Package ‘RCDT’

August 7, 2022

Type Package

Title Fast 2D Constrained Delaunay Triangulation

Version 1.2.0

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Description Performs 2D Delaunay triangulation, constrained or unconstrained, with the help of the C++ library 'CDT'. A function to plot the triangulation is provided. The constrained Delaunay triangulation has applications in geographic information systems.

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

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

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

Suggests knitr, rmarkdown, testthat (>= 3.0.0), uniformly, viridisLite

LinkingTo BH, Rcpp, RcppArmadillo

SystemRequirements C++ 14

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation yes

Author Stéphane Laurent [aut, cre],
Artem Amirkhanov [cph] (CDT library)

Repository CRAN

Date/Publication 2022-08-07 21:40:02 UTC

R topics documented:

RCDT-package *Fast 2D Constrained Delaunay Triangulation*

Description

Performs 2D Delaunay triangulation, constrained or unconstrained, with the help of the C++ library 'CDT'. A function to plot the triangulation is provided. The constrained Delaunay triangulation has applications in geographic information systems.

Details

The DESCRIPTION file:

Type:	Package
Package:	RCDT
Title:	Fast 2D Constrained Delaunay Triangulation
Version:	1.2.0
Authors@R:	c(person("Stéphane", "Laurent", , "laurent_step@outlook.fr", role = c("aut", "cre")), person("Artem
Maintainer:	Stéphane Laurent <laurent_step@outlook.fr>
Description:	Performs 2D Delaunay triangulation, constrained or unconstrained, with the help of the C++ library
License:	GPL-3
URL:	https://github.com/stla/RCDT
BugReports:	https://github.com/stla/RCDT/issues
Imports:	gplots, graphics, randomcoloR, Rcpp (>= 1.0.8), rgl, Rvcg
Suggests:	knitr, rmarkdown, testthat (>= 3.0.0), uniformly, viridisLite
LinkingTo:	BH, Rcpp, RcppArmadillo
SystemRequirements:	C++ 14
VignetteBuilder:	knitr
Config/testthat/edition:	3
Encoding:	UTF-8
RoxygenNote:	7.2.1
Author:	Stéphane Laurent [aut, cre], Artem Amirkhanov [cph] (CDT library)

Index of help topics:

```
RCDT-package            Fast 2D Constrained Delaunay Triangulation
delaunay                2D Delaunay triangulation
delaunayArea            Area of Delaunay triangulation
plotDelaunay            Plot 2D Delaunay triangulation
```

The `delaunay` function is the main function of this package. It can build a Delaunay triangulation of a set of 2D points, constrained or unconstrained. The constraints are defined by the `edges` argument.

Author(s)

NA

Maintainer: Stéphane Laurent <laurent_step@outlook.fr>

`delaunay` *2D Delaunay triangulation*

Description

Performs a (constrained) Delaunay triangulation of a set of 2d points.

Usage

```
delaunay(points, edges = NULL, elevation = FALSE)
```

Arguments

`points`	a numeric matrix with two or three columns (three colums for an elevated Delaunay triangulation)
`edges`	the edges for the constrained Delaunay triangulation, an integer matrix with two columns; `NULL` for no constraint
`elevation`	Boolean, whether to perform an elevated Delaunay triangulation (also known as 2.5D Delaunay triangulation)

Value

A list. There are three possibilities. #'

- **If the dimension is 2** and `edges=NULL`, the returned value is a list with three fields: `vertices`, `mesh` and `edges`. The `vertices` field contains the given vertices. The `mesh` field is an object of class `mesh3d`, ready for plotting with the **rgl** package. The `edges` field provides the indices of the vertices of the edges, given by the first two columns of a three-columns integer matrix. The third column, named `border`, only contains some zeros and some ones; a border (exterior) edge is labelled by a `1`.
- **If the dimension is 2** and `edges` is not `NULL`, the returned value is a list with four fields: `vertices`, `mesh`, `edges`, and `constraints`. The `vertices` field contains the vertices of the triangulation. They coincide with the given vertices if the constraint edges do not intersect; otherwise there are the intersections in addition to the given vertices. The `mesh` and `edges` fields are similar to the previous case, the unconstrained Delaunay triangulation. The `constraints` field is an integer matrix with two columns, it represents the constraint edges. They are not the same as the ones provided by the user if these ones intersect. If they do not intersect, then in general these are the same, but not always, in some rare corner cases.
- **If** `elevation=TRUE`, the returned value is a list with five fields: `vertices`, `mesh`, `edges`, `volume`, and `surface`. The `vertices` field contains the given vertices. The `mesh` field is an object of class `mesh3d`, ready for plotting with the **rgl** package. The `edges` field is similar to the previous cases. The `volume` field provides a number, the sum of the volumes under the Delaunay triangles, that is to say the total volume under the triangulated surface. Finally, the `surface` field provides the sum of the areas of all triangles, thereby approximating the area of the triangulated surface.

