialwidget, testthat
VignetteBuilder knitr
LinkingTo BH, colourvalues, geojsonsf, justify, rapidjsonr, Rcpp,  
spatialwidget (>= 0.2.0)
NeedsCompilation yes
Author David Cooley [aut, cre]
Maintainer David Cooley <dcooley@symbolix.com.au>
Repository CRAN
Date/Publication 2019-01-22 09:20:03 UTC
### Description

The Arc Layer renders raised arcs joining pairs of source and target coordinates.

### Usage

```r
add_arc(map, data = get_map_data(map), layer_id = NULL, origin,
         destination, id = NULL, stroke_from = NULL,
         stroke_from_opacity = NULL, stroke_to = NULL,
         stroke_to_opacity = NULL, stroke_width = NULL, tooltip = NULL,
         auto_highlight = FALSE, highlight_colour = "#AFFFFF", legend = F,
```
```r
legend_options = NULL, legend_format = NULL, palette = "viridis",
na_colour = "#8080FF", update_view = TRUE, focus_layer = FALSE,
transitions = NULL)
```

**Arguments**

- **map**: a `mapdeck` map object
- **data**: data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- **layer_id**: single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
- **origin**: vector of longitude and latitude columns, or an `sfc` column
- **destination**: vector of longitude and latitude columns, or an `sfc` column
- **id**: an id value in data to identify layers when interacting in Shiny apps.
- **stroke_from**: variable or hex colour to use as the starting stroke colour
- **stroke_from_opacity**: Either a string specifying the column of `data` containing the stroke opacity of each shape, or a value between 1 and 255 to be applied to all the shapes
- **stroke_to**: variable or hex colour to use as the ending stroke colour
- **stroke_to_opacity**: Either a string specifying the column of `data` containing the stroke opacity of each shape, or a value between 1 and 255 to be applied to all the shapes
- **stroke_width**: width of the stroke
- **tooltip**: variable of `data` containing text or HTML to render as a tooltip
- **auto_highlight**: logical indicating if the shape under the mouse should auto-highlight
- **highlight_colour**: hex string colour to use for highlighting. Must contain the alpha component.
- **legend**: either a logical indicating if the legend(s) should be displayed, or a named list indicating which colour attributes should be included in the legend.
- **legend_options**: A list of options for controlling the legend.
- **legend_format**: A list containing functions to apply to legend values. See section legend
- **palette**: string or matrix. String will be one of `colourvalues::colour_palettes()`. A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.
- **na_colour**: hex string colour to use for NA values
- **update_view**: logical indicating if the map should update the bounds to include this layer
- **focus_layer**: logical indicating if the map should update the bounds to only include this layer
- **transitions**: list specifying the duration of transitions.
**Details**

`add_arc` supports POINT sf objects

MULTIPOINT objects will be treated as single points. That is, if an sf objet has one row with a MULTIPOINT object consisting of two points, this will be expanded to two rows of single POINTs. Therefore, if the origin is a MULTIPOINT of two points, and the destination is a single POINT, the code will error as there will be an uneven number of rows.

**data**

If `data` is a simple feature object, you need to supply the origin and destination columns, they aren’t automatically detected.

**id**

The `id` is returned to your R session from an interactive shiny environment by observing layer clicks. This is useful for returning the data.frame row relating to the clicked shape.

From within a shiny server you would typically use `observeEvent({input$map_arc_click})`, where `map` is the `map_id` supplied to `mapdeckOutput()`, and `arc` is the layer you are clicking on.

**legend**

The `legend_options` can be used to control the appearance of the legend. This should be a named list, where the names are one of

- `css` - a string of valid CSS for controlling the appearance of the legend
- `title` - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See examples in `add_arc`.

The `legend_format` can be used to control the format of the values in the legend. This should be a named list, where the names are one of

- `fill_colour`
- `stroke_colour`

depending on which type of colouring the layer supports.

The list elements must be functions to apply to the values in the legend.

**transitions**

The `transitions` argument lets you specify the time it will take for the shapes to transition from one state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported browsers and hardware.

The time is in milliseconds

Available transitions for arc

```r
list( origin = 0, destination = 0, stroke_from = 0, stroke_to = 0, stroke_width = 0 )
```
Examples

```r
## You need a valid access token from Mapbox
key <- 'abc'

flights <- read.csv(url)
flights$id <- seq_len(nrow(flights))
flights$stroke <- sample(1:3, size = nrow(flights), replace = T)
flights$info <- paste0("<b","flights$airport1"," - ", flights$airport2,"</b>")

mapdeck(token = key, style = mapdeck_style("dark"), pitch = 45)
add_arc(
  data = flights,
  layer_id = "arc_layer",
  origin = c("start_lon", "start_lat"),
  destination = c("end_lon", "end_lat"),
  stroke_from = "airport1",
  stroke_to = "airport2",
  stroke_width = "stroke",
  tooltip = "info",
  auto_highlight = TRUE,
  legend = T,
  legend_options = list(
    stroke_from = list(title = "Origin airport"),
    css = "max-height: 100px;"
  )
)

## Using a 2-sfc-column sf object
library(sf)

sf_flights <- cbind(
  sf::st_as_sf(flights, coords = c("start_lon", "start_lat")),
  sf::st_as_sf(flights[, c("end_lon", "end_lat")], coords = c("end_lon", "end_lat"))
)

mapdeck(token = key)
add_arc(
  data = sf_flights,
  origin = 'geometry',
  destination = 'geometry.1',
  layer_id = 'arcs',
  stroke_from = "airport1",
  stroke_to = "airport2"
)
```
add_geojson  

Add Geojson

Description

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

Usage

```r
add_geojson(map, data = get_map_data(map), layer_id = NULL,
            stroke_colour = NULL, stroke_opacity = NULL, stroke_width = NULL,
            fill_colour = NULL, fill_opacity = NULL, radius = NULL,
            elevation = NULL, light_settings = list(), legend = FALSE,
            legend_options = NULL, legend_format = NULL,
            auto_highlight = FALSE, tooltip = NULL,
            highlight_colour = "#aaffffff", palette = "viridis",
            na_colour = "#808080", update_view = TRUE, focus_layer = FALSE,
            transitions = NULL)
```

