Package 'rglwidget'

December 28, 2015

Type Package

Title 'rgl' in 'htmlwidgets' Framework

Version 0.1.1434

Author Duncan Murdoch

Maintainer Duncan Murdoch <murdoch.duncan@gmail.com>

Description Provides an 'htmlwidgets' framework for the 'rgl' package.

License GPL-2

LazyData TRUE

Imports htmlwidgets, htmltools, knitr, jsonlite, shiny, magrittr, rgl (>= 0.95.1367)

Repository CRAN

Repository/R-Forge/Project rgl

Repository/R-Forge/Revision 1434

Repository/R-Forge/DateTimeStamp 2015-12-10 12:12:35

Date/Publication 2015-12-28 22:20:12

NeedsCompilation no

R topics documented:

ageControl *Set attributes of vertices based on their age.*

Description

This is a function to produce actions in response to a playwidget or Shiny input control. The mental model is that each of the vertices of some object has a certain birth time; a control sets the current time, so that vertices have ages depending on the control setting. Attributes of those vertices can then be changed.

Usage

```
ageControl(births, ages, objids, value = 0,
           colors = NULL, alpha = NULL, radii = NULL, vertices = NULL,
           normals = NULL, origins = NULL, texcoords = NULL,
           x = NULL, y = NULL, z = NULL,
           red = NULL, green = NULL, blue = NULL)
```

Arguments

births	Numeric birth times of vertices.
ages	Chosen ages at which the following attributes will apply.
objids	Object ids to which the changes apply.
value	Initial value; typically overridden by input.
colors, alpha, radii, vertices, normals, origins, texcoords	Attributes of the vertices that can be changed. There should be one entry or row for each entry in ages.
x, y, z, red, green, blue	These one-dimensional components of vertices and colors are provided for convenience.

Details

All attributes must have the same number of entries (rows for the matrices) as the ages vector. The births vector must have the same number of entries as the number of vertices in the object.

Not all objects contain all attributes; if one is chosen that is not a property of the corresponding object, a Javascript alert() will be generated. (This restriction may be removed in the future by attempting to add the attribute when it makes sense.)

If a births entry is NA, no change will be made to that vertex.

Value

A list of class "rglControl" of cleaned up parameter values, to be used in an rgl widget.

Author(s)

Duncan Murdoch

Examples

```
saveopts <- options(rgl.useNULL = TRUE)

theta <- seq(0, 4*pi, len=100)
xyz <- cbind(sin(theta), cos(theta), sin(theta/2))
lineid <- rgl::plot3d(xyz, type="l", alpha = 1:0, lwd = 5, col = "blue")["data"]

rglwidget(width=500, height=300) %>%
playwidget(ageControl(births = theta,
                      ages = c(-4*pi, -4*pi, 1-4*pi, 0, 0, 1),
                      objids = lineid,
                      alpha = c(0, 1, 0, 0, 1, 0)),
           start = 0, stop = 4*pi,
           step = 0.1, rate = 4)

options(saveopts)
```

---

clipplaneControl *Sets attributes of a clipping plane.*

Description

This is a function to produce actions in a web display. A `playwidget` or Shiny input control (e.g. a `sliderInput` control) sets a value which controls attributes of one or more clipping planes.

Usage

```
clipplaneControl(a = NULL, b = NULL, c = NULL, d = NULL, plane = 1, clipplaneids, ...)
```

Arguments

`a, b, c, d`	Parameter values for the clipping planes.
`plane`	Which plane in the clipplane object?
`clipplaneids`	The id of the clipplane object.
`...`	Other parameters passed to `propertyControl`.

Value

A list of class `"rglControl"` of cleaned up parameter values, to be used in an rgl widget.

Author(s)

Duncan Murdoch

Examples

```
saveopts <- options(rgl.useNULL = TRUE)
xyz <- matrix(rnorm(300), ncol = 3)
id <- rgl::plot3d(xyz, type="s", col = "blue", zlim = c(-3,3))["clipplanes"]
dvals <- c(3, -3)
rglwidget(width=500, height=300) %>%
  playwidget(clipplaneControl(d = dvals, clipplaneids = id),
             start = 0, stop = 1, step = 0.01,
             rate = 0.5)
options(saveopts)
```

`elementId2Prefix` *Use widget with old-style controls.*

Description

The `rglwidget` control is designed to work in the **htmlwidgets** framework. Older **rgl** web pages that used `writeWebGL` or **knitr** used a different method of linking the controls to the scene. This is a partial bridge between the two systems.

