Package 'svgViewR'

January 20, 2018

Date 2018-01-19

Title 3D Animated Interactive Visualizations Using SVG

Description Creates 3D animated, interactive visualizations that can be viewed in a web browser.

Version 1.3

Depends R (>= 3.2.4)

Imports grid, rjson, Rook, methods

Author Aaron Olsen

Maintainer Aaron Olsen <aarolsen@gmail.com>

Repository CRAN

URL https://aaronolsen.github.io/tutorials/visualization3d.html

NeedsCompilation no

Date/Publication 2018-01-20 15:43:02 UTC

R topics documented:

svgViewR-package *3D Animated Interactive Visualizations using SVG*

Description

The svgViewR package allows users to create 3D animated, interactive visualizations that can be viewed in a web browser.

Details

Package:	svgViewR
Type:	Package
Version:	1.3
Date:	2018-01-19
License:	GPL (>= 2)

Author(s)

Aaron Olsen Maintainer: Aaron Olsen <aarolsen@gmail.com>

objToJSON *Converts OBJ to JSON*

Description

Converts a mesh file in the OBJ format into the JSON format

Usage

```
objToJSON(obj, file = NULL)
```

Arguments

obj	A filepath to a file of type '.obj' or an 'obj' object, the value returned by the function readOBJ.
file	A filepath to a file of type '.json' where the converted obj object will be saved. If is NULL (default) then no file is written and the JSON object is returned.

Details

This function convert a Wavefront .obj file or OBJ object into a JSON (JavaScript Object Notation) object and optionally saves this JSON object as a .json file. This function was written in order directly convert CT scan meshes exported from Horos into the JSON format for web visualization, eliminating intermediate conversions in programs such as meshlab and Blender. See readOBJ for more details and limitations in .obj file reading.

Value

A JSON object, if file is NULL

Author(s)

Aaron Olsen

See Also

readOBJ

Examples

```
## Not run:
# Convert OBJ to JSON file
objToJSON(obj='obj_file.obj', file='json_file.json')

# Convert OBJ to JSON list object
json_list <- objToJSON(obj='obj_file.obj')

## End(Not run)
```

---

readOBJ *Reads an OBJ file*

Description

Reads a mesh file in the OBJ format and returns an object of class 'obj'

Usage

```
readOBJ(file, scaling = 1)

## S3 method for class 'obj'
print(x, ...)
```

Arguments

file	A filepath to a file of type '.obj'.
scaling	A scaling factor to be applied to the vertices.
x	an object used to select a method.
...	further arguments passed to or from other methods.

Details

This function convert a Wavefront .obj file or OBJ object into a JSON (JavaScript Object Notation) object and optionally saves this JSON object as a .json file. This function only provides basic functionality, reading only vertices, normals, and faces. It has only been tested on the .obj files returned from the DICOM Medical Image Viewer Horos (formerly Osirix). This function was written in order directly convert CT scan meshes exported from Horos into the JSON format for web visualization, eliminating intermediate conversions in programs such as meshlab and Blender.

Value

a list of class "obj" with the following elements:

vertices	Mesh vertices.
normals	Normal vectors, at each vertex.
faces	Mesh faces (indices to vertices).
metadata	Mesh metadata.

Author(s)

Aaron Olsen

See Also

objToJSON

Examples

```
## Not run:
# Read an .obj file
obj <- readOBJ('obj_file.obj')

# Print obj object
print(obj)

## End(Not run)
```

svg.arrows *Write arrows to Viewer*

Description

Draws static and animated arrow(s) in Viewer.

