Package ‘mapdeck’
March 24, 2020

Type Package
Title Interactive Maps Using 'Mapbox GL JS' and 'Deck.gl'
Version 0.3.1
Date 2020-03-23
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 (>= 3.5.0)
Imports colourvalues (>= 0.3.4), googlePolylines (>= 0.7.2), geojsonsf (>= 1.3.3), htmltools, htmlwidgets, jsonify (>= 1.1.1), magrittr, Rcpp, shiny, sfheaders (>= 0.2.1)
RoxygenNote 7.0.2
Suggests covr, googleway, jsonlite, knitr, rmarkdown, spatialwidget, testthat
VignetteBuilder knitr
LinkingTo BH, colourvalues, geojsonsf (>= 1.3.3), jsonify (>= 1.1.1), rapidjsonr, Rcpp, sfheaders (>= 0.2.1), spatialwidget (>= 0.2.2)
NeedsCompilation yes
Author David Cooley [aut, cre]
Maintainer David Cooley <dcooley@symbolix.com.au>
Repository CRAN
Date/Publication 2020-03-24 07:40:06 UTC
R topics documented:

- add_arc .................................................. 3
- add_bitmap ................................................ 8
- add_column ............................................... 9
- add_dependencies ....................................... 13
- add_geojson ............................................. 14
- add_greatcircle ....................................... 19
- add_grid .................................................. 23
- add_heatmap ........................................... 27
- add_hexagon ........................................... 29
- add_line ............................................... 33
- add_mesh ............................................. 36
- add_path ................................................ 39
- add_pointcloud ....................................... 42
- add_polygon .......................................... 45
- add_scatterplot ...................................... 49
- add_screengrid ....................................... 54
- add_sf .................................................. 56
- add_text ............................................. 57
- add_title ........................................... 60
- add_trips ............................................ 61
- capitals ............................................... 63
- city_trail ............................................ 64
- clear_arc ............................................ 64
- clear_legend ......................................... 65
- clear_tokens ......................................... 65
- geojson ............................................... 66
- legend_element ...................................... 66
- light_settings ....................................... 67
- mapdeck ............................................... 67
- mapdeck-shiny ....................................... 69
- mapdeck_dependencies ................................ 69
- mapdeck_dispatch ................................... 70
- mapdeck_legend ..................................... 70
- mapdeck_style ....................................... 71
- mapdeck_tokens ..................................... 72
- mapdeck_update ..................................... 72
- mapdeck_view ....................................... 73
- melbourne .......................................... 73
- melbourne_mesh .................................... 74
- roads .................................................. 74
- set_token ............................................ 74
- update_style ........................................ 75
- \%>\% .................................................. 76

Index .................................................. 77
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,
  tilt = NULL,
  height = NULL,
  tooltip = NULL,
  auto_highlight = FALSE,
  highlight_colour = "#AAFFFFFF",
  legend = F,
  legend_options = NULL,
  legend_format = NULL,
  palette = "viridis",
  na_colour = "#808080FF",
  update_view = TRUE,
  focus_layer = FALSE,
  transitions = NULL,
  digits = 6,
  brush_radius = NULL
)
```

Arguments

- `map` a mapdeck map object
- `data` data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
- `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
add_arc

origin vector of longitude and latitude columns, and optionally an elevation column, or an sfc column
destination vector of longitude and latitude columns, and optionally an elevation column, or an sfc column
id an id value in data to identify layers when interacting in Shiny apps.
stroke_from column of data or hex colour to use as the starting stroke colour. If using a hex colour, use either a single value, or a vector the same length as data
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. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
stroke_to column of data or hex colour to use as the ending stroke colour. If using a hex colour, use either a single value, or a vector the same length as data
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. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
stroke_width width of the stroke in pixels
tilt value to tilt the arcs to the side, in degrees [-90, 90]
height value to multiply the height.
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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")
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.
digits number of digits for rounding coordinates
brush_radius radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed
**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
- `digits` - number to round the legend values to

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'
set_token(key)

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(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;" )
  )

mapdeck(style = mapdeck_style("dark")) %>%
  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"
  )

## Arcs can have an elevated start & destination
flights$start_elev <- sample(100000:1000000, size = nrow(flights), replace = TRUE)

mapdeck(style = mapdeck_style("dark")) %>%
  add_arc(
    data = flights,
    layer_id = "arc_layer",
    origin = c("start_lon", "start_lat", "start_elev"),
    destination = c("end_lon", "end_lat", "start_elev"),
    stroke_from = "airport1",
    stroke_to = "airport2"
  )
```
## Using a 2-sfc-column sf object

```r
library(sfheaders)

sf_flights <- sfheaders::sf_point(
  flights
  , x = "start_lon"
  , y = "start_lat"
  , z = "start_elev"
  , keep = TRUE
)
destination <- sfheaders::sfc_point(
  flights
  , x = "end_lon"
  , y = "end_lat"
  , z = "start_elev"
)

sf_flights$destination <- destination

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

## using a brush

```r
mapdeck(
  style = mapdeck_style("light")
  add_arc(
    data = sf_flights
    , origin = 'geometry'
    , destination = 'destination'
    , layer_id = 'arcs'
    , stroke_from = "airport1"
    , stroke_to = "airport2"
    , stroke_width = 4
    , brush_radius = 500000
  )
)
add_bitmap  

Description

Adds an image to a map

Usage

add_bitmap(
  map,
  image,
  bounds,
  desaturate = 0,
  transparent_colour = "#000000",
  tint_colour = "#FFFFFF",
  layer_id = NULL,
  update_view = TRUE,
  focus_layer = FALSE
)