Arguments

- **map**  
  a mapdeck map object
- **data**  
  data to be used in the layer. Can be a url to GeoJSON
- **layer_id**  
  single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
- **stroke_colour**  
  column of an sf object, or field inside a GeoJSON property to use for colour
- **stroke_opacity**  
  column of an sf object, or field inside a GeoJSON property to use for opacity
- **stroke_width**  
  column of an sf object, or field inside a GeoJSON property to use for width
- **fill_colour**  
  column of an sf object, or field inside a GeoJSON property to use for colour
- **fill_opacity**  
  column of an sf object, or field inside a GeoJSON property to use for opacity
- **radius**  
  radius of points in meters. See details
- **elevation**  
  elevation of polygons. See details
- **light_settings**  
  list of light setting parameters. See `light_settings`
- **legend**  
  either a logical indicating if the legend(s) should be displayed, or a named list indicating which colour attributes should be included in the legend.
- **legend_options**  
  A list of options for controlling the legend.
- **legend_format**  
  A list containing functions to apply to legend values. See section `legend`
- **auto_highlight**  
  logical indicating if the shape under the mouse should auto-highlight
- **tooltip**  
  variable of data containing text or HTML to render as a tooltip. Only works on sf objects.
highlight_colour  
hex string colour to use for highlighting. Must contain the alpha component.

palette  
string or matrix. String will be one of `colourvalues::colour_palettes()`. A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.

na_colour  
hex string colour to use for NA values

update_view  
logical indicating if the map should update the bounds to include this layer

focus_layer  
logical indicating if the map should update the bounds to only include this layer

transitions  
list specifying the duration of transitions.

transitions

The transitions argument lets you specify the time it will take for the shapes to transition from one state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported browsers and hardware.

The time is in milliseconds

Available transitions for geojson

list( fill_colour = 0, stroke_colour = 0, stroke_width = 0, elevation = 0, radius = 0 )

Raw Geojson

If using a GeoJSON string, and you do not supply one of the colouring arguments, the function will look for these fields inside the properties field of the Geojson

fill_colour

- fill_colour
- fillColour
- fill_color
- fillColor
- fill

stroke_colour

- stroke_colour
- strokeColour
- stroke_color
- strokeColor
- stroke
- line_colour
- lineColour
- line_color
- lineColor
- line
stroke_width

• stroke_width
• stroke_Width
• line_width
• lineWidth
• width
• elevation
• radius

These colour values should be valid hex-colour strings.

If you do provide values for the colouring arguments, the function will assume you want to use specific fields in the geojson for colouring. However, if you only supply a fill_colour value, the function will not automatically detect the stroke_colour (and vice versa).

data

If the data is a simple feature object, the geometry column is automatically detected. If the sf object contains more than one geometry column and you want to use a specific one, you’ll need to set the active geometry using `sf::st_geometry(x) <- "your_column", where "your_column" is the name of the column you’re activating. See `?sf::st_geometry`.

legend

The legend_options can be used to control the appearance of the legend. This should be a named list, where the names are one of

• css - a string of valid css for controlling the appearance of the legend
• title - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See examples in `add_arc`.

The legend_format can be used to control the format of the values in the legend. This should be a named list, where the names are one of

• fill_colour
• stroke_colour

depending on which type of colouring the layer supports.

The list elements must be functions to apply to the values in the legend.
Examples

```r
## You need a valid access token from Mapbox
key <- 'abc'

## Not supplying colouring arguments, the function will try and find them in the GeoJSON
mapdeck(
  token = key,
  location = c(145, -37.9),
  zoom = 8,
  style = mapdeck_style("dark"),
  pitch = 35
)
  
## only supplying values to use for fill, the stroke will be default
mapdeck(
  token = key,
  location = c(145, -37.9),
  zoom = 8,
  style = mapdeck_style("dark"),
  pitch = 35
)
  
mapdeck(
  token = key,
  location = c(145, -37.9),
  zoom = 8,
  style = mapdeck_style("dark"),
  pitch = 35
)
  
mapdeck(
  token = key,
  location = c(145, -37.9),
  zoom = 8,
  style = mapdeck_style("dark"),
  pitch = 35
)
```

add_geojson(
    data = geojson,
    fill_colour = "random",
    stroke_colour = "random",
    elevation = 300
)

## putting elevation and width values onto raw GeoJSON
library(geojsonsf)
sf <- geojsonsf::geojson_sf(geojson)
sf$width <- sample(1:100, size = nrow(sf), replace = TRUE)
sf$elevation <- sample(100:1000, size = nrow(sf), replace = T)
geo <- geojsonsf::sf_geojson(sf)

mapdeck(
    token = key,
    location = c(145, -37.9),
    zoom = 8,
    style = mapdeck_style("dark"),
    pitch = 35
) %>%
add_geojson(
    data = geo
)

---

### add_grid

#### Add Grid

**Description**

The Grid Layer 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(map, data = get_map_data(map), lon = NULL, lat = NULL,
    polyline = NULL, colour_range = NULL, cell_size = 1000,
    extruded = TRUE, elevation_scale = 1, auto_highlight = FALSE,
    highlight_colour = "#AFFFFF", layer_id = NULL,
    update_view = TRUE, focus_layer = FALSE)
```

**Arguments**

- `map` a mapdeck map object
**add_grid**

data  data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
lon  column containing longitude values
lat  column containing latitude values
polyline  optional column of data containing the polylines, if using encoded polylines
colour_range  vector of 6 hex colours
cell_size  size of each cell in meters
extruded  logical indicating if cells are elevated or not
elevation_scale  cell elevation multiplier
auto_highlight  logical indicating if the shape under the mouse should auto-highlight
highlight_colour  hex string colour to use for highlighting. Must contain the alpha component.
layer_id  single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
update_view  logical indicating if the map should update the bounds to include this layer
focus_layer  logical indicating if the map should update the bounds to only include this layer

**Details**

add_grid supports POINT and MULTIPOINT sf objects

data

If the data is a simple feature object, the geometry column is automatically detected. If the sf object contains more than one geometry column and you want to use a specific one, you’ll need to set the active geometry using sf::st.geometry( x ) <- "your_column", where "your_column" is the name of the column you’re activating. See ?sf::st.geometry

**Examples**