Usage

```
svg.arrows(x, y=NULL, name="arrow", col="black", z.index=0, layer="",
           label="", lwd=1, len='auto', angle=0.4, opacity=1, file=NULL)
```

Arguments

x	For a single static arrow, a 2x3 matrix in which the rows are the start and end point in 3D. For a single animated arrow, a 2x3xi array in which the first two dimensions specify the start and end point of the arrow and i is the number of animation iterations. For multiple static arrows, a list of length n, each element being a 2x3 matrix, in which n is the number of arrows. For multiple animated arrows, a list of length n, each element being a 2x3xi array indicating the start and end points of each arrow over i iterations.
y	If x is a vector, the coordinates on the y-axis.
name	The name of the drawn object.
col	The stroke color of the arrow.
z.index	A number indicating the relative order in which the object will be drawn in the viewer. Higher numbers correspond to closer to the front or top.
layer	A text string indicating the layer in which the object belongs (not yet fully implemented).
label	A label to be added to the object in SVG tag.
lwd	The thickness of the arrow.
len	The length of the arrowhead.
angle	The angle (in radians) between the arrowhead lines and the main shaft of the arrow.
opacity	The opacity of the arrow.
file	File path (having the extenstion ".html") to add lines to a current Viewer file. By default (i.e. NULL) lines are added to the Viewer opened by svg.new.

Details

This function accepts many different input types for x to plot four different arrow types: a single static arrow, multiple static arrows, a single animated arrow, and multiple animated arrows. For worked examples, please see Plotting arrows with svgViewR. The first of these worked examples is included in the examples below.

Value

NULL

Author(s)

Aaron Olsen

See Also

svg.new, svg.lines

Examples

```
## Not run:
## Create arrow
arrow <- rbind(rep(0,3), rep(1,3))

# Open a connection to .html file
svg.new(file='plot_static_arrow.html')

# Add arrow
svg.arrows(arrow)

# Add a coordinate axis planes around the arrow
svg_frame <- svg.frame(arrow)

# Close the file connection
svg.close()

## End(Not run)
```

---

svg.bboxLight *Adds light(s) to Viewer*

Description

Adds point light(s) in Viewer at corners of bounding box.

Usage

```
svg.bboxLight(x=c(1,1,1), col='#FFFFDD', intensity=1, distance=3, hidden = TRUE)
```

Arguments

x	The corner(s) where a light source (vector) or multiple light sources (matrix) should be added.
col	The color of the light.
intensity	The intensity of the light.

distance	The distance from the light where the intensity is 0.
hidden	A logical indicating whether the light source location should be indicated by a yellow sphere. By default the sphere is hidden. Setting this to FALSE may be useful for troubleshooting light positions (so that they can be viewed directly in the scene).

Details

This function only works in the webgl plotting mode (i.e. when the mode parameter of svg.new is 'webgl'). This function creates a point light (if x is a vector) or lights (if x is a matrix) in the Viewer scene at the specified coordinates (x). The point light is the PointLight object in the three.js library.

Value

NULL

Author(s)

Aaron Olsen

See Also

svg.new

Examples

```
## Not run:
# Create new Viewer
svg.new(mode='webgl')

# Add object
svg.cylinder(ends=rbind(c(0,0,0), c(0,10,0)), radius=1, col='green')

# Add lights in four corners of the bounding box
svg.bboxLight(x=rbind(c(1,1,1), c(-1,1,1), c(-1,-1,-1), c(1,-1,-1)))

# Close connection
svg.close()

## End(Not run)
```

---

svg.close *Closes Viewer connection*

Description

Closes the connection to the Viewer file.

Usage

```
svg.close()
```

Details

This function closes the connection to the Viewer file opened by svg.new. If no file is input to svg.new and the mode is 'webgl' then calling svg.close will signal that all objects have been plotted and the Viewer will open in the default web browser.

Value

NULL

Author(s)

Aaron Olsen

See Also

svg.new

Examples

```
## Not run:
# Create new Viewer
svg.new(mode='webgl')

# Add object
svg.cylinder(ends=rbind(c(0,0,0), c(0,10,0)), radius=1, col='green')

# Close connection
svg.close()

## End(Not run)
```

---

svg.lines *Add Connected Line Segments to SVG Viewer*

Description

A function taking coordinates given in various ways and joining the corresponding points with line segments in an SVG Viewer.