Arguments

map  
a mapdeck map object
image  
url to an image to use on the map
bounds  
coordinates of the bounding box of the image [left, bottom, right, top]
desaturate  
the desaturation of the bitmap, in range [0,1], 0 being the original colour and 1 being greyscale
transparent_colour  
the colour to use for transparent pixels as a hex string
tint_colour  
the colour to tint the bitmap by, as a hex string
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

Examples

set_token( "MAPBOX_TOKEN" )

mapdeck(location = c(-122.3, 37.8), zoom = 10) %>%
  add_bitmap(
add_column

image = paste0(
  'https://raw.githubusercontent.com/uber-common/deck.gl-data/master/'
  , 'website/sf-districts.png'
  , bounds = c(-122.519, 37.7045, -122.355, 37.829)
)

mapdeck(location = c(-75.9, 40.9), zoom = 4) %>%
  add_bitmap(
    image = 'https://docs.mapbox.com/mapbox-gl-js/assets/radar.gif'
    , bounds = c(-80.425, 37.936, -71.516, 46.437)
  )

mapdeck(location = c(-75.9, 40.9), zoom = 4) %>%
  add_bitmap(
    image = 'https://docs.mapbox.com/mapbox-gl-js/assets/radar.gif'
    , bounds = c(-80.425, 37.936, -71.516, 46.437)
    , tint_colour = "#FF0000"
  )

mapdeck(location = c(-75.9, 40.9), zoom = 4) %>%
  add_bitmap(
    image = 'https://docs.mapbox.com/mapbox-gl-js/assets/radar.gif'
    , bounds = c(-80.425, 37.936, -71.516, 46.437)
    , desaturate = 1
  )

---

add_column  Add column

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(
  map,
  data = get_map_data(map),
  polyline = NULL,
  lon = NULL,
  lat = NULL,
  fill_colour = NULL,
  fill_opacity = NULL,
  stroke_colour = NULL,
  stroke_opacity = NULL,
stroke_width = NULL,
radius = 1000,
elevation = NULL,
elevation_scale = 1,
coverage = 1,
angle = 0,
disk_resolution = 20,
tooltip = NULL,
auto_highlight = FALSE,
highlight_colour = "#AFFFFF",
layer_id = NULL,
id = NULL,
palette = "viridis",
na_colour = "#808080FF",
legend = FALSE,
legend_options = NULL,
legend_format = NULL,
update_view = TRUE,
focus_layer = FALSE,
digits = 6,
transitions = NULL,
brush_radius = NULL
)

Arguments

map a mapdeck map object
data data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
polyline column of data containing the polylines
lon column containing longitude values
lat column containing latitude values
fill_colour column of data or hex colour for the fill colour. If using a hex colour, use either a single value, or a vector the same length as data
fill_opacity Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
stroke_colour variable of data or hex colour for the stroke. If used, elevation is ignored. If using a hex colour, use either a single value, or a vector the same length as data
stroke_opacity Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
stroke_width width of the stroke in meters. If used, elevation is ignored. Default 1.
radius in metres. Default 1000
elevation  the height the polygon extrudes from the map. Only available if neither stroke_colour or stroke_width are supplied. Default 0

elevation_scale   value to scale the elevations of the columns Default 1

coverage  radius multiplier, in range [0,1]. The radius of the disk is calculated by coverage * radius

angle  disk rotation, counter-clockwise, in degrees

disk_resolution   The number of sides to render the disk as. The disk is a regular polygon that fits inside the given radius. A higher resolution will yield a smoother look close-up, but also requires more resources to render.

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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")

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

digits  number of digits for rounding coordinates

transitions  list specifying the duration of transitions.

brush_radius  radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed

Details

add_column 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`

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
- digits - number to round the legend values to

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
## Not run:
## You need a valid access token from Mapbox
key <- 'abc'
set_token( key )

df <- capitals
df$elev <- sample(50000:500000, size = nrow(df), replace = T)

mapdeck(style = mapdeck_style("dark"), pitch = 45) %>%
  add_column(
    data = df
    , lat = "lat"
```
add_dependencies

Description

Adds the required mapdeck javascript dependencies to a map when not using a mapdeck map.

Usage

add_dependencies(map)

Arguments

map

the map object to which dependencies will be added

Examples

## use with a google map from googleway
library(googleway)

set_key("GOOGLE_MAP_KEY")
```r
google_map() %>%
add_dependencies() %>%
add_scatterplot(
  data = capitals
  , lon = "lon"
  , lat = "lat"
  , fill_colour = "country"
  , radius = 10000
)
```

---

### 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,
  dash_size = NULL,
  dash_gap = NULL,
  fill_colour = NULL,
  fill_opacity = NULL,
  radius = NULL,
  elevation = NULL,
  extruded = FALSE,
  light_settings = list(),
  legend = F,
  legend_options = NULL,
  legend_format = NULL,
  auto_highlight = FALSE,
  tooltip = NULL,
  highlight_colour = "#AAFFFFFF",
  palette = "viridis",
```
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 (in meters)
dash_size   size of each dash, relative to the width of the stroke
dash_gap    size of the gap between dashes, relative to the width of the stroke
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. Default 1. See details
elevation   elevation of polygons. Default 0. See details
extruded    logical indicating if polygons should extrude from the map. If TRUE, stroke_colour for polygons is ignored
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. A legend is only shown if you supply one of the colour arguments (fill or stroke)
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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. `list(fill_colour = "viridis", stroke_colour = "inferno")`

na_colour  hex string colour to use for NA values

line_width_units  The units of the line width, one of 'meters', 'pixels'. When zooming in and out, meter sizes scale with the base map, and pixel sizes remain the same on screen.

line_width_scale  The line width multiplier that multiplied to all lines, including the LineString and MultiLineString features and also the outline for Polygon and MultiPolygon features if the stroked attribute is true

line_width_min_pixels  The minimum line width in pixels.

elevation_scale  Elevation multiplier. The final elevation is calculated by `elevationScale * getElevation(d)`. `elevationScale` is a handy property to scale all polygon elevation without updating the data

point_radius_scale  A global radius multiplier for all points.

point_radius_min_pixels  The minimum radius in pixels.

