Package ‘delaunay’

October 19, 2022

Type Package

Title 2d, 2.5d, and 3d Delaunay Tessellations

Version 1.1.1

Author Stéphane Laurent

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Description Construction and visualization of 2d Delaunay triangulations, possibly constrained, 2.5d (i.e. elevated) Delaunay triangulations, and 3d Delaunay triangulations.

License GPL-3

URL https://github.com/stla/delaunay

BugReports https://github.com/stla/delaunay/issues

Imports gplots, graphics, randomcoloR, Rcpp (>= 1.0.8), rgl, Rvcg, utils

Suggests uniformly

LinkingTo Rcpp, RcppCGAL, RcppEigen, BH

Encoding UTF-8

RoxygenNote 7.2.0

SystemRequirements C++ 17, gmp, mpfr

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-10-19 08:45:08 UTC

R topics documented:

`delaunay` *Delaunay tessellation*

Description

Delaunay tessellation of a set of 2D or 3D points.

Usage

```
delaunay(points, elevation = FALSE, constraints = NULL, quick3d = FALSE)
```

Arguments

`points`	numeric matrix which stores the points, one point per row
`elevation`	if points are three-dimensional and `elevation=TRUE`, then the function performs an elevated two-dimensional Delaunay triangulation, using the `z` coordinates of the points for the elevations; see the example
`constraints`	*for 2D only*, some edges to perform a constrained Delaunay triangulation, given as an integer matrix with two columns (each row provides the indices of the two points forming the edge); `NULL` for no constraint
`quick3d`	Boolean, for 3D only; if `FALSE`, there is more information in the output about the Delaunay tessellation; see the **Value** section for details

Value

The Delaunay tessellation.

- **If the dimension is 2** and `constraints=NULL`, the returned value is a list with three fields: `faces`, `edges` and `area`. The `faces` field contains an integer matrix with three columns; each row represents a triangle whose each vertex is given by the index (row number) of this point in the `points` matrix. The `edges` field also contains an integer matrix with three columns. The two first integers of a row are the indices of the two points which form the edge. The third column, named `border`, only contains some zeros and some ones; a border (exterior) edge is labelled by a `1`. The `area` field contains only a number: the area of the triangulated region (that is, the area of the convex hull of the points).
- **If the dimension is 2** and `constraints` is not `NULL`, the returned value is a list with four fields: `faces`, `edges`, `constraints`, and `area`. The `faces` field contains an integer matrix with three columns; each row represents a triangle whose each vertex is given by the index (row number) of this point in the `points` matrix. The `edges` field is a dataframe with four columns. The first two columns provide the edges of the triangulation; they are given by row, the two integers of a row are the indices of the two points which form the edge. Each integer of the third column is the index of the face the corresponding edge belongs to. The fourth column, named `border`, only contains some zeros and some ones; a border edge is labelled by a `1`. The `constraints` field is an integer matrix with two columns, it represents the constraint edges. Finally, the `area` field contains only a number: the area of the triangulated region.

- **If the dimension is 3**, the returned value is a list with four fields: `cells`, `facets`, `edges`, and `volume`. The `cells` field represents the tetrahedra which form the tessellation. The `facets` field represents the faces of these tetrahedra, some triangles. The `edges` field represents the edges of these triangles. The `volume` field provides only one number, the volume of the tessellation, in other words the volume of the convex hull of the given points. If `quick3d=TRUE`, then `cells`, `facets` and `edges` are integer matrices with four, three, and two columns respectively; each integer is a vertex index. If `quick3d=FALSE`, the `cells` field is a list of lists. Each sublist is composed of three fields: `cell` provides the indices of the four vertices of the corresponding tetrahedron, `faces` provides the indices of the four faces of the tetrahedron, that is to say the row number of the `facets` field which represents this face, and finally there is a `volume` field which provides the volume of the tetrahedron. The `facets` field is an integer matrix with four columns. The three first integers of a row are the indices of the points which form the corresponding facet. The fourth column, named `onhull` is composed of zeros and ones only, and a `1` means that the corresponding facet lies on the convex hull of the points. The `edges` field contains an integer matrix with three columns. Each row represents an edge, given by the two indices of the points which form this edge, and the third integer, in the column named `onhull` is a `0/1` indicator of whether the edge lies on the convex hull. Finally the `volume` field provides only one number, the volume of the tessellation (i.e. the volume of the convex hull of the points).
- **If** `elevation=TRUE`, the returned value is a list with five fields: `mesh`, `edges`, `faceVolumes`, `volume` and `area`. The `mesh` field is an object of class `mesh3d`, ready for plotting with the **rgl** package. The `edges` field provides the indices of the edges, given as an integer matrix with two columns. The `faceVolumes` field is a numeric vector, it provides the volumes under the faces that can be found in the `mesh` field. The `volume` field provides the sum of these volumes, that is to say the total volume under the triangulated surface. Finally, the `area` field provides the sum of the areas of all triangles, thereby approximating the area of the triangulated surface.