```r
## You need a valid access token from Mapbox
key <- 'abc'

df <- read.csv(paste0('https://raw.githubusercontent.com/uber-common/deck.gl-data/master/',
                      'examples/3d-heatmap/heatmap-data.csv'))

df <- df[ !is.na(df$lng ), ]

mapdeck( token = key, style = mapdeck_style("dark"), pitch = 45 ) %>%
add_grid(
    data = df
    , lat = "lat"
```
add_hexagon

Description

The Hexagon Layer 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
define_hexagon_layer(data = get_map_data(map), polyline = NULL, lon = NULL, lat = NULL, layer_id = NULL, radius = 1000, elevation_scale = 1, auto_highlight = FALSE, highlight_colour = "#AFFFFF", colour_range = NULL, update_view = TRUE, focus_layer = FALSE)
```

Arguments

- **map**: a mapdeck map object
- **data**: data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- **polyline**: column of data containing the polylines
- **lon**: column containing longitude values
**add_hexagon**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat</td>
<td>column containing latitude values</td>
</tr>
<tr>
<td>layer_id</td>
<td>single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly</td>
</tr>
<tr>
<td>radius</td>
<td>in metres</td>
</tr>
<tr>
<td>elevation_scale</td>
<td>value to scale the elevations of the hexagons</td>
</tr>
<tr>
<td>auto_highlight</td>
<td>logical indicating if the shape under the mouse should auto-highlight</td>
</tr>
<tr>
<td>highlight_colour</td>
<td>hex string colour to use for highlighting. Must contain the alpha component.</td>
</tr>
<tr>
<td>colour_range</td>
<td>vector of 6 hex colours</td>
</tr>
<tr>
<td>update_view</td>
<td>logical indicating if the map should update the bounds to include this layer</td>
</tr>
<tr>
<td>focus_layer</td>
<td>logical indicating if the map should update the bounds to only include this layer</td>
</tr>
</tbody>
</table>

**Details**

add_hexagon supports POINT and MULTIPOINT sf objects

**data**

If the data is a simple feature object, the geometry column is automatically detected. If the sf object contains more than one geometry column and you want to use a specific one, you’ll need to set the active geometry using `sf::st_geometry(x) <- "your_column"`, where "your_column" is the name of the column you’re activating. See ?sf::st_geometry

**Examples**

```r
## Not run:

## You need a valid access token from Mapbox
key <- 'abc'

df <- read.csv(paste0('https://raw.githubusercontent.com/uber-common/deck.gl-data/master/examples/','3d-heatmap/heatmap-data.csv'))

df <- df[!is.na(df$lng), ]

mapdeck( token = key, style = mapdeck_style("dark"), pitch = 45) %>%
add_hexagon(
  data = df,
  lat = "lat",
  lon = "lng",
  layer_id = "hex_layer",
  elevation_scale = 100
)

library( sf )
```
add_line

Description
The Line Layer renders raised lines joining pairs of source and target coordinates

Usage
```
add_line(map, data = get_map_data(map), layer_id = NULL, origin, destination, id = NULL, stroke_colour = NULL, stroke_width = NULL, stroke_opacity = NULL, tooltip = NULL, auto_highlight = FALSE, highlight_colour = "#FFFFFF", palette = "viridis", na_colour = "#808080FF", legend = FALSE, legend_options = NULL, legend_format = NULL, update_view = TRUE, focus_layer = FALSE, transitions = NULL)
```

Arguments
- **map** a mapdeck map object
- **data** data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- **layer_id** single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
- **origin** vector of longitude and latitude columns, or an sfc column
- **destination** vector of longitude and latitude columns, or an sfc column
- **id** an id value in data to identify layers when interacting in Shiny apps.
- **stroke_colour** variable or hex colour to use as the ending stroke colour. transition enabled
- **stroke_width** width of the stroke
- **stroke_opacity** Either a string specifying the column of data containing the stroke opacity of each shape, or a value between 0 and 255 to be applied to all the shapes
- **tooltip** variable of data containing text or HTML to render as a tooltip
auto_highlight  logical indicating if the shape under the mouse should auto-highlight
highlight_colour  hex string colour to use for highlighting. Must contain the alpha component.
palette  string or matrix. String will be one of colourvalues::colour_palettes(). A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.
na_colour  hex string colour to use for NA values
legend  either a logical indicating if the legend(s) should be displayed, or a named list indicating which colour attributes should be included in the legend.
legend_options  A list of options for controlling the legend.
legend_format  A list containing functions to apply to legend values. See section legend
update_view  logical indicating if the map should update the bounds to include this layer
focus_layer  logical indicating if the map should update the bounds to only include this layer
transitions  list specifying the duration of transitions.

details

add_line supports POINT sf objects
MULTIPOINT objects will be treated as single points. That is, if an sf object has one row with a MULTIPOINT object consisting of two points, this will be expanded to two rows of single POINTs. Therefore, if the origin is a MULTIPOINT of two points, and the destination is a single POINT, the code will error as there will be an uneven number of rows.

transitions
The transitions argument lets you specify the time it will take for the shapes to transition from one state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported browsers and hardware.
The time is in milliseconds
Available transitions for line
list( origin = 0, destination = 0, stroke_colour = 0, stroke_width = 0 )

legend
The legend_options can be used to control the appearance of the legend. This should be a named list, where the names are one of

• css - a string of valid css for controlling the appearance of the legend
• title - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See examples in add_arc.
The legend_format can be used to control the format of the values in the legend. This should be a named list, where the names are one of

• fill_colour
• stroke_colour

depending on which type of colouring the layer supports.
The list elements must be functions to apply to the values in the legend.

id

The id is returned to your R session from an interactive shiny environment by observing layer clicks. This is useful for returning the data.frame row relating to the clicked shape.

From within a shiny server you would typically use `observeEvent(input$map_arc_click)` where 'map' is the map_id supplied to `mapdeckOutput()`, and 'arc' is the layer you are clicking on.