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

digits  number of digits for rounding coordinates

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
• strokeWdith
• 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
- `digits` - number to round the legend values to

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'
set_token(key)

## Not supplying colouring arguments, the function will try and find them in the GeoJSON
mapdeck(
  , location = c(145, -37.9)
  , zoom = 8
  , style = mapdeck_style("dark")
  , pitch = 35
) %>%
  add_geojson(
    data = geojson
    , auto_highlight = TRUE
  )

## only supplying values to use for fill, the stroke will be default
mapdeck(
  , location = c(145, -37.9)
  , zoom = 8
  , style = mapdeck_style("dark")
  , pitch = 35
) %>%
  add_geojson(
    data = geojson
    , fill_colour = "random"
  )
```
```r
mapdeck(
  , location = c(145, -37.9)
  , zoom = 8
  , style = mapdeck_style("dark")
  , pitch = 35
)

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

mapdeck(
  , 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(
  , location = c(145, -37.9)
  , zoom = 8
  , style = mapdeck_style("dark")
  , pitch = 35
)

add_geojson(
  , data = geo
)
```
add_greatcircle

Description

Renders flat arcs along the great circle joining pairs of source and target points, specified as longitude/latitude coordinates.

Usage

```r
add_greatcircle(
  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,
  wrap_longitude = FALSE,
  tooltip = NULL,
  auto_highlight = FALSE,
  highlight_colour = "#AAFFFFFF",
  legend = F,
  legend_options = NULL,
  legend_format = NULL,
  palette = "viridis",
  na_colour = "#808080FF",
  update_view = TRUE,
  focus_layer = FALSE,
  transitions = NULL,
  digits = 6
)
```

Arguments

- **map**: a mapdeck map object
- **data**: data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
- **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, and optionally an elevation column, or an sfc column
- **destination**: vector of longitude and latitude columns, and optionally an elevation column, or an sfc column
- **id**: an id value in data to identify layers when interacting in Shiny apps.
**add_greatcircle**

- **stroke_from**: column of data or hex colour to use as the starting stroke colour. If using a hex colour, use either a single value, or a vector the same length as data
- **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. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
- **stroke_to**: column of data or hex colour to use as the ending stroke colour. If using a hex colour, use either a single value, or a vector the same length as data
- **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. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
- **stroke_width**: width of the stroke in pixels
- **wrap_longitude**: logical, whether to automatically wrap longitudes over the 180th antimeridian.
- **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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")
- **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.
- **digits**: number of digits for rounding coordinates

**Details**

`add_greatcircle` 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
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
- digits - number to round the legend values to

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
set_token("MAPBOX_TOKEN")

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( style = mapdeck_style("dark"), pitch = 45 )
  %>%
  add_greatcircle(
    data = flights,
    layer_id = "greatcircle_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
  )
```
### add_grid

The `add_grid` function adds a grid heatmap to a mapdeck object. It takes an array of points and renders a grid based on the constant size of each cell. The color and height of each cell are scaled by the number of points it contains.

#### Usage

```r
add_grid(
  map,
```
data = get_map_data(map),
lon = NULL,
lat = NULL,
polyline = NULL,
cell_size = 1000,
extruded = TRUE,
elevation = NULL,
elevation_function = c("sum", "mean", "min", "max"),
colour = NULL,
colour_function = c("sum", "mean", "min", "max"),
elevation_scale = 1,
colour_range = NULL,
legend = FALSE,
legend_options = NULL,
auto_highlight = FALSE,
highlight_colour = "#AAFFFFFF",
layer_id = NULL,
update_view = TRUE,
focus_layer = FALSE,
digits = 6,
transitions = NULL,
brush_radius = NULL)

Arguments

map         a mapdeck map object
data        data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
lon          column containing longitude values
lat          column containing latitude values
polyline     optional column of data containing the polylines, if using encoded polylines
cell_size    size of each cell in meters. Default 1000
extruded     logical indicating if cells are elevated or not. Default TRUE
elevation    the height the polygon extrudes from the map. Only available if neither stroke_colour or stroke_width are supplied. Default 0
elevation_function    one of 'min', 'mean', 'max', 'sum'. IF supplied it specifies how the elevation values are calculated. Defaults to sum.
colour       column containing numeric values to colour by.
colour_function    one of 'min', 'mean', 'max', 'sum'. If supplied it specifies how the colour values are calculated. Defaults to sum.
elevation_scale    elevation multiplier.
colour_range    vector of 6 hex colours
add_grid

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.

