Package 'geomeTriD'

April 24, 2025

Type Package

Title A R/Bioconductor package for interactive 3D plot of epigenetic data or single cell data

Version 1.3.0

Description geomeTriD (Three Dimensional Geometry Package)
create interactive 3D plots using the GL library with the 'three.js' visualization library (https://threejs.org) or the rgl library.
In addition to creating interactive 3D plots, the application also generates simplified models in 2D.
These 2D models provide a more straightforward visual representation, making it easier to analyze and interpret the data quickly.
This functionality ensures that users have access to both detailed three-dimensional visualizations and more accessible two-dimensional views, catering to various analytical needs.

License MIT + file LICENSE

Depends R (>= 4.4.0)

Imports BiocGenerics, dbscan, GenomeInfoDb, GenomicRanges, graphics, grDevices, grid, htmlwidgets, igraph, InteractionSet, IRanges, MASS, Matrix, methods, plotrix, RANN, rgl, rjson, S4Vectors, scales, stats, trackViewer

Suggests RUnit, org.Hs.eg.db, TxDb.Hsapiens.UCSC.hg19.knownGene, manipulateWidget, shiny, BiocStyle, knitr, rmarkdown, testthat

biocViews Visualization

VignetteBuilder knitr

RoxygenNote 7.3.2

Encoding UTF-8

URL https://github.com/jianhong/geomeTriD

BugReports https://github.com/jianhong/geomeTriD/issues

git_url https://git.bioconductor.org/packages/geomeTriD

git_branch devel

git_last_commit 01ff21f

git_last_commit_date 2025-04-15

Repository Bioconductor 3.22

Date/Publication 2025-04-24

Author Jianhong Ou [aut, cre] (ORCID: <https://orcid.org/0000-0002-8652-2488>)

Maintainer Jianhong Ou <jou@morgridge.org>

Contents

`geomeTriD-package` *Interactive 3D plot of epigenetic data or single cell data*

Description

geomeTriD (Three Dimensional Geometry Package) create interactive 3D plots using the GL library with the 'three.js' visualization library (https://threejs.org) or the rgl library. In addition to creating interactive 3D plots, the application also generates simplified models in 2D. These 2D models provide a more straightforward visual representation, making it easier to analyze and interpret the data quickly. This functionality ensures that users have access to both detailed three-dimensional visualizations and more accessible two-dimensional views, catering to various analytical needs.

Author(s)

Maintainer: Jianhong Ou <jou@morgridge.org> (ORCID)

See Also

Useful links:

- https://github.com/jianhong/geomeTriD
- Report bugs at https://github.com/jianhong/geomeTriD/issues

Examples

```
if(interactive()){
  ## quick start from a simple data
  library(geomeTriD)
  set.seed(123)
  obj <- GRanges("1", IRanges(seq.int(10), width = 1),
                 x = sample.int(10, 10),
                 y = sample.int(10, 10),
                 z = sample.int(10, 10)
               )
  feature.gr <- GRanges("1", IRanges(c(3, 7), width = 3),
                        label = c("gene1", "gene2"),
                        col = c("red", "blue"),
                        type = "gene"
  )
  view3dStructure(obj, feature.gr,
                  renderer = "threejs",
                  coor_mark_interval = 5, coor_tick_unit = 2
  )
}
```

---

alignCoor *Aligns two sets of genomic with x,y,z*

Description

Aligns two sets of points via rotations and translations by Kabsch Algorithm.

Usage

```
alignCoor(query, subject)
```

Arguments

query, subject	GRanges objects to alignment.

Value

A GRanges object of query aligned to subject.

Examples

```
x <- readRDS(system.file("extdata", "4DNFI1UEG1HD.chr21.FLAMINGO.res.rds",
  package = "geomeTriD"
))
res <- alignCoor(x, x)
A <- view3dStructure(x, k = 3, renderer = "none")
B <- view3dStructure(res, k = 3, renderer = "none")
B <- lapply(B, function(.ele) {
  .ele$side <- "right"
  .ele
})
threeJsViewer(c(A, B))
```

availableGeometries *Available Geometries*

Description

The Geometries supported by threeJsGeometry class

Usage

```
availableGeometries
```

Format

An object of class `character` of length 18.