Examples

```
library(delaunay)
# elevated Delaunay triangulation ####
f <- function(x, y){
  2 * exp(-(x^2 + y^2)) # integrate to 2pi
}
x <- y <- seq(-4, 4, length.out = 50)
grd <- transform(expand.grid(x = x, y = y), z = f(x, y))
del <- delaunay(as.matrix(grd), elevation = TRUE)
# `del` is a list; its first component is a mesh representing the surface:
mesh <- del[["mesh"]]
library(rgl)
open3d(windowRect = c(50, 50, 562, 562))
shade3d(mesh, color = "limegreen")
wire3d(mesh)
# in `del` you can also found the volume under the surface, which should
#   approximate the integral of the function:
del[["volume"]]
```

mesh2d *Convert a 2D Delaunay triangulation to a 'rgl' mesh*

Description

Makes a 'rgl' mesh (mesh3d object) from a 2D Delaunay triangulation, unconstrained or constrained.

Usage

```
mesh2d(triangulation)
```

Arguments

triangulation	an output of delaunay executed with 2D points

Value

A list with three fields; mesh, a mesh3d object, borderEdges, a numeric matrix that can be used with segments3d to plot the border edges, and constraintEdges, a numeric matrix that can be used with segments3d to plot the constraint edges which are not border edges.

See Also

plotDelaunay2D

Examples

```
library(delaunay)
# outer and inner hexagons ####
nsides <- 6L
angles <- seq(0, 2*pi, length.out = nsides+1L)[-1L]
outer_points <- cbind(cos(angles), sin(angles))
inner_points <- outer_points / 2
points <- rbind(outer_points, inner_points)
# constraint edges
indices <- 1L:nsides
edges <- cbind(
  indices, c(indices[-1L], indices[1L])
)
edges <- rbind(edges, edges + nsides)
# constrained Delaunay triangulation
del <- delaunay(points, constraints = edges)
# mesh
m2d <- mesh2d(del)
mesh <- m2d[["mesh"]]
# plot all edges with `wire3d`
library(rgl)
open3d(windowRect = c(100, 100, 612, 612))
```

```
shade3d(mesh, color = "red", specular = "orangered")
wire3d(mesh, color = "black", lwd = 3, specular = "black")
# plot only the border edges
open3d(windowRect = c(100, 100, 612, 612))
shade3d(mesh, color = "darkred", specular = "firebrick")
segments3d(m2d[["borderEdges"]], lwd = 3)
```

---

plotDelaunay2D *Plot 2D Delaunay triangulation*

Description

Plot a constrained or unconstrained 2D Delaunay triangulation.

Usage

```
plotDelaunay2D(
  triangulation,
  col_edges = "black",
  col_borders = "red",
  col_constraints = "green",
  fillcolor = "distinct",
  hue = "random",
  luminosity = "light",
  lty_edges = par("lty"),
  lwd_edges = par("lwd"),
  lty_borders = par("lty"),
  lwd_borders = par("lwd"),
  lty_constraints = par("lty"),
  lwd_constraints = par("lwd"),
  ...
)
```