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

digits number of digits for rounding coordinates

transitions list specifying the duration of transitions.

brush_radius radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed

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

See Also

add_hexagon

Examples

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

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( style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_grid(
```
add_grid

data = df
, lat = "lat"
, lon = "lng"
, cell_size = 5000
, elevation_scale = 50
, layer_id = "grid_layer"
, auto_highlight = TRUE
}

## using sf object
library(sfheaders)
sf <- sfheaders::sf_point( df, x = "lng", y = "lat")

mapdeck( style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_grid(
    data = sf
    , cell_size = 5000
    , elevation_scale = 50
    , layer_id = "grid_layer"
    , auto_highlight = TRUE
  )

## using colour and elevation functions, and legends
df$v <- sample(1:10, size = nrow(df), replace = T)

mapdeck( style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_grid(
    data = df
    , lat = "lat"
    , lon = "lng"
    , layer_id = "hex_layer"
    , elevation_scale = 100
    , legend = T
    , colour_function = "max"
    , colour = "v"  
  )

mapdeck( style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_grid(
    data = df
    , lat = "lat"
    , lon = "lng"
    , layer_id = "hex_layer"
    , elevation_scale = 10
    , legend = T
    , elevation_function = "mean"
    , elevation = "v"  
  )
add_heatmap

Description

The Heatmap Layer can be used to visualise spatial distribution of data. It implements Gaussian Kernel Density Estimation to render the heatmaps.

Usage

add_heatmap(
  map,
  data = get_map_data(map),
  lon = NULL,
  lat = NULL,
  polyline = NULL,
  weight = NULL,
  colour_range = NULL,
  radius_pixels = 30,
  intensity = 1,
  threshold = 0.05,
  layer_id = NULL,
  update_view = TRUE,
  focus_layer = FALSE,
  digits = 6,
  transitions = NULL
)

Arguments

map a mapdeck map object
data data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
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. Default 1
colour_range vector of 6 hex colours
radius_pixels Radius of the circle in pixels, to which the weight of an object is distributed
intensity Value that is multiplied with the total weight at a pixel to obtain the final weight. A value larger than 1 biases the output color towards the higher end of the spectrum, and a value less than 1 biases the output color towards the lower end of the spectrum
The HeatmapLayer reduces the opacity of the pixels with relatively low weight to create a fading effect at the edge. A larger threshold smoothens the boundaries of color blobs, while making pixels with low relative weight harder to spot (due to low alpha value). Threshold is defined as the ratio of the fading weight to the max weight, between 0 and 1. For example, 0.1 affects all pixels with weight under 10% of the max.

The current version of this layer is supported only for WebGL2 enabled browsers. So you may find it doesn’t render in the RStudio viewer.

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

df <- read.csv(paste0('}
### 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
add_hexagon(
  map,
  data = get_map_data(map),
  polyline = NULL,
  lon = NULL,
  lat = NULL,
  layer_id = NULL,
  radius = 1000,
)```

#### Examples

```r
df <- df[ !is.na(df$lng), ]
df$weight <- sample(1:10, size = nrow(df), replace = T)

mapdeck( style = mapdeck_style('dark'), pitch = 45 ) %>%
  add_heatmap(  
data = df  
  , lat = "lat"  
  , lon = "lng"  
  , weight = "weight",  
  , layer_id = "heatmap_layer"  
)  

## as an sf object
library(sfheaders)
sf <- sfheaders::sf_point( df, x = "lng", y = "lat")

mapdeck( style = mapdeck_style('dark'), pitch = 45 ) %>%
  add_heatmap(  
data = sf  
  , weight = "weight",  
  , layer_id = "heatmap_layer"  
)```
elevation = NULL,
elevation_function = c("sum", "mean", "min", "max"),
colour = NULL,
colour_function = c("sum", "mean", "min", "max"),
legend = FALSE,
legend_options = NULL,
elevation_scale = 1,
auto_highlight = FALSE,
highlight_colour = "#AFFFFFF",
colour_range = NULL,
update_view = TRUE,
focus_layer = FALSE,
digits = 6,
transitions = NULL,
brush_radius = NULL
)

Arguments

map a mapdeck map object
data data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
polyline column of data containing the polylines
lon column containing longitude values
lat column containing latitude values
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
radius in metres. Default 1000
elevation column containing the elevation of the value.
elevation_function one of 'min', 'mean', 'max', 'sum'. IF supplied it specifies how the elevation values are calculated. Defaults to sum.
colour column containing numeric values to colour by.
colour_function one of 'min', 'mean', 'max', 'sum'. If supplied it specifies how the colour values are calculated. Defaults to sum.
legend logical indicating if a legend should be displayed
legend_options A list of options for controlling the legend.
elevation_scale value to scale the elevations of the hexagons. Default 1
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.
```
colour_range vector of 6 hex colours
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
digits number of digits for rounding coordinates
transitions list specifying the duration of transitions.
brush_radius radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed

Details

add_hexagon 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 hexagon

list( elevation = 0 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

Examples

```
## Not run:

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

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( style = mapdeck_style("dark"), pitch = 45 ) %>%
add_hexagon( data = df , lat = "lat"
```
```r
add_hexagon

library(sfheaders)
sf <- sfheaders::sf_point(df, x = "lng", y = "lat")

mapdeck(style = mapdeck_style("dark"), pitch = 45) %>%
  add_hexagon(
    data = sf,
    layer_id = "hex_layer",
    elevation_scale = 100
  )

## Using elevation and colour
df$colour <- rnorm(nrow(df))
df$elevation <- rnorm(nrow(df))

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

mapdeck(style = mapdeck_style("dark"), pitch = 45) %>%
  add_hexagon(
    data = df,
    lat = "lat",
    lon = "lng",
    layer_id = "hex_layer",
    elevation_scale = 100,
    elevation = "weight",
    elevation_function = "mean",
    colour = "colour",
    colour_function = "mean"
  )

## with a legend
df$val <- sample(1:10, size = nrow(df), replace = T)

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

Description

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

Usage

```r
ddf_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 = "#AAFFFFFF",
  palette = "viridis",
  na_colour = "#808080FF",
  legend = FALSE,
  legend_options = NULL,
  legend_format = NULL,
  update_view = TRUE,
  focus_layer = FALSE,
  digits = 6,
  transitions = NULL,
  brush_radius = NULL
)
```

Arguments

- `map` a mapdeck map object.
data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system.