Examples

```
availableGeometries
```

create3dGenomicSignals *create 3d Geometry by given genomic signals*

Description

Create a 3d Geometry by given genomic signals for target 3d positions.

Usage

```
create3dGenomicSignals(
  GenoSig,
  targetObj,
  signalTransformFun,
  positionTransformFun,
  genomicScoreRange,
  reverseGenomicSigs,
  type = "segment",
  tag,
  name,
  color = c("gray30", "darkred"),
  rotation = c(0, 0, 0),
  ...
)
```

Arguments

GenoSig	The Genomic signals. An object of GRanges, Pairs, or GInteractions with scores or an object of track.
targetObj	The GRanges object with mcols x0, y0, z0, x1, y1, and z1
signalTransformFun	The transformation function for genomic signals.
positionTransformFun	The transformation function for the coordinates. The function must have input as a data.frame with colnames x0, y0, z0, x1, y1, and z1. And it must have output as same dimension data.frame.
genomicScoreRange	The genomic signals range.
reverseGenomicSigs	Plot the genomic signals in reverse values.
type	The Geometry type.See threeJsGeometry
tag	The tag used to group geometries.
name	The prefix for the name of the geometries.
color	The color of the signal. If there is metadata 'color' in GenoSig this parameter will be ignored.
rotation	The rotations in the x, y and z axis in radians.
...	the parameters for each different type of geometries. If type is 'segments', lwd.maxGenomicSigs (the maximal lwd of the line) is required. If type is 'circle', radius (the radius of the circle) and the maxVal (the value for 2*pi) is required. If type is 'sphere', 'dodecahedron', 'icosahedron', 'octahedron', or 'tetrahedron', radius is required. If type is 'box', 'capsule', 'cylinder', 'cone', or 'torus', if the properties of correspond geometry is not set, they will be set to the transformed score value. If type is 'json', please refer the documentation about BufferGeometryLoader at threejs.org If input 'GenoSig' is an object of Pairs or GInteractions, the type will be set to 'polygon' and topN is used to set how many top events will be plot.

Value

threeJsGeometry objects or NULL

Examples

```
library(GenomicRanges)
GenoSig <- GRanges("chr1", IRanges(seq(1, 100, by = 10), width = 10),
  score = seq.int(10)
)
pos <- matrix(rnorm(303), ncol = 3)
pos <- cbind(
  x0 = pos[seq.int(100), 1],
  x1 = pos[seq.int(101)[-1], 1],
  y0 = pos[seq.int(100), 2],
  y1 = pos[seq.int(101)[-1], 2],
  z0 = pos[seq.int(100), 3],
  z1 = pos[seq.int(101)[-1], 3]
)
```

```
targetObj <- GRanges("chr1", IRanges(seq.int(100), width = 1))
mcols(targetObj) <- pos
ds <- create3dGenomicSignals(GenoSig, targetObj,
  signalTransformFun = function(x) {
    log2(x + 1)
  },
  reverseGenomicSigs = FALSE,
  type = "segment",
  lwd.maxGenomicSigs = 8,
  name = "test",
  tag = "test"
)
threeJsViewer(ds)
```

`createTADGeometries` *create 3d Geometry by given TADs*

Description

Create a 3d Geometry by given TADs for target 3d positions.

Usage

```
createTADGeometries(
  tad,
  targetObj,
  type = "sphere",
  name = "TAD_",
  tag = "TAD",
  alpha = 0.2,
  lwd = 3
)
```

Arguments

`tad`	The TAD. An object of GRanges.
`targetObj`	The GRanges object with mcols x0, y0, z0, x1, y1, and z1
`type`	The Geometry type. default is sphere. Possible types are sphere or segment.
`name`	The prefix for the name of the geometries.
`tag`	The tag used to group geometries.
`alpha`	alpha value. default is 0.2
`lwd`	line width for segment.