Usage

```
svg.lines(x, y=NULL, col="black", z.index=0, layer="", name="line", label="",
          lwd=1, opacity=1, file=NULL)
```

Arguments

x	A vector, matrix or array of 2D or 3D coordinates to be joined by a line or lines. Coordinates input as an array will be animated.
y	If x is a vector, the coordinates on the y-axis to be joined by a line or lines.
col	The color of the line(s).
z.index	A number indicating the relative order in which the SVG object will be drawn in the viewer. Higher numbers correspond to closer to the front or top.
layer	A text string indicating the layer in which the SVG object belongs (not yet fully implemented).
name	The name of the drawn object.
label	A label to be added to the SVG object in SVG tag.
lwd	The thickness of the line(s).
opacity	A number between 0 and 1 indicating the opacity of the line(s).
file	File path (having the extenstion ".html") to add lines to a current SVG Viewer file. By default (i.e. NULL) lines are added to the SVG Viewer opened by svg.new.

Details

This function accepts input similar to the native plot function lines(). If x and y are vectors, they are combined into a matrix using cbind(). If x is a matrix, this matrix is used directly. Lines are then drawn between points indicated by consecutive rows. So for a 2-row matrix one line would be drawn, for a 3-row matrix two lines would be drawn, etc.

If x is an array, the array is interpreted as a series of matrices, each representing a state of the line or line(s) in an animation of length dim(x)[3]. Each of the dim(x)[3] matrices is used to draw each state in a manner identical to when x is a matrix.

The graphical parameters col, z.index, layer, label, lwd, and opacity can all be vectors of length one or of the same length as the number of lines to be drawn (see Examples). This allows different parameters to be specified for each line or for different animation states, depending on the number of graphical parameters specified.

Value

NULL

Author(s)

Aaron Olsen

See Also

svg.new, svg.pathsC, svg.points

Examples

```
## Not run:
## Create static and animated lines
# Create new viewer
svg.new(file='svgviewr.html', animate.duration=1)

# Plot 3 connected lines with 3 different colors
svg.lines(x=rbind(c(30,-20,0), c(30,-30,0), c(40,-30,0), c(40,-35,0)),
col=c("red", "green", "blue"), lwd=5, opacity=0.7)

# Plot single line that switches among 3 colors
svg.lines(x=rbind(c(15,0,0), c(15,-20,0)), col=c("red", "green", "blue"), lwd=3, opacity=0.7)

# Create a line in two animation states
arr <- array(c(rbind(c(15,-30,0), c(15,-50,0)), rbind(c(10,-30,0), c(10,-50,0))), dim=c(2,3,2))

# Plot
svg.lines(x=arr, col=c("red", "green"), lwd=3, opacity=0.7)

# Create two connected lines in 3 animation states
arr <- array(c(30,30,40, -40,-50,-50, 0,0,0, 40,40,50, -40,-50,-50,
0,0,0, 50,50,60, -40,-50,-50, 0,0,0), dim=c(3,3,3))

# Plot
svg.lines(x=arr, col=c("red", "green"), lwd=5, opacity=0.7)

# Close connection
svg.close()

# Open svgviewr.html to visualize

## End(Not run)
```

`svg.mesh` *Write mesh to Viewer*

Description

Draws mesh in Viewer.

Usage

```
svg.mesh(file=NULL,
         name=gsub('[.][A-Za-z]+$', '', tail(strsplit(file, '/')[[1]], 1)),
         col='#F5F5F5', emissive='black', opacity=1, get.lim=TRUE)
```

Arguments

file	Preferably a mesh file in the .json format (see objToJSON). An .obj file can be input but this will take longer to run since the function will first convert the OBJ to JSON. If svg.new is called without a filename (loading the Viewer as a local server), you will be prompted to create a .json file since the server must read a .json file.
name	The name of the mesh. By default, the filename is used (without the file extension). This is used when applying transformations to drawn objects.
col	The mesh color.
emissive	The mesh emissive color.
opacity	The mesh opacity. A value of 1 (default) is fully opaque.
get.lim	A logical indicating whether the limits of the mesh should be found and returned. This can be used to position other objects relative to the mesh bounds. This will be done at a later point (if not done when svg.mesh is called) so it does not save processing time to set this to FALSE.

Details

This function only works in the webgl plotting mode (i.e. when the mode parameter of svg.new is 'webgl'). This function adds a mesh to the Viewer. See readOBJ for more details and limitations on what meshes can be plotted.

