Package 'geomeTriD'

November 13, 2026

Type Package

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

Version 1.7.1

Description
The geomeTriD (Three-Dimensional Geometry) Package provides interactive 3D visualization of chromatin structures using the WebGL-based 'three.js' (https://threejs.org/) or the rgl rendering library. It is designed to identify and explore spatial chromatin patterns within genomic regions. The package generates dynamic 3D plots and HTML widgets that integrate seamlessly with Shiny applications, enabling researchers to visualize chromatin organization, detect spatial features, and compare structural dynamics across different conditions and data types.

License MIT + file LICENSE

Depends R (>= 4.4.0)

Imports aricode, BiocGenerics, Biostrings, clue, cluster, dbscan, future.apply, Seqinfo, GenomicRanges, graphics, grDevices, grid, htmlwidgets, igraph, InteractionSet, IRanges, MASS, Matrix, methods, plotrix, progressr, RANN, rgl, rjson, S4Vectors, scales, stats, trackViewer

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

biocViews Visualization

VignetteBuilder knitr

RoxygenNote 8.0.0

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 71100ab

git_last_commit_date 2026-07-22

Repository Bioconductor 3.24

Date/Publication 2026-11-12

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

Maintainer Jianhong Ou <jou@morgridge.org>

Contents

geomeTriD-package . . . 2
alignCoor . . . 3
autoK . . . 4
availableGeometries . . . 5
cellClusters . . . 5
create3dGenomicSignals . . . 7
createTADGeometries . . . 8
extractBackbonePositions . . . 9
fill_NA . . . 10
gaussianBlur . . . 11
loopBouquetPlot . . . 11
mdsPlot . . . 13
pointCluster . . . 15
rglViewer . . . 16
SDC . . . 17
smooth3dPoints . . . 17
spatialDistanceMatrix . . . 18
SRD . . . 20
threeJsGeometry-class . . . 20
threeJsViewer . . . 21
threeJsViewer-shiny . . . 23
view3dCells . . . 24
view3dStructure . . . 25

Index **28**

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

Description

The geomeTriD (Three-Dimensional Geometry) Package provides interactive 3D visualization of chromatin structures using the WebGL-based three.js or the rgl rendering library. It is designed to identify and explore spatial chromatin patterns within genomic regions. The package generates dynamic 3D plots and HTML widgets that integrate seamlessly with Shiny applications, enabling researchers to visualize chromatin organization, detect spatial features, and compare structural dynamics across different conditions and data types.

Author(s)

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

Authors:

- Jianhong Ou <jou@morgridge.org> (ORCID)
- Kenneth Poss <kposs@morgridge.org> [funder]

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))
```

---

autoK *Automate Cluster Number Selection*

Description

Automate cluster number selection using Silhouette Width

Usage

```
autoK(d, hc, max_k)
```

Arguments

d	A dist object.
hc	A hclust object.
max_k	The maximal k.

Value

The best k number.

Examples

```
x <- matrix(rnorm(100), nrow = 5)
d <- dist(x)
hc <- hclust(d)
autoK(d, hc)
```

availableGeometries *Available Geometries*

Description

The Geometries supported by threeJsGeometry class

Usage

```
availableGeometries
```

Examples

```
availableGeometries
```

cellClusters *cluster single cell 3D structures*

Description

Perform Hierarchical clustering for given 3D structures.

Calculate distance for each pair of cells after alignment.

Usage

```
cellClusters(
  xyzs,
  TADs,
  distance_method = "NID",
  cluster_method = "ward.D2",
  rescale = TRUE,
  quite = FALSE,
  parallel = FALSE,
  ...
)

cellDistance(
  xyzs,
  TADs,
  distance_method = c("NID", "RMSD", "SRD", "DSDC", "NMI", "ARI", "AMI"),
  eps,
  k,
  rescale = TRUE,
  quite = FALSE,
  parallel = FALSE,
  ...
)
```

Arguments

xyzs	A list of data.frame with x, y, z coordinates or output of cellDistance.
TADs	A list of index vectors, where each vector represents a TAD. For example, if the first TAD spans the 2nd to 4th coordinates and the second spans the 8th to 10th coordinates, the list would be: list(c(2, 3, 4), c(8, 9, 10)).
distance_method	'SRD', 'DSDC', 'RMSD', 'NMI', 'ARI', 'NID', or 'AMI'. SRD method will first perform clustering and then calculate the Sequence Relabeling Distance SRD. DSDC method will calculate the Euclidean distance of SDC. RMSD method will first do alignment for each cell x, y, z coordinates and the calculate Root Mean Square Deviation (RMSD, the square root of the mean of squared Euclidean distance between corresponding points). ARI, NID, NMI, and AMI method will first perform clustering and then calculate the Adjusted Rand Index (ARI), Normalized information distance (NID), Normalized Mutal Information (NMI), Adjusted Mutual Information (AMI).
cluster_method	The agglomeration method to be used for hclust. Default is 'ward.D2'.
rescale	Re-scale the object to similar size.
quite	Print the message or not.
parallel	Run parallel by future or not.
...	not used.
eps	numeric or 'auto'. The size (radius) of the epsilon neighborhood. If eps is set, use DBSCAN to cluster the points for each cell.
k	numeric or 'auto'. The number of groups. If k is set, use hclust to cluster the points for each cell.

Value

cellClusters return an object of class hclust.

cellDistance return distance matrix as an object of 'dist'

Examples