Value

threeJsGeometry objects

Examples

```
library(GenomicRanges)
obj <- readRDS(system.file("extdata", "4DNFI1UEG1HD.chr21.FLAMINGO.res.rds",
  package = "geomeTriD"
))
tjg <- view3dStructure(obj, renderer = "none")
pc <- pointCluster(as.data.frame(mcols(obj)))
tads <- split(obj, pc$cluster)
tads <- tads[names(tads)!="0"] # cluster 0 is noise
tads <- unlist(range(GRangesList(tads)))
backbone <- extractBackbonePositions(tjg)
tad_geometries <- createTADGeometries(tads, backbone)
threeJsViewer(tjg, tad_geometries)
```

`extractBackbonePositions` *Extract the backbone coordinates from output of mdsPlot*

Description

Extract the positions from output of mdsPlot and used as the 'targetObj' for function create3dGenomicSignals

Usage

```
extractBackbonePositions(v3d_output)
```

Arguments

`v3d_output`	The output of mdsPlot or view3dStructure for k=3.

Value

An GRanges object with positions of x0, x1, y0, y1, z0 and z1.

Examples

```
library(GenomicRanges)
gi_nij <- readRDS(system.file("extdata", "nij.chr6.51120000.53200000.gi.rds",
                  package = "geomeTriD"))
range_chr6 <- GRanges("chr6", IRanges(51120000, 53200000))
geos <- mdsPlot(gi_nij, range = range_chr6, k = 3, render = "none")
extractBackbonePositions(geos)
```

loopBouquetPlot *plot GInteractions*

Description

plot graph for GInteractions

Usage

```
loopBouquetPlot(
  gi,
  range,
  feature.gr,
  genomicSigs,
  signalTransformFun = function(x) {
     log2(x + 1)
 },
  label_region = FALSE,
  show_edges = TRUE,
  show_cluster = TRUE,
  lwd.backbone = 2,
  col.backbone = "gray",
  lwd.maxGenomicSigs = 8,
  reverseGenomicSigs = TRUE,
  col.backbone_background = "gray70",
  alpha.backbone_background = 0.5,
  lwd.gene = 2,
  lwd.nodeCircle = 1,
  col.nodeCircle = "#DDDDDD25",
  lwd.edge = 2,
  col.edge = "gray80",
  coor_mark_interval = 1e+05,
  col.coor = "black",
  show_coor = TRUE,
  coor_tick_unit = 1000,
  label_gene = TRUE,
  col.tension_line = "black",
  lwd.tension_line = 1,
  length.arrow = NULL,
  safe_text_force = 3,
  method = 1,
  doReduce = FALSE,
  ...
)
```

Arguments

gi	An object of GInteractions
range	The region to plot. an object of GRanges
feature.gr	The annotation features to be added. An object of GRanges.

genomicSigs The genomic signals. An object of GRanges with scores or an object of track.

signalTransformFun
The transformation function for genomic signals.

label_region Label the region node or not.

show_edges Plot the interaction edges or not.

show_cluster Plot the cluster background or not.

lwd.backbone, lwd.gene, lwd.nodeCircle, lwd.edge, lwd.tension_line, lwd.maxGenomicSigs
Line width for the linker, gene, interaction node circle, the dashed line of interaction edges, the tension line and the maximal reversed genomic signal.

col.backbone, col.backbone_background, col.nodeCircle, col.edge, col.tension_line, col.coor
Color for the DNA chain, the compact DNA chain, the node circle, the linker, the tension line and the coordinates marker.

reverseGenomicSigs
Plot the Genomic signals in reverse values.

alpha.backbone_background
Alpha channel for transparency of backbone background.

coor_mark_interval
The coordinates marker interval. Numeric(1). Set to 0 to turn it off. The default value 1e5 means show coordinates every 0.1M bp.

show_coor Show coordinates or not.

coor_tick_unit The bps for every ticks. Default is 1K.

label_gene Show gene symbol or not.

length.arrow Length of the edges of the arrow head (in inches).

safe_text_force
The loops to avoid the text overlapping.

method Plot method. Could be 1 or 2.

doReduce Reduce the GInteractions or not.

... Parameter will be passed to layout_with_fr.

Value

A invisible list with the key points of the plot.

