Package 'rDeckgl'

June 1, 2026

Type Package

Title R Bindings to 'Deck.gl'

Version 0.1.0

Description Provides R bindings for 'deck.gl', a 'WebGL' framework for rendering large interactive spatial and tabular visualizations. The package supplies 'htmlwidgets' and 'shiny' bindings, supports 'DuckDB'-backed data hydration, and bundles the JavaScript assets needed to render 'deck.gl' specifications from R.

License MIT + file LICENSE

URL https://github.com/TiRizvanov/rDeckgl

BugReports https://github.com/TiRizvanov/rDeckgl/issues

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports htmlwidgets (>= 1.5.4), shiny (>= 1.7.0), jsonlite (>= 1.8.0), yaml (>= 2.3.0), DBI (>= 1.1.0), duckdb (>= 1.4.0), arrow (>= 12.0.0), base64enc (>= 0.1.3), stats

Suggests adbcdrivermanager, nanoarrow, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Timur Rizvanov [aut, cre],
Edward C. Ruiz [aut],
Ruben Dries [aut, rev],
Dries Lab [fnd] (Host laboratory and funding source),
Boston University [fnd] (Undergraduate Research Opportunities Program (UROP))

Maintainer Timur Rizvanov <timurr@bu.edu>

Repository CRAN

Date/Publication 2026-06-01 13:40:02 UTC

Contents

`deckgl` *Render a Deck.gl visualization*

Description

Creates an interactive deck.gl visualization from a JSON or YAML specification. Supports server-side data hydration via DuckDB for efficient data handling.

Usage

```
deckgl(
  spec,
  specType = c("auto", "json", "yaml"),
  data = NULL,
  con = NULL,
  data_transport = c("auto", "file", "inline"),
  data_dir = NULL,
  width = NULL,
  height = NULL
)
```

Arguments

`spec`	Deck.gl specification as an R list, JSON text, JSON file path, YAML text, or YAML file path.
`specType`	One of "auto" (default), "json", or "yaml". Auto-detection attempts to infer the format from the input.
`data`	Named list of data.frames to register in DuckDB. These tables can be referenced in the spec using ‘type = "duckdb"‘ data nodes.
`con`	Optional DuckDB connection to use for queries. If provided, this connection will be used instead of creating a new one. This is useful for GeoArrow workflows where you need spatial extension and geometry tables already set up.

data_transport	How hydrated Arrow/Parquet query results are delivered to the browser. ‘"auto"‘ uses ‘"file"‘ when ‘data_dir‘ is supplied and otherwise falls back to ‘"inline"‘ for portable widgets; ‘"inline"‘ embeds base64 payloads in the widget; ‘"file"‘ writes binary files to ‘data_dir‘ and uses relative URLs.
data_dir	Directory for ‘"file"‘ transport. Serve or save the widget from the same directory so relative URLs resolve.
width	CSS or pixel width (e.g. "100%", "600px", or numeric).
height	CSS or pixel height (e.g. "100%", "600px", or numeric).

Value

An htmlwidget that renders the Deck.gl visualization.

Examples

```
if (interactive()) {
  # Simple scatterplot with inline data
  spec <- list(
    `@type` = "DeckGL",
    initialViewState = list(
      longitude = -122.4,
      latitude = 37.76,
      zoom = 12,
      pitch = 0,
      bearing = 0
    ),
    layers = list(
      list(
        `@type` = "ScatterplotLayer",
        id = "scatterplot",
        data = list(
          type = "duckdb",
          query = "SELECT lon, lat, radius FROM points"
        ),
        getPosition = "@=[lon, lat]",
        getRadius = "@=radius",
        getFillColor = c(255, 0, 0)
      )
    )
  )

  data <- list(
    points = data.frame(
      lon = c(-122.4, -122.45, -122.35),
      lat = c(37.76, 37.78, 37.74),
      radius = c(100, 150, 200)
    )
  )

  deckgl(spec = spec, data = data)
}
```

deckglOutput *Shiny output for Deck.gl widget*

Description

Use this in the UI to create a placeholder for the Deck.gl visualization.

Usage

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

Arguments

outputId	output variable name
width, height	CSS dimensions (e.g. '100%', '400px') for the container.

Value

A Shiny output binding for use in the UI.

Examples

```
if (interactive()) {
  library(shiny)
  library(rDeckgl)

  ui <- fluidPage(
    deckglOutput("myDeckgl", height = "600px")
  )

  server <- function(input, output, session) {
    output$myDeckgl <- renderDeckgl({
      deckgl(spec = my_spec, data = my_data)
    })
  }

  shinyApp(ui, server)
}
```

ggsql *Render a ggsql query as a Deck.gl visualization*

Description

'ggsql()' lets you describe a Deck.gl visualization using the ggsql dialect (VISUALIZE / DRAW / PLACE / SCALE / LABEL / SETTING) instead of the native Deck.gl spec. The parser is shared with rMosaic; this entry point compiles for the Deck.gl rendering path, which is the right choice for spatial layers (polygons, hex grids, big point clouds, GeoArrow data).

Usage

```
ggsql(sql, data = NULL, con = NULL, width = NULL, height = NULL, ...)
```

Arguments

sql	A character scalar containing ggsql.
data	Optional named list of data.frames to register in the widget's DuckDB before rendering. Use this when the FROM source in the SQL is not already a table the widget's DuckDB can see.
con	Optional DuckDB connection. If supplied it is reused instead of a fresh one (mirrors the 'deckgl()' argument). Useful for GiottoDB / dbProject workflows.
width, height	Optional widget dimensions.
...	Reserved for forward compatibility.

Value

An htmlwidget produced by [deckgl()].

See Also

[deckgl()].

renderDeckgl *Shiny render function for Deck.gl*

Description

Use this in the server to render the Deck.gl visualization to the output.

Usage

```
renderDeckgl(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr	An expression that generates a call to deckgl().
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

Value

A Shiny render function for use in the server.

Examples

```
if (interactive()) {
  library(shiny)
  library(rDeckgl)

  ui <- fluidPage(
    deckglOutput("myDeckgl")
  )

  server <- function(input, output, session) {
    output$myDeckgl <- renderDeckgl({
      spec <- list(
        `@type` = "DeckGL",
        initialViewState = list(longitude = -122.4, latitude = 37.76, zoom = 12),
        layers = list(
          list(
            `@type` = "ScatterplotLayer",
            id = "points",
            data = list(type = "duckdb", query = "SELECT * FROM points"),
            getPosition = "@=[lon, lat]",
            getRadius = 100,
            getFillColor = c(255, 0, 0)
          )
        )
      )
      deckgl(spec = spec, data = list(points = data.frame(lon = -122.4, lat = 37.76)))
    })
  }

  shinyApp(ui, server)
}
```

Index