Examples

```r
## You need a valid access token from Mapbox
key <- 'abc'

flights <- read.csv(url)
flights$id <- seq_len(nrow(flights))
flights$stroke <- sample(1:3, size = nrow(flights), replace = T)

mapdeck( token = key, style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_line(
    data = flights,
    layer_id = "line_layer",
    origin = c("start_lon", "start_lat"),
    destination = c("end_lon", "end_lat"),
    stroke_colour = "airport1",
    stroke_width = "stroke",
    auto_highlight = TRUE
  )

## Using a 2-sfc-column sf object
library(sf)

sf_flights <- cbind(
  sf::st_as_sf(flights, coords = c("start_lon", "start_lat")),
  sf::st_as_sf(flights[, c("end_lon", "end_lat")], coords = c("end_lon", "end_lat"))
)

mapdeck( token = key )
  add_line(
    data = sf_flights,
    origin = 'geometry',
    destination = 'geometry.1',
    layer_id = 'arcs',
    stroke_colour = "airport1"
  )
```
Description

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

Usage

```r
code
add_path(map, data = get_map_data(map), polyline = NULL,
  stroke_colour = NULL, stroke_width = NULL, stroke_opacity = NULL,
  tooltip = NULL, layer_id = NULL, id = NULL,
  auto_highlight = FALSE, highlight_colour = "#AFFFFF",
  palette = "viridis", na_colour = "#808080FF", legend = FALSE,
  legend_options = NULL, legend_format = NULL, update_view = TRUE,
  focus_layer = FALSE, transitions = NULL)
```

Arguments

- **map**
  a mapdeck map object
- **data**
  data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- **polyline**
  optional column of `data` containing the polylines, if using encoded polylines
- **stroke_colour**
  variable of `data` or hex colour for the stroke. If used, elevation is ignored. transition enabled
- **stroke_width**
  width of the stroke. If used, elevation is ignored. transition enabled
- **stroke_opacity**
  value between 0 and 255. Either a string specifying the column of `data` containing the stroke opacity of each shape, or a single value to be applied to all the shapes
- **tooltip**
  variable of `data` containing text or HTML to render as a tooltip
- **layer_id**
  single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
- **id**
  an id value in `data` to identify layers when interacting in Shiny apps.
- **auto_highlight**
  logical indicating if the shape under the mouse should auto-highlight
- **highlight_colour**
  hex string colour to use for highlighting. Must contain the alpha component.
- **palette**
  string or matrix. String will be one of colourvalues::colour_palettes(). A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.
add_path

na_colour        hex string colour to use for NA values
legend           either a logical indicating if the legend(s) should be displayed, or a named list
                  indicating which colour attributes should be included in the legend.
legend_options   A list of options for controlling the legend.
legend_format    A list containing functions to apply to legend values. See section legend
update_view      logical indicating if the map should update the bounds to include this layer
focus_layer      logical indicating if the map should update the bounds to only include this layer
transitions      list specifying the duration of transitions.

Details

add_path supports LINESTRING and MULTILINESTRING sf objects

transitions

The transitions argument lets you specify the time it will take for the shapes to transition from one
state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported
browsers and hardware.
The time is in milliseconds
Available transitions for path
list( path = 0, stroke_colour = 0, stroke_width = 0 )

data

If the data is a simple feature object, the geometry column is automatically detected. If the sf object
contains more than one geometry column and you want to use a specific one, you’ll need to set the
active geometry using sf::st_geometry( x ) <= "your_column" , where "your_column" is
the name of the column you’re activating. See ?sf::st_geometry

legend

The legend_options can be used to control the appearance of the legend. This should be a named
list, where the names are one of

• css - a string of valid css for controlling the appearance of the legend
• title - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See
examples in add_arc.
The legend_format can be used to control the format of the values in the legend. This should be a
named list, where the names are one of

• fill_colour
• stroke_colour

depending on which type of colouring the layer supports.
The list elements must be functions to apply to the values in the legend.
id

The id is returned to your R session from an interactive shiny environment by observing layer clicks. This is useful for returning the data.frame row relating to the clicked shape.

From within a shiny server you would typically use `observeEvent(input$map_arc_click)`, where 'map' is the map_id supplied to mapdeckOutput(), and 'arc' is the layer you are clicking on.

Examples

```r
## You need a valid access token from Mapbox
key <- 'abc'

mapdeck(
  token = key,
  style = 'mapbox://styles/mapbox/dark-v9',
  location = c(145, -37.8),
  zoom = 10)
add_path(
  data = roads,
  stroke_colour = "RIGHT_LOC",
  layer_id = "path_layer",
  tooltip = "ROAD_NAME",
  auto_highlight = TRUE,
  legend = T
)
```

---

**add_pointcloud**

**Add Pointcloud**

**Description**

The Pointcloud Layer takes in coordinate points and renders them as circles with a certain radius.

**Usage**

```r
add_pointcloud(map, data = get_map_data(map), lon = NULL, lat = NULL,
  elevation = NULL, polyline = NULL, radius = 10,
  fill_colour = NULL, fill_opacity = NULL, tooltip = NULL,
  auto_highlight = FALSE, highlight_colour = "#AFFFFF",
  light_settings = list(), layer_id = NULL, id = NULL,
  palette = "viridis", na_colour = "#808080", legend = FALSE,
  legend_options = NULL, legend_format = NULL, update_view = TRUE,
  focus_layer = FALSE, transitions = NULL)
```
Arguments

map a mapdeck map object

data data to be used in the layer. All coordinates are expected to be in Web Mercator Projection

lon column containing longitude values

lat column containing latitude values
elevation column containing the elevation values

data optional column of data containing the polylines, if using encoded polylines

radius value in pixels of each point

fill_colour column of data or hex colour for the fill colour. transition enabled

fill_opacity value between 0 and 255. Either a string specifying the column of data containing the fill opacity of each shape, or a single value to be applied to all the shapes

tooltip variable of data containing text or HTML to render as a tooltip

auto_highlight logical indicating if the shape under the mouse should auto-highlight

highlight_colour hex string colour to use for highlighting. Must contain the alpha component.

light_settings list of light setting parameters. See light_settings

layer_id single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly

id an id value in data to identify layers when interacting in Shiny apps.

palette string or matrix. String will be one of colourvalues::colour_palettes(). A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.

na_colour hex string colour to use for NA values

legend either a logical indicating if the legend(s) should be displayed, or a named list indicating which colour attributes should be included in the legend.

legend_options A list of options for controlling the legend.

legend_format A list containing functions to apply to legend values. See section legend

update_view logical indicating if the map should update the bounds to include this layer

focus_layer logical indicating if the map should update the bounds to only include this layer

transitions list specifying the duration of transitions.