Examples

```
library(InteractionSet)
gi <- readRDS(system.file("extdata", "gi.rds", package = "trackViewer"))
range <- GRanges("chr2", IRanges(234500000, 235000000))
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(org.Hs.eg.db)
feature.gr <- genes(TxDb.Hsapiens.UCSC.hg19.knownGene)
feature.gr <- subsetByOverlaps(feature.gr, range(regions(gi)))
symbols <- mget(feature.gr$gene_id, org.Hs.egSYMBOL, ifnotfound = NA)
feature.gr$label[lengths(symbols) == 1] <- unlist(symbols[lengths(symbols) == 1])
feature.gr$col <- sample(1:7, length(feature.gr), replace = TRUE)
feature.gr$type <- sample(c("cRE", "gene"),
  length(feature.gr),
  replace = TRUE,
```

```
  prob = c(0.1, 0.9)
)
feature.gr$pch <- rep(NA, length(feature.gr))
feature.gr$pch[feature.gr$type == "cRE"] <- 11
loopBouquetPlot(gi, range, feature.gr)
```

`mdsPlot` *Plot genomic interactions by multi-dimensional scaling plot*

Description

This function will convert the interactions scores into a distance matrix and then plot the matrix by multi-dimensional scaling plot.

Usage

```
mdsPlot(
  gi,
  range,
  feature.gr,
  k = 2,
  genomicSigs,
  signalTransformFun = function(x) {
     log2(x + 1)
 },
  lwd.backbone = 2,
  col.backbone = "gray",
  lwd.maxGenomicSigs = 8,
  reverseGenomicSigs = TRUE,
  col.backbone_background = if (k == 2) "gray70" else c("white", "darkred"),
  alpha.backbone_background = 0.5,
  lwd.gene = 3,
  coor_mark_interval = 5e+05,
  col.coor = "black",
  show_coor = TRUE,
  coor_tick_unit = 50000,
  label_gene = TRUE,
  col.tension_line = "black",
  lwd.tension_line = 1,
  length.arrow = NULL,
  safe_text_force = 3,
  square = TRUE,
  renderer = c("rgl", "threejs", "none", "granges"),
  ...
)
```

Arguments

`gi`	An object of GInteractions
`range`	The region to plot. an object of GRanges
`feature.gr`	The annotation features to be added. An object of GRanges.

k The dimension of plot. 2: 2d, 3: 3d.

genomicSigs The genomic signals. An object of GRanges with scores or an object of track.

signalTransformFun
The transformation function for genomic signals.

lwd.backbone, lwd.gene, lwd.tension_line, lwd.maxGenomicSigs
Line width for the linker, gene, interaction node circle, the dashed line of interaction edges, the tension line and the maximal reversed genomic signal.

col.backbone, col.backbone_background, col.tension_line, col.coor
Color for the DNA chain, the compact DNA chain, the node circle, the linker, the tension line and the coordinates marker.

reverseGenomicSigs
Plot the genomic signals in reverse values.

alpha.backbone_background
Alpha channel for transparency of backbone background.

coor_mark_interval
The coordinates marker interval. Numeric(1). Set to 0 to turn it off. The default value 1e5 means show coordinates every 0.1M bp.

show_coor Plot ticks in the line to show the DNA compact tension.

coor_tick_unit The bps for every ticks. Default is 1K.

label_gene Show gene symbol or not.

length.arrow Length of the edges of the arrow head (in inches).

safe_text_force
The loops to avoid the text overlapping.

square A logical value that controls whether control points for the curve are created city-block fashion or obliquely. See grid.curve.

renderer The renderer of the 3D plots. Could be rgl or threejs. The threejs will create a htmlwidgets. If 'none' is set, a list of object will be returned. If 'granges' is set, A GRanges with coordinates will be returned.

... Parameter will be passed to isoMDS.

Value

Coordinates for 2d or 3d.

Examples

```
library(InteractionSet)
gi <- readRDS(system.file("extdata", "nij.chr6.51120000.53200000.gi.rds",
  package = "geomeTriD"
))
range <- GRanges("chr6", IRanges(51120000, 53200000))
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(org.Hs.eg.db)
feature.gr <- genes(TxDb.Hsapiens.UCSC.hg19.knownGene)
feature.gr <- subsetByOverlaps(feature.gr, range(regions(gi)))
symbols <- mget(feature.gr$gene_id, org.Hs.egSYMBOL, ifnotfound = NA)
feature.gr$label[lengths(symbols) == 1] <- unlist(symbols[lengths(symbols) == 1])
feature.gr$col <- sample(1:7, length(feature.gr), replace = TRUE)
feature.gr$type <- sample(c("cRE", "gene"),
  length(feature.gr),
```

```
  replace = TRUE,
  prob = c(0.1, 0.9)
)
mdsPlot(gi, range, feature.gr)
```

`pointCluster` *Perform DBSCAN clustering*

Description

Perform DBSCAN clustering for given 3D coordinates.