```
set.seed(1)
xyzs <- lapply(seq.int(20), function(i){
  matrix(sample.int(100, 60, replace = TRUE),
   nrow=20, dimnames=list(NULL, c('x', 'y', 'z')))
})
cd <- cellDistance(xyzs, distance_method='RMSD')
cc <- cellClusters(cd)
# plot(cc)
cutree(cc, k=3)
cd2 <- cellDistance(xyzs, distance_method='SRD', eps=40)
```

`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.
...	other properties.

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, n = "backbone")
```

Arguments

v3d_output	The output of mdsPlot or view3dStructure for k=3.
n	The backbone name of in the inputs.

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)
```

---

fill_NA *fill NA values by upstream and downstream points*

Description

Fill NA values by previous and next points coordinates.

Usage

```
fill_NA(xyz)
```

Arguments

xyz	A matrix or data.frame with columns 'x', 'y', 'z'

Value

A matrix or data.frame.

Examples

```
xyz <- matrix(seq.int(21), ncol=3, dimnames=list(NULL, c('x', 'y', 'z')))
xyz[c(1, 5, 7), ] <- NA
fill_NA(xyz)
```

gaussianBlur *Gaussian blur*

Description

Do Gaussian for the distance matrix.

Usage

```
gaussianBlur(mat, size = 5, sigma = 1, ...)
```

Arguments

mat	A matrix.
size	The kernel size
sigma	The strength of the blur.
...	Not used.

Value

A matrix.

Examples

```
mat <- matrix(runif(100), 10, 10)
blurred_mat <- gaussianBlur(mat, size = 5, sigma = 1)
```

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", quite = FALSE, ...)
```

Arguments

xyz
: A data.frame with x, y, z coordinates

eps
: The size (radius) of the epsilon neighborhood. Default 'auto'.

quite
: Print message or not.

...
: other parameters could be used by dbscan function except x and eps.

Value

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')
}
```

SDC *Distance to centroid*

Description

Calculates the mean of distance from each point to the geometric center (centroid)

Usage

```
SDC(xyz)
```

Arguments

xyz	A data.frame with x, y, z coordinates.

Value

The mean of squared Euclidean distance to the centroid.

Examples

```
xyz <- matrix(seq.int(12), ncol = 3, dimnames=list(NULL, c('x', 'y', 'z')))
SDC(xyz)
```

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)
```

`spatialDistanceMatrix` *Create the spatial distance matrix*

Description

Create the spatial distance matrix for given 3D coordinates.

boundaryScore calculate the boundary score for a distance matrix. Please note that, this boundary score is the reverse of insulation score because we are using the distance matrix but not the interaction matrix.

boundaryScoreTAD assign the TAD boundaries via boundary score.

hierarchicalClusteringTAD assign the TAD boundaries via hierarchical clustering.

compartment calculate the compartment by principal component analysis.

spatialDistanceHeatmap will use base R to plot the spatial distance matrix.

Usage

```
spatialDistanceMatrix(xyz, output = "matrix", fill_NA = FALSE, ...)

boundaryScore(spatialDistances, window = 5, background = 10, ...)

boundaryScoreTAD(
  spatialDistances,
  bin_size,
  window = 5,
  Z_cutoff = 2.3,
  norm = FALSE,
  boundaryScores,
  ...
)

hierarchicalClusteringTAD(spatialDistances, bin_size, window = 5, k, ...)

compartment(xyz.gr, genome, minWidth = 1)

spatialDistanceHeatmap(
  spatialDistances,
 components = c("compartment", "boundaryScoreTAD", "hierarchicalClusteringTAD"),
  col = hcl.colors(n = 12, "OrRd"),
  at = seq(0, 1, length.out = 2),
  label_unit = "M",
  window = 5,
```

```
  background = 10,
  d_cutoff = Inf,
  Z_cutoff = 2.3,
  norm = FALSE,
  Gaussian_blur = FALSE,
  useRaster = FALSE,
  ...
)
```