Details

add_pointcloud supports POINT and MULTIPOINT sf objects
add_pointcloud

transitions
The transitions argument lets you specify the time it will take for the shapes to transition from one state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported browsers and hardware.
The time is in milliseconds
Available transitions for pointcloud
list( position = 0, fill_colour = 0 )

data
If the data is a simple feature object, the geometry column is automatically detected. If the sf object contains more than one geometry column and you want to use a specific one, you’ll need to set the active geometry using sf::st_geometry(x) <- "your_column", where "your_column" is the name of the column you’re activating. See ?sf::st_geometry

legend
The legend_options can be used to control the appearance of the legend. This should be a named list, where the names are one of
• css - a string of valid css for controlling the appearance of the legend
• title - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See examples in add_arc.
The legend_format can be used to control the format of the values in the legend. This should be a named list, where the names are one of
• fill_colour
• stroke_colour

depending on which type of colouring the layer supports.
The list elements must be functions to apply to the values in the legend.

id
The id is returned to your R session from an interactive shiny environment by observing layer clicks. This is useful for returning the data.frame row relating to the clicked shape.
From within a shiny server you would typically use observeEvent(input$map_arc_click), where ‘map’ is the map_id supplied to mapdeckOutput(), and ‘arc’ is the layer you are clicking on.

Examples

## You need a valid access token from Mapbox
key <- 'abc'
df <- capitals
df$z <- sample(10000:1000000, size = nrow(df))

mapdeck(token = key, style = mapdeck_style("dark")) %>%
add_pointcloud(
  data = df,
  lon = 'lon',
  lat = 'lat',
  elevation = 'z',
  layer_id = 'point',
  fill_colour = "country",
  tooltip = "country",
  update_view = FALSE
)

## as an sf object with a Z attribute
library(sf)
sf <- sf::st_as_sf(df, coords = c("lon","lat","z"))

mapdeck(token = key, style = mapdeck_style("dark")) %>%
add_pointcloud(
  data = sf,
  layer_id = 'point',
  fill_colour = "country",
  tooltip = "country",
  update_view = FALSE
)

---

### add_polygon

**Add Polygon**

**Description**

The Polygon Layer renders filled and/or stroked polygons. If using sf objects only POLYGONs are supported, MULTIPOLYGONs are ignored.

**Usage**

```r
add_polygon(map, data = get_map_data(map), polyline = NULL,
stroke_colour = NULL, stroke_width = NULL, stroke_opacity = NULL,
fill_colour = NULL, fill_opactiy = NULL, elevation = NULL,
tooltip = NULL, auto_highlight = FALSE,
highlight_colour = "#AFFFFF", light_settings = list(),
layer_id = NULL, id = NULL, palette = "viridis",
na_colour = "#808080FF", legend = FALSE, legend_options = NULL,
legend_format = NULL, update_view = TRUE, focus_layer = FALSE,
transitions = NULL)
```
**add_polygon**

**Arguments**

- **map**: a mapdeck map object
- **data**: data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- **polyline**: optional column of data containing the polylines, if using encoded polylines
- **stroke_colour**: variable of data or hex colour for the stroke. If used, elevation is ignored. transition enabled
- **stroke_width**: width of the stroke. If used, elevation is ignored. transition enabled
- **stroke_opacity**: value between 0 and 255. Either a string specifying the column of data containing the stroke opacity of each shape, or a single value to be applied to all the shapes
- **fill_colour**: column of data or hex colour for the fill colour. transition enabled
- **fill_opacity**: value between 0 and 255. Either a string specifying the column of data containing the fill opacity of each shape, or a single value to be applied to all the shapes
- **elevation**: the height the polygon extrudes from the map. Only available if neither stroke_colour or stroke_width are supplied. transition enabled
- **tooltip**: variable of data containing text or HTML to render as a tooltip
- **auto_highlight**: logical indicating if the shape under the mouse should auto-highlight
- **highlight_colour**: hex string colour to use for highlighting. Must contain the alpha component.
- **light_settings**: list of light setting parameters. See [light_settings](#)
- **layer_id**: single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
- **id**: an id value in data to identify layers when interacting in Shiny apps.
- **palette**: string or matrix. String will be one of `colourvalues::colour_palettes()`. A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.
- **na_colour**: hex string colour to use for NA values
- **legend**: either a logical indicating if the legend(s) should be displayed, or a named list indicating which colour attributes should be included in the legend.
- **legend_options**: A list of options for controlling the legend.
- **legend_format**: A list containing functions to apply to legend values. See section legend
- **update_view**: logical indicating if the map should update the bounds to include this layer
- **focus_layer**: logical indicating if the map should update the bounds to only include this layer
- **transitions**: list specifying the duration of transitions.

**Details**

add_polygon supports POLYGON and MULTIPOLYGON sf objects
data

If the data is a simple feature object, the geometry column is automatically detected. If the sf object contains more than one geometry column and you want to use a specific one, you’ll need to set the active geometry using sf::st_geometry(x) <- "your_column", where "your_column" is the name of the column you’re activating. See ?sf::st_geometry

transitions

The transitions argument lets you specify the time it will take for the shapes to transition from one state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported browsers and hardware.

The time is in milliseconds

Available transitions for polygon

list( polygon = 0, fill_colour = 0, stroke_colour = 0, stroke_width = 0, elevation = 0 )

legend

The legend_options can be used to control the appearance of the legend. This should be a named list, where the names are one of

- css - a string of valid css for controlling the appearance of the legend
- title - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See examples in add_arc.

The legend_format can be used to control the format of the values in the legend. This should be a named list, where the names are one of

- fill_colour
- stroke_colour

depending on which type of colouring the layer supports.

The list elements must be functions to apply to the values in the legend.

id

The id is returned to your R session from an interactive shiny environment by observing layer clicks. This is useful for returning the data.frame row relating to the clicked shape.

From within a shiny server you would typically use observeEvent(input$map_arc_click), where 'map' is the map_id supplied to mapdeckOutput(), and 'arc' is the layer you are clicking on

Examples

```r
## You need a valid access token from Mapbox
key <- 'abc'
```
add_scatterplot

```r
library(sf)
library(geosf)

sf <- geosf::geojson_sf("https://symbolixau.github.io/data/geojson/SA2_2016_VIC.json")

mapdeck(
  token = key
  , style = mapdeck_style('dark')
)

add_polygon(
  data = sf
  , layer = "polygon_layer"
  , fill_colour = "SA2_NAME16"
)

df <- melbourne ## data.frame with encoded polylnies
df$elevation <- sample(100:5000, size = nrow(df))
df$info <- paste0("<b>SA2 - </b><br">, df$SA2_NAME)

mapdeck(
  token = key
  , style = mapdeck_style('dark')
  , location = c(145, -38)
  , zoom = 8
)

add_polygon(
  data = df
  , polyline = "geometry"
  , layer = "polygon_layer"
  , fill_colour = "SA2_NAME"
  , elevation = "elevation"
  , stroke_width = 200
  , tooltip = 'info'
  , legend = T
)
```

---

**add_scatterplot**  
*Add Scatterplot*

**Description**

The Scatterplot Layer takes in coordinate points and renders them as circles with a certain radius.