Usage

```
pointCluster(xyz, eps = "auto", ...)
```

Arguments

`xyz`	A data.frame with x, y, z coordinates
`eps`	The size (radius) of the epsilon neighborhood. Default 'auto'.
`...`	other parameters could be used by dbscan function except x and eps.

Value

A an object of class dbscan_fast.

Examples

```
xyz <- readRDS(system.file('extdata', '4DNFI1UEG1HD.chr21.FLAMINGO.res.rds',
 package='geomeTriD'))
pc <- pointCluster(xyz)
```

`rglViewer` *rgl Viewer View the 3d structure by rgl.*

Description

rgl Viewer View the 3d structure by rgl.

Usage

```
rglViewer(..., background = "gray")
```

Arguments

`...`	objects of threeJsGeometry.
`background`	background of the main camera.

Value

MULL

Examples

```
obj <- readRDS(system.file("extdata", "4DNFI1UEG1HD.chr21.FLAMINGO.res.rds",
  package = "geomeTriD"
))
feature.gr <- readRDS(system.file("extdata", "4DNFI1UEG1HD.feature.gr.rds",
  package = "geomeTriD"
))
tjg <- view3dStructure(obj,
  k = 3, feature.gr = feature.gr, renderer = "none",
  length.arrow = grid::unit(0.000006, "native")
)
if(interactive()){
  rglViewer(tjg, background = 'white')
}
```

`smooth3dPoints` *Calculate the smoothed curve for input GRanges*

Description

This function will do smooth for given resolution (tile) for inputs and it is important step to prepare the inputs for create3dGenomicSignals and view3dStructure.

Usage

```
smooth3dPoints(obj, resolution = 30, ...)
```

Arguments

`obj`	GRanges object with mcols x, y, and z
`resolution`	number of points at which to evaluate the smooth curve.
`...`	parameters passed to splinefun

Value

GRanges object with smoothed points of x0, y0, z0, x1, y1, and z1.

Examples

```
library(GenomicRanges)
obj <- GRanges("1", IRanges(seq.int(5) * 10, width = 10),
  x = seq.int(5), y = seq.int(5), z = seq.int(5)
)
smooth3dPoints(obj, 5)
```

`threeJsGeometry-class` *Class* `"threeJsGeometry"`

Description

An object of class `"threeJsGeometry"` represents 'three.js' geometry.

Usage

```
threeJsGeometry(...)

## S4 method for signature 'threeJsGeometry'
x$name

## S4 replacement method for signature 'threeJsGeometry'
x$name <- value

## S4 method for signature 'threeJsGeometry'
show(object)
```

Arguments

`...`	Each argument in ... becomes an slot in the new threeJsGeometry.
`x`	an object of threeJsGeometry
`name`	slot name of threeJsGeometry
`value`	value to be assigned
`object`	an object of threeJsGeometry

Slots

`x,y,z` `"numeric"`, specify the x, y, and z coordinates.

`rotation` `"numeric"`, specify the rotations in the x, y and z axis in radians.

`colors` `"character"`, the colors for each geometry.

`type` `"charater"`, the type of the geometry. See availableGeometries.

`side` `'character'`, the side for side by side plot in threeJsViewer.

`layer` `'character'`, the two layer plot in threeJsViewer.

`tag` `'character'`, the tag used to group geometries.

`properties` A `"list"`, the properties to control the geometry.

Examples

```
tjg <- threeJsGeometry()
```

threeJsViewer *threeJs Viewer The htmlwidgets viewer for threeJs.*

Description

threeJs Viewer The htmlwidgets viewer for threeJs.

Usage

```
threeJsViewer(
  ...,
  background = c("#33333388", "#444444DD", "#444444DD", "#33333388"),
  maxRadius = 1,
  maxLineWidth = 50,
  title = NULL,
  width = NULL,
  height = NULL
)
```

Arguments

...	objects of threeJsGeometry.
background	background of the main camera (left and right).
maxRadius	max value of the controls for radius.
maxLineWidth	max value of the controls for line width.
title	the titles of the plot.
width, height	width and height of the widgets.