Usage

```
elementId2Prefix(elementId, prefix = elementId)
```

Arguments

`elementId`	An element identifier from a `rglwidget` call.
`prefix`	The prefix to use in the old-style control.

Details

Because of the order of initialization, this isn't a perfect bridge. The old-style control will not set the scene to the initial value from the control, but subsequent changes to the control should be passed to the widget.

Value

This function generates Javascript code, so it should be used in an `results = "asis"` block in a **knitr** document.

Author(s)

Duncan Murdoch

Examples

```
## Not run:
rglwidget(elementId = "thewidget")
elementId2Prefix("thewidget", "theprefix")
subsetSlider(subsets = as.list(1:5),
             prefixes = "theprefix", subscenes = 1)

## End(Not run)
```

`import` *Imported from magrittr*

Description

This object is imported from **magrittr**. Follow the link to its documentation.

magrittr `%>%`

In **rglwidget**, pipes can be used to string together `rglwidget` calls and `playwidget` calls. See `ageControl` for an example.

`playwidget` *Add a widget to play animations.*

Description

This is a widget that can be put in a web page to allow animations with or without Shiny.

Usage

```
playwidget(sceneId, ...)

## Default S3 method:
playwidget(sceneId, controls,
            start = 0, stop = Inf, interval = 0.05, rate = 1,
            components = c("Reverse", "Play", "Slower", "Faster",
                           "Reset", "Slider", "Label"),
            loop = TRUE,
            step = 1, labels = NULL,
            precision = 3,
            elementId = NULL, respondTo = NULL,
            reinit = NULL,
            ...)

## S3 method for class 'rglWebGL'
playwidget(sceneId, controls, elementId = NULL, ...)
```

```
## S3 method for class 'rglPlayer'
playwidget(sceneId, controls, ...)

## S3 method for class 'shiny.tag.list'
playwidget(sceneId, controls, elementId = NULL, ...)
```

Arguments

sceneId	The HTML id of the rgl scene being controlled, or an object. See the Details below.
controls	A single "rglControl" object, e.g. propertyControl, or a list of several.
start, stop	The starting and stopping values of the animation. If labels is supplied stop will default to step through the labels.
interval	The requested interval (in seconds) between updates. Updates may occur at longer intervals.
rate	The number of units of "nominal" time per real world second.
components	Which components should be displayed? See Details below.
loop	When the player reaches the end of the interval, should it loop back to the beginning?
step	Step size in the slider.
labels	Optional labels to use, corresponding to slider steps. Set to NULL for autogenerated labels.
precision	If labels=NULL, the precision to use when displaying timer values.
elementId	The HTML id of the generated widget, containing buttons, slider, etc.
respondTo	The HTML ID of a Shiny input control (e.g. a sliderInput control) to respond to.
reinit	A vector of ids that will need re-initialization before being drawn again.
...	The default method passes additional arguments to htmlwidgets::createWidget.

Details

The components are buttons to control the animation, a slider for manual control, and a label to show the current value. They will be displayed in the order given in components. Not all need be included.

The buttons have the following behaviour:

Reverse Reverse the direction.

Play Play the animation.

Slower Decrease the playing speed.

Faster Increase the playing speed.

Reset Stop the animation and reset to the start value.

If respondTo is used, no components are shown, as it is assumed Shiny (or whatever control is being referenced) will provide the UI components.

The sceneId component can be another playwidget or a rglwidget result. This allows you to use a **magrittr**-style "pipe" command to join an rglwidget with one or more playwidgets. If a playwidget comes first, sceneId should be set to NA. If the rglwidget does not come first, previous values should be piped into its controllers argument. Other HTML code (including other widgets) can be used in the chain if wrapped in htmltools::tagList.

Value

A widget suitable for use in an **Rmarkdown**-generated web page, or elsewhere.

Appearance

The appearance of the controls is set by the stylesheet in system.file("htmlwidgets/lib/rglClass/rgl.css").

The overall widget is of class rglPlayer, with id set according to elementId.

The buttons are of HTML class rgl-button, the slider is of class rgl-slider, and the label is of class rgl-label. Each element has an id prefixed by the widget id, e.g. elementId-button-Reverse, elementId-slider, etc. (where elementId should be replaced by the actual id).