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, and optionally an elevation column, or an sfc column.

destination vector of longitude and latitude columns, and optionally an elevation column, 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.

stroke_width width of the line in metres.

stroke_opacity Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string.

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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")

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.

digits number of digits for rounding coordinates.

transitions list specifying the duration of transitions.

brush_radius radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed.

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.
add_line

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
- digits - number to round the legend values to

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'
set_token( key )

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

sf_flights <- sfheaders::sf_point( flights, x = "start_lon", y = "start_lat", keep = TRUE )
destination <- sfheaders::sfc_point( flights, x = "end_lon", y = "end_lat" )

sf_flights$destination <- destination

mapdeck() %>%
  add_line(
    data = sf_flights
    , origin = 'geometry'
    , destination = 'destination'
    , layer_id = 'arcs'
    , stroke_colour = "airport1"
  )
```

---

**add_mesh**

**Add Mesh**

**Description**

Adds polygons to the map from a mesh3d object

**Usage**

```r
add_mesh(
  map,
  data = get_map_data(map),
  fill_opacity = NULL,
  elevation = NULL,
  tooltip = NULL,
  auto_highlight = FALSE,
  highlight_colour = "#AFFFFFF",
  light_settings = list(),
  layer_id = NULL,
```
add_mesh

id = NULL,
palette = "viridis",
na_colour = "#808080FF",
legend = FALSE,
legend_options = NULL,
legend_format = NULL,
update_view = TRUE,
focus_layer = FALSE,
digits = 6,
transitions = NULL,
brush_radius = NULL
)

Arguments

map

a mapdeck map object

data

data to be used in the layer. All coordinates are expected to be EPSG:4326
(WGS 84) coordinate system

fill_opacity

Either a string specifying the column of data containing the opacity of each
shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes.
Default 255. If a hex-string is used as the colour, this argument is ignored and
you should include the alpha on the hex string

elevation

the height the polygon extrudes from the map. Only available if neither stroke_colour
or stroke_width are supplied. Default 0

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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255],
where the 4th column represents the alpha. You can use a named list
to specify a different palette for different colour options (where available), e.g.
list(fill_colour = "viridis", stroke_colour = "inferno")

na_colour

hex string colour to use for NA values

legend

either a logical indiciating 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
digits  number of digits for rounding coordinates
transitions  list specifying the duration of transitions.
brush_radius  radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed

Details

add_mesh supports mesh3d objects

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
• digits - number to round the legend values to

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
## exaggerate the elevation slightly
m <- melbourne_mesh
m$vbb[3, ] <- m$vbb[3, ] * 50

mapdeck() %>%
  add_mesh(
    data = m
```
add_path

Arguments

map  a mapdeck map object

data  data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system

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

Description

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

Usage

```r
add_path(
  map,
  data = get_map_data(map),
  polyline = NULL,
  stroke_colour = NULL,
  stroke_width = NULL,
  stroke_opacity = NULL,
  dash_size = NULL,
  dash_gap = NULL,
  tooltip = NULL,
  billboard = FALSE,
  layer_id = NULL,
  id = NULL,
  auto_highlight = FALSE,
  highlight_colour = "#AAFFFFFF",
  palette = "viridis",
  na_colour = "#808080FF",
  legend = FALSE,
  legend_options = NULL,
  legend_format = NULL,
  update_view = TRUE,
  focus_layer = FALSE,
  digits = 6,
  transitions = NULL,
  brush_radius = NULL
)
```
stroke_colour variable of data or hex colour for the stroke. If used, elevation is ignored. If using a hex colour, use either a single value, or a vector the same length as data
stroke_width width of the stroke in meters. Default 1.
stroke_opacity Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
dash_size size of each dash, relative to the width of the stroke
dash_gap size of the gap between dashes, relative to the width of the stroke
tooltip variable of data containing text or HTML to render as a tooltip
billboard logical indicating if the path always faces the camera (TRUE) or if it always faces up (FALSE)
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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")
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
digits number of digits for rounding coordinates
transitions list specifying the duration of transitions.
brush_radius radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed

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
- digits - number to round the legend values to

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'
set_token(key)

mapdeck(
  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
  )
```

---

### 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 = "#AAFFFFFF",
  light_settings = list(),
  layer_id = NULL,
  id = NULL,
  palette = "viridis",
```
add_pointcloud

na_colour = "#808080FF",
legend = FALSE,
legend_options = NULL,
legend_format = NULL,
update_view = TRUE,
focus_layer = FALSE,
digits = 6,
transitions = NULL,
brush_radius = NULL
)

Arguments

map a mapdeck map object
data data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
lon column containing longitude values
lat column containing latitude values
elevation column containing the elevation values. Default 0
polyline optional column of data containing the polylines, if using encoded polylines
radius value in pixels of each point. Default 10.
fill_colour column of data or hex colour for the fill colour. If using a hex colour, use either a single value, or a vector the same length as data
fill_opacity Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")
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.
add_pointcloud

- **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.
- **digits**: number of digits for rounding coordinates.
- **transitions**: list specifying the duration of transitions.
- **brush_radius**: radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed.