**Usage**

```r
add_scatterplot(map, data = get_map_data(map), lon = NULL,
    lat = NULL, polyline = NULL, radius = NULL, fill_colour = NULL,
```
fill_opacity = NULL, tooltip = NULL, auto_highlight = FALSE,
highlight_colour = "#AFFFFF", layer_id = NULL, id = NULL,
palette = "viridis", na_colour = "#880088FF", legend = FALSE,
legend_options = NULL, legend_format = NULL, update_view = TRUE,
focus_layer = FALSE, transitions = NULL)

Arguments

map                          a mapdeck map object
data                         data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
lon                          column containing longitude values
lat                          column containing latitude values
polyline                     optional column of data containing the polylines, if using encoded polylines
radius                       in metres
fill_colour                  column of data or hex colour for the fill colour. transition enabled
fill_opacity                 value between 0 and 255. Either a string specifying the column of data containing the fill opacity of each shape, or a single value to be applied to all the shapes
tooltip                      variable of data containing text or HTML to render as a tooltip
auto_highlight               logical indicating if the shape under the mouse should auto-highlight
highlight_colour             hex string colour to use for highlighting. Must contain the alpha component.
layer_id                     single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
id                           an id value in data to identify layers when interacting in Shiny apps.
palette                      string or matrix. String will be one of colourvalues::colour_palettes(). A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.
na_colour                    hex string colour to use for NA values
legend                       either a logical indicating if the legend(s) should be displayed, or a named list indicating which colour attributes should be included in the legend.
legend_options               A list of options for controlling the legend.
legend_format                A list containing functions to apply to legend values. See section legend
update_view                  logical indicating if the map should update the bounds to include this layer
focus_layer                  logical indicating if the map should update the bounds to only include this layer
transitions                  list specifying the duration of transitions.

Details

add_scatterplot supports POINT and MULTIPOINT sf objects
transitions

The transitions argument lets you specify the time it will take for the shapes to transition from one state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported browsers and hardware.

The time is in milliseconds

Available transitions for scatterplot

list( position = 0, fill_colour = 0, radius = 0 )

data

If the data is a simple feature object, the geometry column is automatically detected. If the sf object contains more than one geometry column and you want to use a specific one, you'll need to set the active geometry using sf::st.geometry( x ) <- "your_column", where "your_column" is the name of the column you're activating. See ?sf::st.geometry

legend

The legend_options can be used to control the appearance of the legend. This should be a named list, where the names are one of

• css - a string of valid css for controlling the appearance of the legend
• title - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See examples in add.arc.

The legend_format can be used to control the format of the values in the legend. This should be a named list, where the names are one of

• fill_colour
• stroke_colour

depending on which type of colouring the layer supports.

The list elements must be functions to apply to the values in the legend.

id

The id is returned to your R session from an interactive shiny environment by observing layer clicks. This is useful for returning the data.frame row relating to the clicked shape.

From within a shiny server you would typically use observeEvent(input$map_arc_click), where 'map' is the map_id supplied to mapdeckOutput(), and 'arc' is the layer you are clicking on

Examples

```r
# You need a valid access token from Mapbox
key <- 'abc'
```
The Screen Grid Layer takes in an array of latitude and longitude coordinated points, aggregates them into histogram bins and renders as a grid.
Usage

```r
add_screengrid(map, data = get_map_data(map), lon = NULL, lat = NULL,
polyline = NULL, weight = NULL, colour_range = NULL,
opacity = 0.8, cell_size = 50, layer_id = NULL,
update_view = TRUE, focus_layer = FALSE)
```

Arguments

- **map**: a mapdeck map object
- **data**: data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- **lon**: column containing longitude values
- **lat**: column containing latitude values
- **polyline**: optional column of data containing the polylines, if using encoded polylines
- **weight**: the weight of each value
- **colour_range**: vector of 6 hex colours
- **opacity**: opacity of cells. Value between 0 and 1
- **cell_size**: size of grid squares in pixels
- **layer_id**: single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly
- **update_view**: logical indicating if the map should update the bounds to include this layer
- **focus_layer**: logical indicating if the map should update the bounds to only include this layer

Details

`add_screengrid` supports POINT and MULTIPOINT sf objects

**data**

If the `data` is a simple feature object, the geometry column is automatically detected. If the sf object contains more than one geometry column and you want to use a specific one, you'll need to set the active geometry using `sf::st_geometry( x ) <- "your_column"`, where "your_column" is the name of the column you're activating. See `sf::st_geometry`

Examples

```r
## You need a valid access token from Mapbox
key <- 'abc'

df <- read.csv(paste0(
  'https://raw.githubusercontent.com/uber-common/deck.gl-data/master/','
  'examples/3d-heatmap/heatmap-data.csv'))
```
df <- df[!is.na(df$lng),]
df$weight <- sample(1:10, size = nrow(df), replace = T)

mapdeck( token = key, style = mapdeck_style('dark'), pitch = 45 )
add_screengrid(
  data = df
  , lat = "lat"
  , lon = "lng"
  , weight = "weight",
  , layer_id = "screengrid_layer"
  , cell_size = 10
  , opacity = 0.3
)

## as an sf object
library(sf)
sf <- sf::st_as_sf( df, coords = c("lng", "lat") )
mapdeck( token = key, style = mapdeck_style('dark'), pitch = 45 )
add_screengrid(
  data = sf
  , weight = "weight",
  , layer_id = "screengrid_layer"
  , cell_size = 10
  , opacity = 0.3
)

---

**add_sf**

*Add sf*

**Description**

Adds an sf object to the map.

**Usage**

```r
add_sf(map, data = get_map_data(map), ...)
```

**Arguments**

- `map` - a mapdeck map object
- `data` - data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- `...` - other arguments passed to one of the plotting layers. See details
Details

The plotting layer is determined by the type of sf geometries.