Value

If get.lim is TRUE, a list with the following elements:

lim	The minimum and maximum values of each dimension of the mesh vertices.
corners	The eight corners of the bounding box surrounding the mesh.

Author(s)

Aaron Olsen

See Also

readOBJ, objToJSON

Examples

```
## Not run:
# Create new viewer
svg.new(mode='webgl')

# Add mesh
svg.mesh(file='mesh.json')

# Close connection
svg.close()
```

```
## End(Not run)
```

---

svg.new *Create new Viewer file*

Description

Creates a new Viewer file as an HTML document to which objects can be added, with optional specification of various animation parameters.

Usage

```
svg.new(file = NULL, window.title="svgViewR", animate.duration = 1,
        animate.speed = 1, animate.reverse = FALSE,
        animate.repeat = -1, margin = 20, col = "white",
        time.units = 'sec', clock = FALSE, stats = FALSE,
        show.control = TRUE, start.rotate = TRUE,
        rotate.speed = 1.2, zoom.speed = 1, pan.speed = 0.2,
        layers = NULL, connection = TRUE, mode = c('svg', 'webgl'),
        debug = FALSE)
```

Arguments

file	File path and name (having the extenstion ".html") where Viewer will be created. If file is NULL then the viewer will open over a local server in the default web browser and mode 'webgl' will be automatically selected.
window.title	The Viewer title, visible at the top of the web browser window.
animate.duration	Only used in 'svg' mode. The approximate duration in seconds of the animation. When the number of objects to be displayed is large, the actual duration might exceed this number.
animate.speed	Only used in 'webgl' mode. The relative speed at which the animation will playback. For example, to play the animation at half the real speed, animate.speed would be 0.5.
animate.reverse	A logical indicating whether the animation is to be played in reverse after each iteration. Only used in 'svg' mode.
animate.repeat	An integer specifying the number of times the animation will repeat. A value of -1 will cause the animation to repeat indefinitely. Only used in 'svg' mode.
margin	Margin when resizing visualization to fit the browser window.
col	Viewer background color.
time.units	The units for the times specified in any transformation.
clock	Whether a clock should be visible or not.

stats	Whether processing stats should be visible or not.
show.control	Whether control panel should be visible or not. Only used in 'svg' mode.
start.rotate	Whether visualization should start with 'rotate' enabled or 'translate' enabled. Only used in 'svg' mode.
rotate.speed	How much the camera rotates in response to mouse click and drag. Only used in 'webgl' mode.
zoom.speed	How much the camera zooms in response to mouse click and drag. Only used in 'webgl' mode.
pan.speed	How much the camera pans in response to mouse click and drag. Only used in 'webgl' mode.
layers	Not yet fully enabled.
connection	Whether to open a file connection or create a closed file.
mode	Whether to draw using the old mode ('svg') or the new mode ('webgl'). See details.
debug	Used for debugging.

Details

This function is used to initialize a new Viewer. Before adding shapes to a Viewer, this function is called to create the HTML file to which the objects can be added. svgViewR is currently undergoing a significant overhaul. The previous plotting using SVG (scalable vector graphics) is being replaced with visualizations created using WebGL (the Web Graphics Library) and the javascript library three.js. All backward compatability with the svg format should be maintained. To use the new plotting mode, set the mode parameter to 'webgl'. Plotting in the 'webgl' mode is limited as I have only begun implementing it.

The 'svg' (old) mode has a single visualization type: an .html file. The 'webgl' mode (new) has two visualization types: a local server based visualization (using the R package Rook) and an .html file. The .html file output is ideal if you want to create portable files that can be easily visualized anytime and shared without the need for hosting a server. The server output is ideal if you want to visualize many different tranformations of large mesh files; creating files for each visualization would take up a large amount of space because all of the mesh specifications (vertices, normals, faces) would have to be contained within each html file.