The reinit parameter handles the case where an object needs re-initialization after each change. For example, plane objects may need this if their intersection with the bounding box changes shape. Note that re-initialization is generally incompatible with the vertexControl as it modifies values which are set during initialization.

Author(s)

Duncan Murdoch

See Also

subsetControl, propertyControl, ageControl and vertexControl are possible controls to use.

Examples

```
saveopts <- options(rgl.useNULL = TRUE)

objid <- rgl::plot3d(1:10, 1:10, rnorm(10), col=c("red", "red"), type = "s")["data"]

control <- ageControl(value=0,
             births=1:10,
             ages = c(-5,0,5),
             colors = c("green", "yellow", "red"),
             objids = objid)

# This example uses explicit names

rglwidget(elementId = "theplot", controllers = "theplayer",
          height = 300, width = 300)
playwidget("theplot", control, start = -5, stop = 5,
           rate = 3, elementId = "theplayer",
```

```
            components = c("Play", "Slider"))

# This example uses pipes, and can skip the names

rglwidget(height = 300, width = 300) %>%
playwidget(control, start = -5, stop = 5,
            rate = 3, components = c("Play", "Slider"))


options(saveopts)
```

`rglwidget` *An htmlwidget to hold an rgl scene.*

Description

The **htmlwidgets** package provides a framework for embedding graphical displays in HTML documents of various types. This function provides the necessities to embed an **rgl** scene in one.

Usage

```
rglwidget(x = scene3d(), width = NULL, height = NULL,
          controllers = NULL, snapshot = FALSE,
          elementId = NULL, reuse = !interactive(), ...)
```

Arguments

`x`	An **rgl** scene produced by the `scene3d` function.
`width, height`	The width and height of the display in pixels.
`controllers`	Names of `playwidget` objects associated with this scene, or objects (typically piped in). See Details below.
`snapshot`	Control of snapshot of scene. See `writeWebGL` for details.
`elementId`	The id to use on the HTML `div` component that will hold the scene.
`reuse`	A logical variable indicating whether rgl objects from earlier scenes should be referenced. See the Details below.
`...`	Additional arguments to pass to `htmlwidgets::createWidget`.

Details

This produces a WebGL version of an **rgl** scene using the **htmlwidgets** framework. This allows display of the scene in an **rmarkdown** document or in a **shiny** app.

In a **shiny** app, there will often be one or more `playwidget` objects in the app, taking input from the user. In order to be sure that the initial value of the user control is reflected in the scene, you should list all players in the `controllers` argument. See the sample application in `system.file("shinyDemo", package = "rglwidget")` for an example.

In RMarkdown or in standalone code, you can use a **magrittr**-style "pipe" command to join an `rglwidget` with a `playwidget`. If the `playwidget` comes first, it should be piped into the `controllers` argument. If the `rglwidget` comes first, it can be piped into the first argument of `playwidget`.

If the `reuse` argument is `FALSE` (the default in interactive use), earlier information will be cleared before drawing the new scene. If `TRUE`, earlier data will be re-used in the current scene, so it may be smaller and faster to load. In both cases information from the current scene (added to earlier information if `reuse=TRUE`) will be saved for possible use in a future scene. If `reuse=NA`, the saved information will neither be used nor updated.

If `elementId` is `NULL` and we are not in a Shiny app, `elementId` is set to a random value to facilitate re-use of information.

Value

An object of class `"htmlwidget"` (or `"shiny.tag.list"` if pipes are used) that will intelligently print itself into HTML in a variety of contexts including the R console, within R Markdown documents, and within Shiny output bindings.

It should also display in the RStudio viewer.

If `reuse = TRUE`, a record will be kept of objects in the scene and they need not be included in the HTML generated for later scenes. This is normally useful only in **knitr** documents which can have many **rgl** scenes; if the widget is displayed in RStudio, only one scene will be shown.

Appearance

The appearance of the display is set by the stylesheet in `system.file("htmlwidgets/lib/rglClass/rgl.css")`.

The widget is of class `rglWebGL`, with id set according to `elementId`. (As of this writing, no special settings are given for class `rglWebGL`, but you can add your own.)

Author(s)

Duncan Murdoch

See Also

`hook_webgl` for an earlier approach to this problem. `rglwidgetOutput` for Shiny details.

Examples

```
save <- getOption("rgl.useNULL")
options(rgl.useNULL=TRUE)
example("plot3d", "rgl")
rglwidget()
```

rglwidget-internal *rglwidget internals*

Description

These functions are planned to implement **rgl** functions, they are not meant to be called directly by the user.