Details

We convert data frames to JSON by getOption("shiny.json.digits", 7) to avoid the error "Uncaught SyntaxError: Expected ',' or ']' after array element in JSON" for json parse process when handling big data. User can change the option 'shiny.json.digits' larger or smaller number to increase or decrease the digits when converting numbers.

Value

A htmlwidgets widget.

Examples

```
library(GenomicRanges)
flamingo <- system.file("extdata", "4DNFI1UEG1HD.chr21.FLAMINGO.res.rds", package = "geomeTriD")
x <- readRDS(flamingo[[1]])
## resize to bigger value to get better init view
mcols(x) <- as.data.frame(mcols(x)) * 1e5
set.seed(1)
line <- threeJsGeometry(
  x = x$x, y = x$y, z = x$z,
  colors = sample(palette(), length(x), replace = TRUE),
  type = "line",
```

```
  properties = list(size = 4)
)
sphere <- x[sample.int(length(x), 100)]
sphere <- threeJsGeometry(
  x = sphere$x, y = sphere$y, z = sphere$z,
  colors = "red",
  type = "sphere",
  properties = list(radius = 0.08)
)
torus <- x[sample.int(length(x), 100)]
torus <- threeJsGeometry(
  x = torus$x, y = torus$y, z = torus$z,
  colors = "blue",
  type = "torus",
  properties = list(
    radius = 0.08,
    tube = 0.03
  )
)
cylinder <- x[sample.int(length(x), 100)]
cylinder <- threeJsGeometry(
  x = cylinder$x, y = cylinder$y, z = cylinder$z,
  colors = "green",
  type = "cylinder",
  properties = list(
    "height" = 0.07,
    "radiusTop" = 0.05,
    "radiusBottom" = 0.09
  )
)
labels <- x[sample.int(length(x), 5)]
fontURL <- paste0('https://raw.githubusercontent.com/mrdoob/three.js/refs/',
   'heads/dev/examples/fonts/helvetiker_regular.typeface.json')
labels <- threeJsGeometry(
  x = labels$x, y = labels$y, z = labels$z,
  colors = "black",
  type = "text",
  properties = list(
    "label" = "text",
    "font" = readLines(fontURL),
    "size" = .5,
    "depth" = .1
  )
)
threeJsViewer(line, sphere, torus, cylinder)
```

`threeJsViewer-shiny` *Shiny bindings for threeJsViewer*

Description

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

Usage

```
threejsOutput(outputId, width = "100%", height = "600px")

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

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like '100%', '600px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
expr	An expression that generates a threeJsViewer
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

An output or render function that enables the use of the threeJsViewer widget.

Examples

```
if (interactive()) {
  library(GenomicRanges)
 flamingo <- system.file("extdata", "4DNFI1UEG1HD.chr21.FLAMINGO.res.rds", package = "geomeTriD")
  x <- readRDS(flamingo[[1]])
  ## resize to bigger value to get better init view
  mcols(x) <- as.data.frame(mcols(x)) * 1e5
  line <- threeJsGeometry(
    x = x$x, y = x$y, z = x$z,
    colors = sample(palette(), length(x), replace = TRUE),
    type = "line",
    properties = list(size = 4)
  )
  library(shiny)
  runApp(list(
    ui = bootstrapPage(
      threejsOutput("plot")
    ),
    server = function(input, output) {
      output$plot <- renderthreeJsViewer({
        threeJsViewer(line)
      })
    }
  ))
}
```

---

view3dCells *Plot cell xyz data in 2d or 3d*

Description

Plot cell xyz data with grid or rgl package.

Usage

```
view3dCells(
  cells,
  x,
  y,
  z,
  color = "blue",
  colorFun = function(x, pal = seq.int(8)) {
     if (is.character(x))
         x <-
    as.numeric(factor(x))
     limits <- range(x)
     pal[findInterval(x, seq(limits[1],
    limits[2], length.out = length(pal) + 1), all.inside = TRUE)]
 },
  shape = "sphere",
  radius = 0.1,
  tag = "cell",
  renderer = c("rgl", "threejs", "none"),
  ...
)
```

Arguments

cells	A data.frame.
x, y, z	Column names of x, y, z.
color, shape, radius	The column names for color, shape, radius or the value(length=1) of them.
colorFun	The function to map values into colors.
tag	The tag for controller.
renderer	The renderer of the 3D plots. Could be rgl or threejs. The threejs will create a htmlwidgets. If 'none' is set, a list of object will be returned.
...	Not used.

Value

A list of threeJsGeometry objects or a htmlwidget.

Examples