For worked examples, please see 3D visualization in R. Here are common interactive commands between the two modes:

- **spacebar** : Pauses and plays the animation
- **browser refresh** : Returns shapes to state when browser was originally opened
- **scroll up/down** : Zoom in/out by moving the shapes into and out of the screen

To rotate the camera in 'webgl' mode, left-click and drag the mouse. To pan the camera in 'webgl' mode, right-click and drag the mouse. For a key to the interactive commands in the 'svg' (old) mode, see svgViewR Interactive Commands.

Value

NULL

Author(s)

Aaron Olsen

See Also

svg.close

Examples

```
## Not run:
# Set number of points to draw
n <- 300

# Create a cloud of normally distributed 3D points
points3d <- cbind(rnorm(n, sd=3), rnorm(n, sd=2), rnorm(n, sd=1))

# Open a connection to .html file
svg.new(file='plot_static_points.html')

# Get distance of points from the center of point cloud
pdist <- sqrt(rowSums((points3d - matrix(colMeans(points3d), n, 3, byrow=TRUE))^2))

# Set color gradient between red and blue
colfunc <- colorRampPalette(c('red', 'blue'))

# Set desired number of colors along gradient
col_grad <- colfunc(50)

# Scale distances to indices and find corresponding color along gradient
col <- col_grad[(length(col_grad)-1)*(pdist - min(pdist)) / diff(range(pdist))+1]

# Add points to file
svg.points(points3d, col=col)

# Add a coordinate axis planes around the points
svg_frame <- svg.frame(points3d)

# Close the file connection
svg.close()

## End(Not run)
```

---

svg.pathsC *Connect points with path lines in SVG Viewer*

Description

Creates paths by drawing lines between points in the SVG Viewer.

Usage

```
svg.pathsC(path, col = NULL, col.fill = "none", col.stroke = "black",
               z.index = 0, layer = "", label = "", lwd = 1,
               opacity.stroke = 1, opacity.fill = 1, index.add = 0,
               file=NULL)
```

Arguments

path	A vector, matrix or list of integers specifying the points to be connected by lines. See Details.
col	The fill and stroke color of the points(s). If non-NULL, col overrides col.fill and col.stroke.
col.fill	The fill color of the path(s).
col.stroke	The stroke (border) color of the path line(s).
z.index	A number indicating the relative order in which the SVG object will be drawn in the viewer. Higher numbers correspond to closer to the front or top.
layer	A text string indicating the layer in which the SVG object belongs (not yet fully implemented).
label	A label to be added to the SVG object in SVG tag.
lwd	The thickness of the path line(s).
opacity.stroke	A number between 0 and 1 indicating the opacity of the border of the path line(s).
opacity.fill	A number between 0 and 1 indicating the opacity of the fill of the path line(s).
index.add	An integer to add to all indices in path.
file	File path (having the extenstion ".html") to add lines to a current SVG Viewer file. By default (i.e. NULL) lines are added to the SVG Viewer opened by svg.new.

Details

This function creates SVG paths by drawing lines between points that have been written to a SVG Viewer through a separate function call. This is particularly useful when animated points have been written to the SVG Viewer and the user simply wants to create paths defined by the animated points. Since paths are drawn, not simply lines, the fill color (col.fill) and fill opacity (opacity.fill) of the path can also be specified. Whether the paths written by this function are animated depends on the points that make up the path. If the points making up the path are animated, the path will follow the motion of its constitutive points.

The input path can be a vector, matrix or list of integers. The integers indicate which points should be joined by lines and in what order; these integers correspond to the points in the same order in which they were written to the SVG Viewer, starting with 1. Thus, 1 corresponds to the first point written to the SVG Viewer, 2 corresponds to the second point written to the SVG Viewer, etc. If path is a vector, a single path is drawn connecting the points corresponding to the indices in path. If path is a matrix, a separate path is drawn for each matrix row, connecting the points corresponding to the indices in each row. Similarly, if path is a list, a separate path is drawn for each list element, connecting the points corresponding to the indices in each list element.

The graphical parameters col, z.index, layer, label, lwd, and opacity can all be vectors of length one or of the same length as the number of paths to be drawn. This allows different parameters to be specified for each path.