Arguments

`triangulation`	an output of `delaunay` without constraints (`constraints=NULL`) or with constraints
`col_edges`	the color of the edges of the triangles which are not border edges nor constraint edges; `NULL` for no color
`col_borders`	the color of the border edges; note that the border edges can contain the constraint edges for a constrained Delaunay tessellation; `NULL` for no color
`col_constraints`	for a constrained Delaunay tessellation, the color of the constraint edges which are not border edges; `NULL` for no color
`fillcolor`	controls the filling colors of the triangles, either `NULL` for no color, a single color, `"random"` to get multiple colors with `randomColor`, or `"distinct"` get multiple colors with `distinctColorPalette`

hue, luminosity
: if color = "random", these arguments are passed to randomColor

lty_edges, lwd_edges
: graphical parameters for the edges which are not border edges nor constraint edges

lty_borders, lwd_borders
: graphical parameters for the border edges

lty_constraints, lwd_constraints
: in the case of a constrained Delaunay triangulation, graphical parameters for the constraint edges which are not border edges

...
: arguments passed to points for the vertices, such as type="n" or asp=1

Value

No value, just renders a 2D plot.

See Also

mesh2d for an interactive plot

Examples

```
library(delaunay)
# random points in a square ####
square <- rbind(
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1)
)
library(uniformly)
set.seed(314)
ptsinsquare <- runif_in_cube(10L, d = 2L)
pts <- rbind(square, ptsinsquare)
d <- delaunay(pts)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay2D(
  d, type = "n", xlab = NA, ylab = NA, axes = FALSE, asp = 1,
  fillcolor = "random", luminosity = "dark", lwd_borders = 3
)
par(opar)

# a constrained Delaunay triangulation: outer and inner hexagons ####
nsides <- 6L
angles <- seq(0, 2*pi, length.out = nsides+1L)[-1L]
outer_points <- cbind(cos(angles), sin(angles))
inner_points <- outer_points / 2
points <- rbind(outer_points, inner_points)
# constraint edges
indices <- 1L:nsides
edges <- cbind(
  indices, c(indices[-1L], indices[1L])
)
edges <- rbind(edges, edges + nsides)
```

```
# constrained Delaunay triangulation
d <- delaunay(points, constraints = edges)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay2D(
  d, type = "p", pch = 19, xlab = NA, ylab = NA, axes = FALSE, asp = 1,
  fillcolor = "orange", lwd_borders = 3
)
par(opar)

# another constrained Delaunay tesselation: a face ####
V <- as.matrix(read.table(
  system.file("extdata", "face_vertices.txt", package = "delaunay")
))[, c(2L, 3L)]
E <- as.matrix(read.table(
  system.file("extdata", "face_edges.txt", package = "delaunay")
))[, c(2L, 3L)]
d <- delaunay(points = V, constraints = E)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay2D(
  d, type = "n", xlab = NA, ylab = NA, axes = FALSE, asp = 1,
  fillcolor = "salmon", col_borders = "black",
  lwd_borders = 3, lwd_constraints = 2, lty_edges = "dashed"
)
par(opar)
```

`plotDelaunay3D` *Plot 3D Delaunay tessellation*

Description

Plot a 3D Delaunay tessellation with **rgl**.

Usage

```
plotDelaunay3D(
  tessellation,
  color = "distinct",
  hue = "random",
  luminosity = "light",
  alpha = 0.3,
  ...
)
```

Arguments

`tessellation`	the output of `delaunay` with 3D points
`color`	controls the filling colors of the tetrahedra, either `FALSE` for no color, `"random"` to use `randomColor`, or `"distinct"` to use `distinctColorPalette`

`hue, luminosity`
: if `color="random"`, these arguments are passed to `randomColor`

`alpha`
: opacity, number between 0 and 1

`...`
: arguments passed to `material3d`

Value

No value, just renders a 3D plot.

Examples

```
library(delaunay)
pts <- rbind(
  c(-5, -5,  16),
  c(-5,  8,   3),
  c(4,  -1,   3),
  c(4,  -5,   7),
  c(4,  -1, -10),
  c(4,  -5, -10),
  c(-5,  8, -10),
  c(-5, -5, -10)
)
tess <- delaunay(pts)
library(rgl)
open3d(windowRect = c(50, 50, 562, 562))
plotDelaunay3D(tess)
```

Index