Usage

```
.ageSetter(births, ages, colors = NULL, alpha = NULL, radii = NULL,
           vertices = NULL, normals = NULL, origins = NULL,
           texcoords = NULL, objids, prefixes = "", digits = 7,
           param = seq(floor(min(births)), ceiling(max(births))))

.par3dinterpSetter(fn, from, to, steps, subscene = NULL,
           omitConstant = TRUE, rename = character(), ...)

.propertySetter(values = NULL, entries, properties, objids,
                prefixes = "", param = seq_len(NROW(values)),
                interp = TRUE, digits = 7)

.vertexSetter(values = NULL, vertices = 1, attributes, objid,
              prefix = "", param = seq_len(NROW(values)),
              interp = TRUE, digits = 7)

.subsetSetter(subsets, subscenes = currentSubscene3d(), prefixes = "",
 fullset = Reduce(union, subsets),
 accumulate = FALSE)

.propertySlider(setter = .propertySetter,
                minS = NULL, maxS = NULL,
                step = 1, init = NULL, labels,
                id = basename(tempfile("input")), name = id,
                outputid = paste0(id, "text"), index = NULL, ...)

.setupKnitr()

.hook_rgl(before, options, envir)

.hook_webgl(before, options, envir)

.writeWebGL(dir = "webGL", filename = file.path(dir, "index.html"),
         template = system.file(file.path("WebGL", "template.html"), package = "rgl"),
           prefix = "", snapshot = TRUE, commonParts = TRUE,
           reuse = NULL, font = "Arial", width = NULL, height = NULL)
```

Arguments

`births, ages, colors, alpha, radii, vertices, normals, origins, texcoords, objids, prefixes, digits`
: See the corresponding **rgl** function (i.e. leave off the "." prefix) for a description of the arguments.

Author(s)

Duncan Murdoch

See Also

`ageSetter`, `par3dinterpSetter`, `propertySetter`, `vertexSetter`

---

sceneChange *Make large change to a scene from Shiny*

Description

These functions allow Shiny apps to make relatively large changes to a scene, adding and removing objects from it.

Usage

```
sceneChange(elementId, x = scene3d(),
            delete = NULL, add = NULL, replace = NULL,
            material = FALSE, rootSubscene = FALSE,
            delfromSubscenes = NULL, skipRedraw = FALSE)
registerSceneChange()
```

Arguments

`elementId`
: The id of the element holding the `rglClass` instance.

`x`
: The new scene to use as a source for objects to add.

`delete, add, replace`
: Object ids to modify in the scene. The `delete` and `replace` ids must be present in the old scene in the browser; the `add` and `replace` ids must be present in `x`.

`material`
: Logical to indicate whether default material should be updated.

`rootSubscene`
: Logical to indicate whether root subscene should be updated.

`delfromSubscenes`
: A vector of subscene ids that may have been changed by deletions. By default, all subscenes in `x` are used, but the objects may be included in subscenes in the browser that are different.

`skipRedraw`
: If `TRUE`, stop the scene from redrawing until `skipRedraw=FALSE` is sent. If `NA`, don't redraw this time, but don't change the state of the `skipRedraw` flag.

Details

registerSceneChange must be called in the UI component of a Shiny app to register the "sceneChange" custom message.

Value

registerSceneChange returns the HTML code to register the message.

sceneChange returns a list to be used as the "sceneChange" message to change the scene. Use shiny::session$sendCustomMessage to send it.

Author(s)

Duncan Murdoch

See Also

playwidget for a different approach to modifying scenes that can be much faster, but may be less flexible. The Shiny demo in this package makes use of all of these approaches.

Examples

```
## Not run:
shinyUI(fluidPage(
  registerSceneChange(),
  actionButton("thebutton", "Change")
))

shinyServer(function(input, output, session) {
  observeEvent(input$thebutton, {
    session$sendCustomMessage("sceneChange",
      sceneChange("thewidget", delete = deletes, add = adds))
  })
})

## End(Not run)
```

---

shiny *Functions for integration of* rglwidget *into Shiny app.*

Description

These functions allow an **rgl** scene to be embedded in a Shiny app.