- POINT and MULTIPROIN objects will call `add_scatterplot`
- LINESTRING and MULTILINESTRING objects will call `add_path`
- POLYGON and MULTIPOLYGON objects will call `add_polygon`
- GEOMETRY objects will call `add_geojson`

Description

The Text Layer renders text labels on the map

Usage

```r
add_text(map, data = get_map_data(map), text, lon = NULL, lat = NULL,
polyline = NULL, fill_colour = NULL, fill_opacity = NULL,
size = NULL, angle = NULL, anchor = NULL,
alignment_baseline = NULL, tooltip = NULL, layer_id = NULL,
id = NULL, auto_highlight = FALSE, highlight_colour = "#AFFFFF",
palette = "viridis", na_colour = "#808080FF", legend = FALSE,
legend_options = NULL, update_view = TRUE, focus_layer = FALSE,
transitions = NULL)
```

Arguments

- **map**: a mapdeck map object
- **data**: data to be used in the layer. All coordinates are expected to be in Web Mercator Projection
- **text**: column of data containing the text. The data must be a character.
- **lon**: column containing longitude values
- **lat**: column containing latitude values
- **polyline**: optional column of data containing the polylines, if using encoded polylines
- **fill_colour**: column of data or hex colour for the fill colour. transition enabled
- **fill_opacity**: value between 0 and 255. Either a string specifying the column of data containing the fill opacity of each shape, or a single value to be applied to all the shapes
- **size**: column of data containing the size of the text
- **angle**: column of data containing the angle of the text
- **anchor**: column of data containing the anchor of the text. One of 'start', 'middle' or 'end'
add_text supports POINT and MULTIPOINT sf objects

transitions

The transitions argument lets you specify the time it will take for the shapes to transition from one state to the next. Only works in an interactive environment (Shiny) and on WebGL-2 supported browsers and hardware.

The time is in milliseconds

Available transitions for text

list( position = 0, fill_colour = 0, angle = 0, size = 0 )

legend

The legend_options can be used to control the appearance of the legend. This should be a named list, where the names are one of

- css - a string of valid css for controlling the appearance of the legend
- title - a string to use for the title of the legend

If the layer allows different fill and stroke colours, you can use different options for each. See examples in add_arc.

The legend_format can be used to control the format of the values in the legend. This should be a named list, where the names are one of

alignment_baseline
column of data containing the alignment. One of 'top', 'center' or 'bottom'

tooltip
variable of data containing text or HTML to render as a tooltip

layer_id
single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly

id
an id value in data to identify layers when interacting in Shiny apps.

auto_highlight
logical indicating if the shape under the mouse should auto-highlight

highlight_colour
hex string colour to use for highlighting. Must contain the alpha component.

palette
string or matrix. String will be one of colourvalues::colour_palettes(). A matrix is a 3 or 4 column numeric matrix of values between [0, 255], where the 4th column represents the alpha.

na_colour
hex string colour to use for NA values

legend
either a logical indicating if the legend(s) should be displayed, or a named list indicating which colour attributes should be included in the legend.

legend_options
A list of options for controlling the legend.

update_view
logical indicating if the map should update the bounds to include this layer

focus_layer
logical indicating if the map should update the bounds to only include this layer

transitions
list specifying the duration of transitions.
- fill_colour
- stroke_colour

depending on which type of colouring the layer supports.
The list elements must be functions to apply to the values in the legend.

**id**

The id is returned to your R session from an interactive shiny environment by observing layer clicks. This is useful for returning the data.frame row relating to the clicked shape.

From within a shiny server you would typically use `observeEvent(input$map_arc_click)`, where 'map' is the map_id supplied to `mapdeckOutput()`, and 'arc' is the layer you are clicking on.

**Examples**

```r
## You need a valid access token from Mapbox
key <- 'abc'

mapdeck(
  token = key,
  style = mapdeck_style('dark'))
%>%
  add_text(
    data = capitals
    , lon = 'lon'
    , lat = 'lat'
    , fill_colour = 'country'
    , text = 'capital'
    , layer_id = 'text'
  )
```

<table>
<thead>
<tr>
<th>capitals</th>
<th>Capital cities for each country</th>
</tr>
</thead>
</table>

**Description**

A data set containing the coordinates of 200 capital cities in the world

**Usage**

capitals
Format
A data frame with 200 observations and 4 variables

- **country**  country name
- **capital**  capital name
- **lat**  latitude of capital
- **lon**  longitude of capital

---

**clear_arc**
**Clear Arc**

Description
Cleans elements from a map

Usage

```r
clear_arc(map, layer_id = NULL)
clear_geojson(map, layer_id = NULL)
clear_grid(map, layer_id = NULL)
clear_hexagon(map, layer_id = NULL)
clear_line(map, layer_id = NULL)
clear_path(map, layer_id = NULL)
clear_pointcloud(map, layer_id = NULL)
clear_polygon(map, layer_id = NULL)
clear_scatterplot(map, layer_id = NULL)
clear_screengrid(map, layer_id = NULL)
clear_text(map, layer_id = NULL)
```

Arguments
- **map**  a mapdeck map object
- **layer_id**  the layer_id of the layer you want to clear
**clear_legend**

**Clear Legend**

**Description**

Clears the legend for a given layer_id

**Usage**

`clear_legend(map_id, layer_id)`

**Arguments**

- `map_id`: the id of the map you want to clear the legend from.
- `layer_id`: single value specifying an id for the layer. Use this value to distinguish between shape layers of the same type. Layers with the same id are likely to conflict and not plot correctly

---

**clear_tokens**

**Clear tokens**

**Description**

Clears the access tokens

**Usage**

`clear_tokens()`

---

**geojson**

**Geojson**

**Description**

A GeoJSON object of polygons, lines and points in Melbourne

**Usage**

`geojson`

**Format**

- a 'json' object
### light_settings

#### Description

List object containing light settings.

