Package ‘shinyaframe’

November 26, 2017

Type Package

Title 'WebVR' Data Visualizations with 'RStudio Shiny' and 'Mozilla A-Frame'

Version 1.0.1


License AGPL-3

Encoding UTF-8

LazyData true

Imports shiny, htmlwidgets, htmltools

RoxygenNote 5.0.1

Suggests knitr, rmarkdown, scales, dplyr

VignetteBuilder knitr

NeedsCompilation no

Author William Murphy [cre, aut]

Maintainer William Murphy <william.murphy.rd@gmail.com>

Repository CRAN

Date/Publication 2017-11-26 15:29:43 UTC

R topics documented:

aDataScene ................................................................. 2
aDataScene-shiny ......................................................... 3
aframetags ................................................................. 4
shinyaframe ............................................................... 6

Index 9
aDataScene  

A-Frame Scene with R data

Description
Create an HTML widget to sync R data with an A-Frame scene via the data-binding A-Frame component.

Usage
aDataScene(data, elementId = NULL)

Arguments
- **data**: A data frame or a list of vectors, matrices, and/or data frames
- **elementId**: Optionally define the output HTML element id

Details
Data will be synced to the data-binding system from the `gg-aframe` JavaScript library for A-Frame. Data can be bound to automatically update components in the scene with the data-binding A-Frame component. Repeat calls (e.g. within a Shiny reactive expression) will update the data-binding store and refresh bound components with the new data.

If `data` is a data frame, each variable will be available, by name, as a JavaScript Array in the scene data store (i.e. long form). If it is a list, each list item will be available, by name, as a JavaScript Array in the scene data store. Data frames within the list will be available as an array of Objects, with each object representing a row from the data frame (i.e. wide form).

To send multiple data frames in long form, combine them with `c`, and each column will be available by name in the A-Frame data-binding system. To send multiple data frames in wide form, combine them as named items in a `list`, and each data frame will be available as an array of objects (rows) under the name used.

Note: `aDataScene` is only compatible for use in Shiny apps viewed with a modern Web browser and internet connection. WebVR data visualizations are not available in Rmd documents, R Notebooks, or the RStudio Viewer at this time.

See Also
- `renderADataScene`

Examples
```r
library(dplyr)
library(scales)

# Execute within a renderADataScene call in a Shiny server
iris %>%
  # scale positional data to (0,1)
```
mutate_if(is.numeric, rescale) %>%
  # make data available in JavaScript
  aDataScene()

aDataScene-shiny

Shiny bindings for aDataScene

Description
Output and render functions for using aDataScene within Shiny applications.

Usage
aDataSceneOutput(outputId, ..., skipDependencies = FALSE)
renderADataScene(expr, env = parent.frame(), quoted = FALSE)

Arguments
outputId          output variable to read from
...               Attributes, A-Frame components, and/or child elements for output in HTML.
skipDependencies  Option to omit packaged A-Frame JavaScript libraries. See details.
expr              An expression that returns a call to aDataScene
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.

Details
A-Frame v0.7.1, gg-aframe v0.2.3, and aframe-environment-component v1.0.0 come packaged with shinyaframe. To use different versions, set skipDependencies to TRUE and source the libraries directly (for example with includeScript or tag).

See Also
'gg-aframe' syntax documentation

Examples
# Simple 3D scatterplot.
# See package vignette for additional aesthetics, guides, and legends
if (interactive()) {
  library(dplyr)
  library(shiny)
  library(scales)
shinyApp(
  ui = fluidPage(
    aDataSceneOutput,
      outputId = "mydatascene",
      # gg-aframe plot syntax
    atags$entity(
      plot = "", position = "0 1.6 -1.38", rotation = "0 45 0",
      atags$entity(
        'layer-point' = "", 'mod-oscillate' = "",
        'data-binding__sepal.length'="target: layer-point.x",
        'data-binding__sepal.width'="target: layer-point.y",
        'data-binding__petal.length'="target: layer-point.z",
        # add 4th positional by animating y position between two mappings
        'data-binding__petal.width'="target: mod-oscillate.y",
        'data-binding__species'="target: layer-point.shape"
      )
    )
  ),
  server = function(input, output, session) {
    output$mydatascene <- renderADataScene({
      names(iris) <- tolower(names(iris))
      iris %>%
      # scale positional data
      mutate_if(is.numeric, rescale) %>%
      aDataScene()
    )
  }
)
)

---

aframetags  

**A-Frame Custom Elements**

**Description**

Functions to output A-Frame’s custom HTML elements

**Usage**

aframeScene(...)
aframeAssets(...)
aframeMixin(...)
aframeEntity(...)

---
### aframeSphere

```javascript
aframeSphere(...)  
```

### aframeBox

```javascript
aframeBox(...)  
```

### aframePrimitive

```javascript
aframePrimitive(primitive = "entity", ...)  
```

### Arguments

- ...: Attributes, components, and/or child elements
- primitive: Primitive name (excluding the "a-")

### Format

The `atags` list contains all of these tag functions for convenient access.

### Details

These functions are just simple wrappers for `tag` to output common A-Frame custom elements.