Usage

```
rglwidgetOutput(outputId, width = "512px", height = "512px")
renderRglwidget(expr, env = parent.frame(), quoted = FALSE)

playwidgetOutput(outputId, width = "0px", height = "0px")
renderPlaywidget(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	The name for the control.
width, height	Width and height to display the control.
expr	An R expression returning a rglwidget (for renderRglwidget) or a playwidget (for renderPlaywidget) as output.
env	The environment in which to evaluate expr.
quoted	Is the expression already quoted?

Details

Use rglwidgetOutput or playwidgetOutput as an output object in a Shiny user interface section; use renderRglwidget or renderPlaywidget as the render function in the server section.

Value

Used internally by Shiny.

Author(s)

Duncan Murdoch

---

vertexControl *Set attributes of vertices.*

Description

This is a function to produce actions in a web display. A playwidget or Shiny input control (e.g. a sliderInput control) sets a value which controls attributes of a selection of vertices.

Usage

```
vertexControl(value = 0, values = NULL, vertices = 1, attributes,
              objid, param = seq_len(NROW(values)) - 1, interp = TRUE)
```

Arguments

value	The value to use for input (typically input$value in a Shiny app.) Not needed with playwidget.
values	A matrix of values, each row corresponding to an input value.
vertices	Which vertices are being controlled? Specify vertices as a number from 1 to the number of vertices in the objid.
attributes	A vector of attributes of a vertex, from c("x", "y", "z", "red", "green", "blue", "alpha", "nx", See Details.
objid	A single **rgl** object id.
param	Parameter values corresponding to each row of values.
interp	Whether to interpolate between rows of values.

Details

Like rgl::vertexSetter, this function modifies attributes of vertices in a single object. The attributes are properties of each vertex in a scene; not all are applicable to all objects. In order, they are: coordinates of the vertex "x", "y", "z", color of the vertex "red", "green", "blue", "alpha", normal at the vertex "nx", "ny", "nz", radius of a sphere at the vertex "radius", origin within a texture "ox", "oy" and perhaps "oz", texture coordinates "ts", "tt".

If only one attribute of one vertex is specified, values may be given as a vector and will be treated as a one-column matrix. Otherwise values must be given as a matrix with ncol(values) == max(length(vertices),length

The value argument is translated into a row (or two rows if interp = TRUE) of values by finding its location in param.

Value

A list of class "rglControl" of cleaned up parameter values, to be used in an rgl widget.

Author(s)

Duncan Murdoch

Examples

```
saveopts <- options(rgl.useNULL = TRUE)

theta <- seq(0, 6*pi, len=100)
xyz <- cbind(sin(theta), cos(theta), theta)
rgl::plot3d(xyz, type="l")
id <- rgl::spheres3d(xyz[1,,drop=FALSE], col="red")

rglwidget(width=500, height=300) %>%
playwidget(vertexControl(values=xyz,
                         attributes=c("x", "y", "z"),
                         objid = id, param=1:100),
           start = 1, stop = 100, rate=10)
options(saveopts)
```

---

webGLcontrols *Controls to use with playwidget().*

Description

These are setter functions to produce actions in a Shiny app, or in an animation.

Usage

```
subsetControl(value = 1, subsets, subscenes = NULL,
                         fullset = Reduce(union, subsets),
                         accumulate = FALSE)
propertyControl(value = 0, entries, properties, objids, values = NULL,
                param = seq_len(NROW(values)) - 1, interp = TRUE)
```

Arguments

value	The value to use for input (typically input$value in a Shiny app.)
subsets	A list of vectors of object identifiers; the value will choose among them.
fullset	Objects in the subscene which are not in fullset will not be touched.
subscenes	The subscenes to be controlled. If NULL, the root subscene.
accumulate	If TRUE, the subsets will accumulate (by union) as the value increases.
entries, properties, objids	Which properties to set.
values	Values to set.
param	Parameter values corresponding to the rows of value
interp	Whether to use linear interpolation between param values

Details

subsetControl produces data for playwidget to display subsets of the object in one or more subscenes. This code will not touch objects in the subscenes if they are not in fullset. fullset defaults to the union of all the object ids mentioned in subsets, so by default if an id is not mentioned in one of the subsets, it will not be controlled by the slider. If value is specified in R code, it will be a 1-based index into the subsets list; when specified internally in Javascript, 0-based indexing into the corresponding array will be used.

propertyControl sets individual properties. Here the row of values is determined by the position of value in param.

Value

These functions return controller data in a list of class "rglControl".

Author(s)

Duncan Murdoch

See Also

subsetSetter for a way to embed a pure Javascript control, and playwidget for a way to use these in animations (including Shiny).

Index