#### Details

Available in add_geojson, add_pointcloud and add_polygon

- `numberOfLights` - the number of lights. Maximum of 5
- `lightsPosition` - vector of x, y, z coordinates. Must be 3 times the number of lights
- `ambientRatio` - the ambient ratio of the lights

#### Examples

```r
light <- list(
  lightsPosition = c(-150, 75, 0),
  numberOfLights = 1,
  ambientRatio = 0.2
)
```

---

### mapdeck

#### Description

mapdeck

#### Usage

```r
mapdeck(data = NULL, token = get_access_token(api = "mapbox"),
  width = NULL, height = NULL, padding = 0,
  style = "mapbox://styles/mapbox/streets-v9", pitch = 0, zoom = 0,
  bearing = 0, location = c(0, 0))
```
Arguments

- **data**: data to be used on the map. All coordinates are expected to be in Web Mercator Projection.
- **token**: Mapbox Access token. Use `set_token()` to set a global token. If left empty layers will still be plotted, but without a Mapbox map.
- **width**: the width of the map
- **height**: the height of the map
- **padding**: the padding of the map
- **style**: the style of the map
- **pitch**: the pitch angle of the map
- **zoom**: zoom level of the map
- **bearing**: bearing of the map between 0 and 360
- **location**: unnamed vector of lon and lat coordinates (in that order)

---

Shiny bindings for mapdeck

---

Description

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

Usage

```r
mapdeckOutput(outputId, width = "100\%", height = "400px")
renderMapdeck(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 mapdeck
- **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.
### mapdeck_dispatch

**mapdeck dispatch**

#### Description

Extension points for plugins

#### Usage

```javascript
mapdeck_dispatch(map, funcName, mapdeck = stop(paste(funcName, "requires a map update object")), mapdeck_update = stop(paste(funcName, "does not support map update objects")))
```

```
invoke_method(map, method, ...)
```

#### Arguments

- **map**: a map object, as returned from `mapdeck`
- **funcName**: the name of the function that the user called that caused this `mapdeck_dispatch` call; for error message purposes
- **mapdeck**: an action to be performed if the map is from `mapdeck`
- **mapdeck_update**: an action to be performed if the map is from `mapdeck_update`
- **method**: the name of the JavaScript method to invoke
- **...**: unnamed arguments to be passed to the JavaScript method

#### Value

`mapdeck_dispatch` returns the value of `mapdeck` or an error. `invokeMethod` returns the map object that was passed in, possibly modified.

### mapdeck_style

**Mapdeck Style**

#### Description

Various styles available to all Mapbox accounts using a valid access token

#### Usage

```javascript
mapdeck_style(style = c("dark", "light", "outdoors", "streets", "satellite", "satellite-streets"))
```

#### Arguments

- **style**: one of streets, outdoors, light, dark, satellite, satellite-streets
Examples

```r
## You need a valid access token from Mapbox
key <- 'abc'

## set a map style
mapdeck(token = key, style = mapdeck_style("dark"))
```

---

**mapdeck_tokens**  
*Mapdeck_tokens*

---

**Description**

Retrieves the mapdeck token that has been set

**Usage**

```r
mapdeck_tokens()
```

---

**mapdeck_update**  
*Mapdeck update*

---

**Description**

Update a Mapdeck map in a shiny app. Use this function whenever the map needs to respond to reactive content.

**Usage**

```r
mapdeck_update(map_id, session = shiny::getDefaultReactiveDomain(),
               data = NULL, deferUntilFlush = TRUE)
```

**Arguments**

- **map_id**: string containing the output ID of the map in a shiny application.
- **session**: the Shiny session object to which the map belongs; usually the default value will suffice.
- **data**: data to be used in the map. All coordinates are expected to be in Web Mercator Projection
- **deferUntilFlush**: indicates whether actions performed against this instance should be carried out right away, or whether they should be held until after the next time all of the outputs are updated; defaults to TRUE.
mapdeck_view  Mapdeck view

Description

Changes the view of the map.

Usage

mapdeck_view(map, location = NULL, zoom = NULL, pitch = NULL,
bearing = NULL, duration = NULL, transition = c("linear", "fly"))

Arguments

- map: a mapdeck map object
- location: unnamed vector of lon and lat coordinates (in that order)
- zoom: zoom level of the map
- pitch: the pitch angle of the map
- bearing: bearing of the map between 0 and 360
- duration: time in milliseconds of the transition
- transition: type of transition

melbourne  Polygons in and around Melbourne

Description

A data set containing statistical area 2 regions of central (and surrounds) Melbourne.

Usage

melbourne

Format

An sfencoded and data frame object with 41 observations and 8 variables. See library googlePoly-lines for information on sfencoded objects.
Description

A simple feature sf object of roads in central Melbourne

Usage

roads

Format

An sf and data frame object with 18286 observations and 16 variables

Details

Obtained from www.data.gov.au and distributed under the Creative Commons 4 License https://creativecommons.org/licenses/by/4.0/

Description

Sets an access token so it’s available for all mapdeck calls. See details

Usage

set_token(token)

Arguments

token Mapbox access token

Details

Use set_token to make access tokens available for all the mapdeck() calls in a session so you don’t have to keep specifying the token argument each time
Description

Uses the pipe operator (\%) to chain statements. Useful for adding layers to a mapdeck map

Arguments

1hs, rhs A mapdeck map and a layer to add to it

Examples

```r
key <- "your_api_key"
mapdeck(key = key) %>%
  add_scatterplot(
    data = capitals,
    lat = "lat",
    lon = "lon",
    radius = 100000,
    fill_colour = "country",
    layer_id = "scatter_layer"
  )
```
Index

*Topic **datasets**
  - capitals, 33
  - geojson, 35
  - melbourne, 40
  - roads, 41

```
%>%, 42
add_arc, 2, 4, 8, 15, 18, 21, 24, 27, 32
add_geojson, 6, 31, 36
add_grid, 10
add_hexagon, 12
add_path, 17, 31
add_pointcloud, 19, 36
add_polygon, 22, 31, 36
add_scatterplot, 25, 31
add_screengrid, 28
add_sf, 30
add_text, 31
```

- capitals, 33
- clear_arc, 34
- clear_geojson (clear_arc), 34
- clear_grid (clear_arc), 34
- clear_hexagon (clear_arc), 34
- clear_legend, 35
- clear_line (clear_arc), 34
- clear_path (clear_arc), 34
- clear_pointcloud (clear_arc), 34
- clear_polygon (clear_arc), 34
- clear_scatterplot (clear_arc), 34
- clear_screengrid (clear_arc), 34
- clear_text (clear_arc), 34
- clear_tokens, 35

- geojson, 35

- invoke_method (mapdeck_dispatch), 38

- light_settings, 6, 20, 23, 36

mapdeck, 36, 38
mapdeck-shiny, 37
mapdeck_dispatch, 38
mapdeck_style, 38
mapdeck_tokens, 39
mapdeck_update, 38, 39
mapdeck_view, 40
mapdeckOutput (mapdeck-shiny), 37
melbourne, 40
renderMapdeck (mapdeck-shiny), 37
roads, 41
set_token, 41