Value

NULL

Author(s)

Aaron Olsen

See Also

svg.new, svg.lines, svg.points

Examples

```
## Transform a circle into an ellipse
# Create new viewer
svg.new(file='svgviewr.html', animate.reverse=TRUE, animate.duration=1)

# Create points
n <- 100
x <- array(NA, dim=c(100, 2, n))
x_seq <- seq(-1, 1, length=dim(x)[1]/2)
x_seq <- sin(seq(-pi/2, pi/2, length=dim(x)[1]/2))
n_seq <- seq(1, 3, length=n)
for(i in 1:dim(x)[3])
x[, , i] <- rbind(cbind(x_seq, n_seq[i]*sqrt(1 - x_seq^2)),
cbind(x_seq[(dim(x)[1]/2):1], -n_seq[i]*sqrt(1 - x_seq[(dim(x)[1]/2):1]^2)))

# Draw points
svg.points(x, cex=1, lwd=1, col="blue")

# Draw paths among points
svg.pathsC(1:dim(x)[1], col.fill="blue", opacity.fill=0.1,
col.stroke="green", lwd=2, z.index=-1)

# Close viewer connection
svg.close()

# Open svgviewr.html to visualize
```

---

svg.points *Write points to SVG Viewer*

Description

Draws a sequence of points at specified coordinates in an SVG Viewer.

Usage

```
svg.points(x, y=NULL, type="p", col=NULL, col.fill="black",
               col.stroke="black", z.index=0, layer="", label="",
               cex=2, lwd=2, opacity.stroke=1, opacity.fill=1,
               file=NULL)
```

Arguments

x	A vector, matrix or array of 2D or 3D coordinates. Coordinates input as an array will be animated.
y	If x is a vector, the coordinates on the y-axis.
type	character indicating the type of plotting. Currently, only "p" is supported.
col	The fill and stroke color of the points(s). If non-NULL, col overrides col.fill and col.stroke.
col.fill	The fill color of the points(s).
col.stroke	The stroke (border) color of the points(s).
z.index	A number indicating the relative order in which the SVG object will be drawn in the viewer. Higher numbers correspond to closer to the front or top.
layer	A text string indicating the layer in which the SVG object belongs (not yet fully implemented).
label	A label to be added to the SVG object in SVG tag.
cex	The size (radius) of the point(s).
lwd	The thickness of the border of the point(s).
opacity.stroke	A number between 0 and 1 indicating the opacity of the border of the point(s).
opacity.fill	A number between 0 and 1 indicating the opacity of the fill of the point(s).
file	File path (having the extenstion ".html") to add lines to a current SVG Viewer file. By default (i.e. NULL) lines are added to the SVG Viewer opened by svg.new.

Details

This function accepts input similar to the native plot function points(). If x and y are vectors, they are combined into a matrix using cbind(). If x is a matrix, this matrix is used directly. Each row of the matrix is drawn as a point. If x is an array, the array is interpreted as a series of matrices, each representing a state of the point or point(s) in an animation of length dim(x)[3]. Each of the dim(x)[3] matrices is used to draw each state in a manner identical to when x is a matrix. If x is an array, each state of points will be drawn as an animation.

The graphical parameters col, col.fill, col.stroke, z.index, layer, label, lwd, opacity.stroke and opacity.fill can all be vectors of length one or of the same length as the number of points to be drawn. This allows different parameters to be specified for each point.

Value

NULL

Author(s)

Aaron Olsen

See Also

svg.new, svg.lines, svg.pathsC

Examples

```
## Not run:
## Create animated sinusoid
# Create new viewer
svg.new(file='svgviewr.html', animate.duration=1)

# Create points with varying sin phase
n <- 100
x <- array(NA, dim=c(40, 2, n))
x_seq <- seq(-pi, pi, length=dim(x)[1])
n_seq <- seq(0, 2*pi, length=n)
for(i in 1:dim(x)[3]) x[, , i] <- cbind(x_seq, sin(x_seq + n_seq[i]))

# Draw points
svg.points(x, cex=2, lwd=1, col="blue")

# Close viewer connection
svg.close()

# Open svgviewr.html to visualize

## End(Not run)
```

Index