### Functions

- **aframeScene**: Top level scene entity
- **aframeAssets**: Specify assets for pre-loading
- **aframeMixin**: Reusable component specifications
- **aframeEntity**: Generic entity
- **aframeSphere**: Sphere primitive
- **aframeBox**: Box primitive
- **aframePrimitive**: All other primitives

### See Also

- A-Frame Documentation

### Examples

```javascript
# Construct A-Frame HTML syntax for a 3D scene with a red box and blue sky
atags$scene(
    atags$box(color = "red", position = "0 0.5 -3"),
    atags$other("sky", color = "#89b6ff")
)
```
shinyaframe  WebVR Data Visualizations

Description


Examples

# Example Shiny app from package vignette
if (interactive()) {
  library(shiny)
  library(dplyr)
  library(scales)
  library(shinyaframe)

  shinyApp(
    ui = fluidPage(
      aDataSceneOutput(
        # attributes and child elements provided as arguments
        # server output variable name
        outputId = "mydata-scene",
        # add backdrop
        environment = "",
        # gg-aframe plot syntax
        atags$entity(
          # an empty string sets attributes with no additional properties
          plot = "",
          # sizable scale option uses polyhedra scaled for equivalent volumes
          "scale-shape" = "sizable",
          position = "0 1.6 -1.38",
          atags$entity(
            "layer-point" = "",
            "data-binding__sepal.length"="target: layer-point.x",
            "data-binding__sepal.width"="target: layer-point.y",
            "data-binding__petal.length"="target: layer-point.z",
            "data-binding__species"="target: layer-point.shape",
            "data-binding__petal.width.size"="target: layer-point.size",
            "data-binding__species.color"="target: layer-point.color"
          ),
          atags$entity(
            "guide-axis" = "axis: x",
            "data-binding__xbreaks" = "target: guide-axis.breaks",
            "data-binding__xlabels" = "target: guide-axis.labels",
            "data-binding__xtitle" = "target: guide-axis.title"
          ),
          atags$entity(
            "guide-axis" = "axis: y",
            "data-binding__ybreaks" = "target: guide-axis.breaks",
            "data-binding__ylabels" = "target: guide-axis.labels",
            "data-binding__ytitle" = "target: guide-axis.title"
          )
        )
      )
  )
}
`data-binding__ybreaks` = "target: guide-axis.breaks",
`data-binding__ylabels` = "target: guide-axis.labels",
`data-binding__ytitle` = "target: guide-axis.title"
);

atags$entity(  
  `guide-axis` = "axis: z",
  `data-binding__zbreaks` = "target: guide-axis.breaks",
  `data-binding__zlabels` = "target: guide-axis.labels",
  `data-binding__ztitle` = "target: guide-axis.title"
);

atags$entity(  
  `guide-legend` = "aesthetic: shape",
  `data-binding__shapetitle` = "target: guide-legend.title"
);

atags$entity(  
  `guide-legend` = "aesthetic: size",
  `data-binding__sizebreaks` = "target: guide-legend.breaks",
  `data-binding__sizelabels` = "target: guide-legend.labels",
  `data-binding__sizetitle` = "target: guide-legend.title"
);

atags$entity(  
  `guide-legend` = "aesthetic: color",
  `data-binding__colorbreaks` = "target: guide-legend.breaks",
  `data-binding__colorlabels` = "target: guide-legend.labels",
  `data-binding__colortitle` = "target: guide-legend.title"
),

# animate the plot rotation
atags$other('animation', attribute = "rotation",
            from = "0 45 0", to = "0 405 0",
            dur = "10000", `repeat` = "indefinite")

)

server = function(input, output, session) {
  output$mydatascene <- renderADataScene({
    names(iris) <- tolower(names(iris))
    # Margin in (0,1) scale keeps polyhedra from sticking out of plot area
    positional_to <- c(0.01, 0.99)
    # convert to #RRGGBB color
    color_scale = setNames(rainbow(3, 0.75, 0.5, alpha = NULL),
                           unique(iris$species))

    iris %>%
       # scale positional data
       mutate_if(is.numeric, rescale, to = positional_to) %>%
       # scale size data to relative percentage, using cube root to correct
       # for radius->volume perception bias
       mutate(petal.width.size = rescale(petal.width^(1/3), to = c(0.5, 2)),
              species.color = color_scale[species]) ->
       iris_scaled

    # provide guide info
    make_guide <- function(var, aes, breaks = c(0.01, 0.5, 0.99)) {
      guide = list()
domain = range(iris[[var]])
guide[[paste0(aes, "breaks")]] <- breaks
guide[[paste0(aes, "labels")]] <- c(domain[1],
    round(mean(domain), 2),
    domain[2])
guide[[paste0(aes, "title")]] <- var
guide
)
Map(make_guide,
    var = c("sepal.length", "sepal.width", "petal.length"),
    aes = c("x", "y", "z")) %>%
    # repeat radius adjustment in the guide
c(list(make_guide("petal.width", "size", c(0.5, 1.25, 2)^(1/3)))) %>%
    Reduce(f = c) ->
guides
guides$shapetitle = "species"
guides$colortitle = "species"
guides$colorbreaks = color_scale
guides$colorlabels = names(color_scale)

    # convert data frame to list and combine with guides list
aDataScene(c(iris_scaled, guides))
})
)
Index

*Topic datasets
  aframetags, 4

aDataScene, 2
aDataScene-shiny, 3
aDataSceneOutput (aDataScene-shiny), 3
aframeAssets (aframetags), 4
aframeBox (aframetags), 4
aframeEntity (aframetags), 4
aframeMixin (aframetags), 4
aframePrimitive (aframetags), 4
aframeScene (aframetags), 4
aframeSphere (aframetags), 4
aframetags, 4
atags (aframetags), 4

includeScript, 3

renderADataScene, 2
renderADataScene (aDataScene-shiny), 3

shinyaframe, 6
shinyaframe-package (shinyaframe), 6

tag, 3, 5