**Details**

add_pointcloud supports POINT and MULTIPONT 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 pointcloud

```r
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
- `digits`: number to round the legend values to

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'
set_token(key)

df <- capitals
df$z <- sample(10000:1000000, size = nrow(df))

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

## as an sf object with a Z attribute
library(sfheaders)
sf <- sfheaders::sf_point(df, x = "lon", y = "lat", z = "z")

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

Description

The Polygon Layer renders filled and/or stroked polygons.
add_polygon

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_opacity = NULL,
  elevation = NULL,
  tooltip = NULL,
  auto_highlight = FALSE,
  elevation_scale = 1,
  highlight_colour = "#AAFFFFFF",
  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,
  digits = 6,
  transitions = NULL,
  brush_radius = NULL
)
```

Arguments

- **map**: a mapdeck map object
- **data**: data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
- **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. If using a hex colour, use either a single value, or a vector the same length as data
- **stroke_width**: width of the stroke in meters. If used, elevation is ignored. Default 1.
- **stroke_opacity**: Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
- **fill_colour**: column of data or hex colour for the fill colour. If using a hex colour, use either a single value, or a vector the same length as data
fill_opacity
Either a string specifying the column of data containing the opacity of each
shape, or a single value in \([0,255]\), or \([0, 1)\), to be applied to all the shapes.
Default 255. If a hex-string is used as the colour, this argument is ignored and
you should include the alpha on the hex string.

elevation
The height the polygon extrudes from the map. Only available if neither stroke_colour
or stroke_width are supplied. Default 0.

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.

elevation_scale
elevation multiplier.

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 must have at least 5 rows, and 3 or 4 columns of values between \([0, 255]\),
where the 4th column represents the alpha. You can use a named list
to specify a different palette for different colour options (where available), e.g.
list(fill_colour = "viridis", stroke_colour = "inferno")

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.

digits
Number of digits for rounding coordinates.

transitions
List specifying the duration of transitions.

brush_radius
Radius of the brush in metres. Default NULL. If supplied, the arcs will only
show if the origin or destination are within the radius of the mouse. If NULL,
all arcs are displayed.

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
- digits - number to round the legend values to

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'
set_token(key)

library(geojsonsf)

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

mapdeck()
```
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,
  radius_min_pixels = 1,
  radius_max_pixels = NULL,
  fill_colour = NULL,
)```
Arguments

map a mapdeck map object

data data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system

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. Default 1

radius_min_pixels the minimum radius in pixels. Can prevent circle from getting too small when zoomed out small for the given zoom level

radius_max_pixels the maximum radius in pixels. Can prevent the circle from getting too big when zoomed in

fill_colour column of data or hex colour for the fill colour. If using a hex colour, use either a single value, or a vector the same length as data

fill_opacity Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string

stroke_colour variable of data or hex colour for the stroke. If used, elevation is ignored. If using a hex colour, use either a single value, or a vector the same length as data

stroke_width width of the stroke in meters. If used, elevation is ignored. Default 1.
stroke_opacity

Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0, 1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string.

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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")

na_colour

hex string colour to use for NA values

legend

either a logical indiciating 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

digits

number of digits for rounding coordinates

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.

brush_radius

radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed

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
- `digits` - number to round the legend values to

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
define access token from Mapbox
key <- 'abc'
set_token(key)

mapdeck(style = mapdeck_style("dark"), pitch = 45) %>%
add_scatterplot(
  data = capitals,
  lat = "lat",
  lon = "lon",
  radius = 100000,
  fill_colour = "country",
  layer_id = "scatter_layer"
)```
```r
add_scatterplot

, tooltip = "capital"
)

## using legend options
mapdeck( style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_scatterplot(
    data = capitals
    , lat = "lat"
    , lon = "lon"
    , radius = 100000
    , fill_colour = "lon"
    , stroke_colour = "lat"
    , layer_id = "scatter_layer"
    , tooltip = "capital"
    , legend = T
    , legend_options = list( digits = 5 )
)

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(style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_scatterplot(
    data = df
    , lat = "lat"
    , lon = "lng"
    , layer_id = "scatter_layer"
    , stroke_colour = "lng"
  )