Arguments

xyz	A GRanges object with x, y, z coordinates
output	"matrix" or "dist".
fill_NA	Fill the missing value or not.
...	Parameters could be used by downstream function.
spatialDistances	The output of spatialDistanceMatrix or the input of spatialDistanceMatrix.
window	The window size for boundary score.
background	The background window size for local background.
bin_size	The bin size.
Z_cutoff	The Z_cutoff value for boundary.
norm	Normalize the boundary score or not.
boundaryScores	The output of boundaryScore.
k	The cluster number. The final TAD numbers will be no greater than this number.
xyz.gr	A GRanges object with x,y,z coordinates.
genome	A BSgenome object
minWidth	The minimal width of input region.
components	The components to plot.
col	a list of colors such as that generated by hcl.colors, gray.colors or similar functions.
at	The label position of X, and Y axis.
label_unit	unit for labels. 'M', 1e6; 'K', 1e3, 'G', 1e9.
d_cutoff	The maximal cutoff value of distance matrix.
Gaussian_blur	Do Gaussian blur or not.
useRaster	logical; if TRUE a bitmap raster is used to plot the image instead of polygons.

Value

A matrix of Euclidean distance with fixed bins.

boundaryScore return a data frame with the boundary score and the Z scores.

boundaryScoreTAD return a list of the index or the positions of coordinates.

hierarchicalClusteringTAD return a list of the index or the positions of coordinates.

compartment return a GRanges object with A,B annotations.

Examples

```
xyz.gr <- readRDS(system.file('extdata', '4DNFI1UEG1HD.chr21.FLAMINGO.res.rds',
 package='geomeTriD'))
bin_size <- 5000 #width(xyz.gr)[1]
sdm <- spatialDistanceMatrix(xyz.gr)
spatialDistanceHeatmap(sdm)
head(boundaryScoreTAD(sdm, bin_size=bin_size))
head(hierarchicalClusteringTAD(sdm, bin_size=bin_size))
library(BSgenome.Hsapiens.UCSC.hg19)
compartment(xyz.gr, genome=BSgenome.Hsapiens.UCSC.hg19)
```

SRD *Sequence Relabeling Distance*

Description

Compares two cluster sequences after best label alignment.

Usage

```
SRD(c1, c2, noise = 0)
```

Arguments

c1, c2	The cluster sequence 1 and 2.
noise	The noise cluster name. Default is 0.

Value

The mean value of hamming distance after label alignment.

Examples

```
c1 <- c(-1, 0, 1, 1, -1, 3, 3, 5, 5, 5)   # `-1` is noise
c2 <- c(-1, 4, 4, 4, -1, 2, 2, 2, 2, 2)   # `-1` is noise
SRD(c1, c2, noise=-1)
```

---

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

∗ **package**
    geomeTriD-package, 2
$,threeJsGeometry-method
        *(*threeJsGeometry-class*)*, 20
$<-,threeJsGeometry-method
        *(*threeJsGeometry-class*)*, 20

alignCoor, 3
autoK, 4
availableGeometries, 5, *21*

boundaryScore *(*spatialDistanceMatrix*)*,
        18
boundaryScoreTAD
        *(*spatialDistanceMatrix*)*, 18

cellClusters, 5
cellDistance *(*cellClusters*)*, 5
compartment *(*spatialDistanceMatrix*)*, 18
create3dGenomicSignals, 7, *17*, *27*
createTADGeometries, 8

extractBackbonePositions, 9

fill_NA, 10

gaussianBlur, 11
geomeTriD *(*geomeTriD-package*)*, 2
geomeTriD-package, 2
GInteractions, *7*, *12*, *14*
GRanges, *7*, *9*, *12*, *14*, *26*
grid.curve, *15*, *27*

hclust, *6*
hierarchicalClusteringTAD
        *(*spatialDistanceMatrix*)*, 18

isoMDS, *15*

layout_with_fr, *13*
loopBouquetPlot, 11

mdsPlot, *10*, 13

Pairs, *7*
pointCluster, 15

renderthreeJsViewer
        *(*threeJsViewer-shiny*)*, 23
rglViewer, 16

SDC, *6*, 17
show,threeJsGeometry-method
        *(*threeJsGeometry-class*)*, 20
smooth3dPoints, 17
spatialDistanceHeatmap
        *(*spatialDistanceMatrix*)*, 18
spatialDistanceMatrix, 18
splinefun, *17*
SRD, *6*, 20

threeJsGeometry, *5*, *7–9*
threeJsGeometry
        *(*threeJsGeometry-class*)*, 20
threeJsGeometry-class, 20
threejsOutput *(*threeJsViewer-shiny*)*, 23
threeJsViewer, *21*, 21
threeJsViewer-shiny, 23
track, *7*, *12*, *14*, *26*

view3dCells, 24
view3dStructure, *10*, *17*, 25