```
cells <- readRDS(system.file("extdata", "pbmc_small.3d.rds",
  package = "geomeTriD"
))
view3dCells(cells,
  x = "umap_1", y = "umap_2", z = "umap_3",
  color = "nCount_RNA",
  renderer = "threejs"
)
```

`view3dStructure` *Plot GRanges xyz data in 2d or 3d*

Description

Plot GRanges xyz data with grid or rgl package.

Usage

```
view3dStructure(
  obj,
  feature.gr,
  genomicSigs,
  region,
  signalTransformFun = function(x) {
     log2(x + 1)
 },
  k = 3,
  renderer = c("rgl", "threejs", "none"),
  lwd.backbone = 2,
  col.backbone = "gray",
  lwd.maxGenomicSigs = 8,
  reverseGenomicSigs = TRUE,
  col.backbone_background = if (k == 2) "gray70" else c("gray30", "darkred"),
  alpha.backbone_background = 0.5,
  lwd.gene = 3,
  coor_mark_interval = 5e+05,
  col.coor = "black",
  show_coor = TRUE,
  coor_tick_unit = 50000,
  label_gene = TRUE,
  col.tension_line = "black",
  lwd.tension_line = 1,
  length.arrow = unit(abs(diff(obj$x))/20, "native"),
  safe_text_force = 3,
  square = TRUE,
  cluster3Dpoints = FALSE,
  ...
)
```

Arguments

`obj`	GRanges object with mcols x, y, and/or z
`feature.gr`	The annotation features to be added. An object of GRanges.
`genomicSigs`	The Genomic signals. An object of GRanges with scores or an object of track.
`region`	A GRanges object with the region to be plot.
`signalTransformFun`	The transformation function for genomic signals.
`k`	The dimension of plot. 2: 2d, 3: 3d.

`renderer` The renderer of the 3D plots. Could be rgl or threejs. The threejs will create a htmlwidgets. If 'none' is set, a list of object will be returned.

`lwd.backbone`, `lwd.gene`, `lwd.tension_line`, `lwd.maxGenomicSigs`
Line width for the linker, gene, interaction node circle, the dashed line of interaction edges, the tension line and the maximal reversed genomic signal.

`col.backbone`, `col.backbone_background`, `col.tension_line`, `col.coor`
Color for the DNA chain, the compact DNA chain, the node circle, the linker, the tension line and the coordinates marker.

`reverseGenomicSigs`
Plot the genomic signals in reverse values.

`alpha.backbone_background`
Alpha channel for transparency of backbone background.

`coor_mark_interval`
The coordinates marker interval. Numeric(1). Set to 0 to turn it off. The default value 1e5 means show coordinates every 0.1M bp.

`show_coor` Plot ticks in the line to show the DNA compact tension.

`coor_tick_unit` The bps for every ticks. Default is 1K.

`label_gene` Show gene symbol or not.

`length.arrow` Length of the edges of the arrow head (in inches).

`safe_text_force`
The loops to avoid the text overlapping.

`square` A logical value that controls whether control points for the curve are created city-block fashion or obliquely. See grid.curve.

`cluster3Dpoints`
A logical value that controls whether cluster the points in 3D. It will be ignored when k=2.

`...` Parameters for create3dGenomicSignals.

Value

Coordinates for 2d or a list of threeJsGeometry objects or a htmlwidget.

Examples

```
obj <- readRDS(system.file("extdata", "4DNFI1UEG1HD.chr21.FLAMINGO.res.rds",
  package = "geomeTriD"
))
feature.gr <- readRDS(system.file("extdata", "4DNFI1UEG1HD.feature.gr.rds",
  package = "geomeTriD"
))
tjg <- view3dStructure(obj,
  k = 3, feature.gr = feature.gr, renderer = "none",
  length.arrow = grid::unit(0.000006, "native")
)
```

Index