## as an sf object
library(sfheaders)
sf <- sfheaders::sf_point( df, x = "lng", y = "lat")

mapdeck( style = mapdeck_style("dark"), pitch = 45 ) %>%
  add_scatterplot(
    data = sf
    , radius = 100
    , fill_colour = "country"
    , layer_id = "scatter_layer"
    , tooltip = "capital"
  )
```
add_screengrid

Description

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

add_screengrid(
  map,
  data = get_map_data(map),
  lon = NULL,
  lat = NULL,
  polyline = NULL,
  weight = NULL,
  aggregation = c("sum", "mean", "min", "max"),
  colour_range = NULL,
  opacity = 0.8,
  cell_size = 50,
  layer_id = NULL,
  update_view = TRUE,
  focus_layer = FALSE,
  digits = 6
)

Arguments

map a mapdeck map object

data data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system

lon column containing longitude values

lat column containing latitude values

data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system

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

weight the weight of each value. Default 1

aggregation one of 'min', 'mean', 'max', 'sum'. If supplied it specifies how the weights used.

colour_range vector of 6 hex colours

opacity opacity of cells. Value between 0 and 1. Default 0.8

cell_size size of grid squares in pixels. Default 50

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
add_screengrid

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
digits: number of digits for rounding coordinates

Details

add_screengrid supports POINT and MULTIPONT 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'
set_token( key )

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( 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(sfheaders)
sf <- sfheaders::sf_point( df, x = "lng", y = "lat"

mapdeck( style = mapdeck_style('dark'), pitch = 45 ) %>%
  add_screengrid(
    data = sf
    , weight = "weight"
    , layer_id = "screengrid_layer"
```
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 EPSG:4326 (WGS 84) coordinate system
- `...` 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 MULTIPOINT 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,
  billboard = TRUE,
  font_family = "Monaco, monospace",
  font_weight = "normal",
  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,
  digits = 6,
  transitions = NULL,
  brush_radius = NULL
)
```

Arguments
- `map` a mapdeck map object
- `data` data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system.
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. If using a hex colour, use either a single value, or a vector the same length as data
fill_opacity Either a string specifying the column of data containing the opacity of each shape, or a single value in [0,255], or [0,1), to be applied to all the shapes. Default 255. If a hex-string is used as the colour, this argument is ignored and you should include the alpha on the hex string
size column of data containing the size of the text. Default 32
angle column of data containing the angle of the text. Default 0
anchor column of data containing the anchor of the text. One of 'start', 'middle' or 'end'
alignment_baseline column of data containing the alignment. One of 'top', 'center' or 'bottom'
billboard logical indicating if the text always faces the camera (TRUE) or if it always faces up (FALSE)
font_family specifies a prioritised list of one or more font family names and/or generic family names. Follow the specifics for CSS font-family https://developer.mozilla.org/en-US/docs/Web/CSS/font-family
font_weight specifies the font weight. Follow the specifics for CSS font-weight https://htmldog.com/references/css/properties/font-weight/
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 must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")
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
digits number of digits for rounding coordinates
transitions list specifying the duration of transitions.
brush_radius radius of the brush in metres. Default NULL. If supplied, the arcs will only show if the origin or destination are within the radius of the mouse. If NULL, all arcs are displayed

Details

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
• digits - number to round the legend values to

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'
set_token(key)

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

### Description

Adds a title to a map

#### Usage

```r
add_title(map, title, layer_id = NULL)
```

#### Arguments

- **map**: A mapdeck map object
- **title**: Either a single string for the title, or a list with a 'title' element, and an optional 'css' element. See examples
- **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

#### Examples

```r
mapdeck() %>%
  add_title(title = "first title", layer_id = "first") %>%
```
add_trips

add_trips

Description

The Trips Layer takes an sf object with Z (elevation) and M (time) attributes and renders it as animated trips.

Usage

```r
add_trips(
  map,
  data = get_map_data(map),
  stroke_colour = NULL,
  stroke_width = NULL,
  opacity = 0.3,
  palette = "viridis",
  trail_length = 180,
  start_time = get_m_range_start(data),
  end_time = get_m_range_end(data),
  animation_speed = 30,
  layer_id = NULL,
  legend = FALSE,
  legend_options = NULL,
  legend_format = NULL,
  digits = 6
)
```

Arguments

- **map**: a mapdeck map object
- **data**: sf object with XYZM dimensions.
- **stroke_colour**: variable of data or hex colour for the stroke.
- **stroke_width**: width of the stroke in meters. Default 1.
- **opacity**: single value in [0,1]
palette string or matrix. String will be one of colourvalues::colour_palettes(). A matrix must have at least 5 rows, and 3 or 4 columns of values between [0, 255], where the 4th column represents the alpha. You can use a named list to specify a different palette for different colour options (where available), e.g. list(fill_colour = "viridis", stroke_colour = "inferno")

trail_length how long it takes for the trail to completely fade out (in same units as timestamps )

start_time the minimum timestamp

end_time the maximum timestamp

animation_speed speed of animation

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

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

digits number of digits for rounding coordinates

Details

add_trips supports LINESTRING and MULTILINESTRING sf objects

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
• digits - number to round the legend values to

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.
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
set_token( "MAPBOX_TOKEN"
sf <- city_trail

mapdeck(
  location = c(145, -37.8)
  , zoom = 10
  , style = mapdeck_style("dark")
) %>%
add_trips(
  data = sf
  , animation_speed = 2000
  , trail_length = 1000
  , stroke_colour = "#FFFFFF"
)
```

**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
**city_trail**

**Description**

An sf object of a cyclist cycling around Melbourne’s Capital City Trail

**Usage**

`city_trail`

**Format**

An object of class sf (inherits from `data.frame`) with 1 rows and 3 columns.

---

**clear_arc**

**Clear Arc**

**Description**

Clears elements from a map

**Usage**

```r
clear_arc(map, layer_id = NULL)
clear_bitmap(map, layer_id = NULL)
clear_column(map, layer_id = NULL)
clear_geojson(map, layer_id = NULL)
clear_greatcircle(map, layer_id = NULL)
clear_grid(map, layer_id = NULL)
clear_heatmap(map, layer_id = NULL)
clear_hexagon(map, layer_id = NULL)
clear_line(map, layer_id = NULL)
clear_mesh(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)
clear_title(map, layer_id = NULL)
clear_trips(map, layer_id = NULL)

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

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
a 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

Description
Clears the access tokens

Usage
clear_tokens()
geojson  

**Description**

A GeoJSON object of polygons, lines and points in Melbourne

**Usage**

```r
geojson
```

**Format**

- a 'json' object

---

legend_element  

**Description**

Creates a mapdeck legend element for when you want to manually specify a legend (using `mapdeck_legend`)

**Usage**

```r
legend_element(
  variables,
  colours,
  colour_type = c("fill", "stroke"),
  variable_type = c("category", "gradient"),
  title = "",
  css = ""
)
```

**Arguments**

- `variables` variables assigned to colours
- `colours` vector of hex colours assigned to variables
- `colour_type` one of "fill" or "stroke"
- `variable_type` one of category (discrete) or gradient (continuous)
- `title` string used as the legend title
- `css` string of css to control appearance.

**See Also**

`mapdeck_legend`
Examples

```r
l1 <- legend_element(
  variables = c("a","b"),
  colours = c("#00FF00","#FF0000"),
  colour_type = "fill",
  variable_type = "category",
  title = "my title"
)
```

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 3x 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
)
```

Description

mapdeck
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),
  show_view_state = FALSE
)

Arguments

data: data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system

token: Mapbox Access token. Use set_token() or Sys.setenv() to set a global token. See Access Tokens section for further details. 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 (see mapdeck_style)

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)

show_view_state: logical, indicating whether to add the current View State to the map. When TRUE, the following is added as an overlay to the map

  • width
  • height
  • latitude & longitude
  • zoom
  • bearing
  • pitch
  • altitude
  • viewBounds
  • interactionState
Access Tokens

If the token argument is not used, the map will search for the token, firstly by checking if `set_token()` was used, then it will search environment variables using `Sys.getenv()` and the following values, in this order:

c("MAPBOX_TOKEN", "MAPBOX_KEY", "MAPBOX_API_TOKEN", "MAPBOX_API_KEY", "MAPBOX", "MAPDECK")

If multiple tokens are found, the first one is used.

Description

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

Usage

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

```r
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 Dependencies

Description

Adds the required mapdeck javascript dependencies to a Shiny UI when you want to use mapdeck layers, but not with a mapdeck map.

Usage

```r
mapdeck_dependencies()
```
mapdeck_dispatch  

Description

Extension points for plugins

Usage

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

dispatch.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

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

mapdeck_legend

Description

Constructs legend elements into the correct JSON format for plotting on the map

Usage

mapdeck_legend(legend_elements)
Arguments

legend_elements

vector of legend elements (made from legend_element)

See Also

legend_element

Examples

l1 <- legend_element(
  variables = c("a","b"),
  colours = c("#00FF00","FF0000"),
  colour_type = "fill",
  variable_type = "category",
  title = "my title"
)

mapdeck_legend(l1)

mapdeck_style

Mapdeck Style

Description

Various styles available to all Mapbox accounts using a valid access token. Available styles are listed at https://docs.mapbox.com/api/maps/#styles.

Usage

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

Arguments

style one of streets, outdoors, light, dark, satellite, satellite-streets

Examples

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

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

**Description**

Retrieves the mapdeck token that has been set

**Usage**

```r
mapdeck_tokens()
```

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(
  data = NULL,
  map_id,
  session = shiny::getDefaultReactiveDomain(),
  deferUntilFlush = TRUE,
  map_type = c("mapdeck_update", "google_map_update")
)
```

**Arguments**

- `data`  
  data to be used in the layer. All coordinates are expected to be EPSG:4326 (WGS 84) coordinate system
- `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.
- `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.
- `map_type`  
  either mapdeck_update or google_map_update
Description

Changes the view of the map.

Usage

```r
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

Description

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

Usage

```r
melbourne
```

Format

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

**Description**

A mesh3d object of Melbourne

**Usage**

`melbourne_mesh`

**Format**

An object of class mesh3d (inherits from shape3d) of length 6.

---

roads  
*Roads in central Melbourne*

**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 [https://www.data.gov.au](https://www.data.gov.au) and distributed under the Creative Commons 4 License
[https://creativecommons.org/licenses/by/4.0/]
**set_token**

**Set Token**

**Description**

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

**Usage**

```python
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.

---

**update_style**

**update style**

**Description**

update style

**Usage**

```python
update_style(map, style)
```

**Arguments**

- `map` (a mapdeck map object)
- `style` (the style of the map (see `mapdeck_style`)
Pipe

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

Arguments
lhs, 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, 63
- city_trail, 64
- geojson, 66
- melbourne, 73
- melbourne_mesh, 74
- roads, 74
- clear_hexagon (clear_arc), 64
- clear_legend, 65
- clear_line (clear_arc), 64
- clear_mesh (clear_arc), 64
- clear_path (clear_arc), 64
- clear_pointcloud (clear_arc), 64
- clear_polygon (clear_arc), 64
- clear_scatterplot (clear_arc), 64
- clear_screengrid (clear_arc), 64
- clear_text (clear_arc), 64
- clear_title (clear_arc), 64
- clear_tokens, 65
- clear_trips (clear_arc), 64
- geojson, 66
- invoke_method (mapdeck_dispatch), 70
- legend_element, 66, 71
- light_settings, 15, 37, 43, 47, 67
- mapdeck, 67, 70
- mapdeck_shiny, 69
- mapdeck_dependencies, 69
- mapdeck_dispatch, 70
- mapdeck_legend, 66, 70
- mapdeck_style, 68, 71, 75
- mapdeck_tokens, 72
- mapdeck_update, 70, 72
- mapdeck_view, 73
- mapdeckOutput (mapdeck_shiny), 69
- melbourne, 73
- melbourne_mesh, 74
- renderMapdeck (mapdeck_shiny), 69
- roads, 74
- set_token, 75
- update_style, 75