Note

The triangulation can depend on the order of the points; this is shown in the examples.

Examples

```
library(RCDT)
# random points in a square ####
set.seed(314)
library(uniformly)
square <- rbind(
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1)
)
pts_in_square <- runif_in_cube(10L, d = 2L)
pts <- rbind(square, pts_in_square)
del <- delaunay(pts)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay(
  del, type = "n", xlab = NA, ylab = NA, asp = 1,
  fillcolor = "random", luminosity = "light", lty_edges = "dashed"
)
par(opar)

# the order of the points matters ####
# the Delaunay triangulation is not unique in general;
#   it can depend on the order of the points
points <- cbind(
  c(1, 2, 1, 3, 2, 1, 4, 3, 2, 1, 4, 3, 2, 4, 3, 4),
  c(1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 3, 4, 4)
)
del <- delaunay(points)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay(
  del, type = "p", pch = 19, xlab = NA, ylab = NA, axes = FALSE,
  asp = 1, lwd_edges = 2, lwd_borders = 3
)
par(opar)
# now we randomize the order of the points
set.seed(666L)
points2 <- points[sample.int(nrow(points)), ]
del2 <- delaunay(points2)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay(
  del2, type = "p", pch = 19, xlab = NA, ylab = NA, axes = FALSE,
  asp = 1, lwd_edges = 2, lwd_borders = 3
)
par(opar)

# a constrained Delaunay triangulation: outer and inner dodecagons ####
# points
nsides <- 12L
angles <- seq(0, 2*pi, length.out = nsides+1L)[-1L]
points <- cbind(cos(angles), sin(angles))
```

```
points <- rbind(points, points/1.5)
# constraint edges
indices <- 1L:nsides
edges_outer <- cbind(
  indices, c(indices[-1L], indices[1L])
)
edges_inner <- edges_outer + nsides
edges <- rbind(edges_outer, edges_inner)
# constrained Delaunay triangulation
del <- delaunay(points, edges)
# plot
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay(
  del, type = "n", fillcolor = "yellow", lwd_borders = 2, asp = 1,
  axes = FALSE, xlab = NA, ylab = NA
)
par(opar)

# another constrained Delaunay triangulation: a face ####
V <- read.table(
  system.file("extdata", "face_vertices.txt", package = "RCDT")
)
E <- read.table(
  system.file("extdata", "face_edges.txt", package = "RCDT")
)
del <- delaunay(
  points = as.matrix(V)[, c(2L, 3L)], edges = as.matrix(E)[, c(2L, 3L)]
)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay(
  del, type="n", col_edges = NULL, fillcolor = "salmon",
  col_borders = "black", col_constraints = "purple",
  lwd_borders = 3, lwd_constraints = 3,
  asp = 1, axes = FALSE, xlab = NA, ylab = NA
)
par(opar)
```

`delaunayArea` *Area of Delaunay triangulation*

Description

Computes the area of a region subject to Delaunay triangulation.

Usage

```
delaunayArea(del)
```

Arguments

`del`	an output of `delaunay` executed with `elevation=FALSE`

Value

A number, the area of the region triangulated by the Delaunay triangulation.

Examples

```
library(RCDT)
# random points in a square ####
set.seed(666L)
library(uniformly)
square <- rbind(
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1)
)
pts <- rbind(square, runif_in_cube(8L, d = 2L))
del <- delaunay(pts)
delaunayArea(del)

# a constrained Delaunay triangulation: outer and inner squares ####
innerSquare <- rbind( # the hole
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1)
) # area: 4
outerSquare <- 2*innerSquare # area: 16
points <- rbind(innerSquare, outerSquare)
edges_inner <- rbind(c(1L, 2L), c(2L, 3L), c(3L, 4L), c(4L, 1L))
edges_outer <- edges_inner + 4L
edges <- rbind(edges_inner, edges_outer)
del <- delaunay(points, edges = edges)
delaunayArea(del) # 16-4
```

`plotDelaunay` *Plot 2D Delaunay triangulation*

Description

Plot a constrained or unconstrained 2D Delaunay triangulation.

Usage

```
plotDelaunay(
  del,
  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

- `del` — an output of `delaunay` without constraints (`edges=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 triangulation; `NULL` for no color
- `col_constraints` — for a constrained Delaunay triangulation, 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`, `"distinct"` get multiple colors with `distinctColorPalette`, or a vector of colors, one color for each triangle; in this case the the colors will be assigned in the order they are provided but after the triangles have been circularly ordered (see the last example)
- `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 `plot` for the vertices, such as `type="n"`, `asp=1`, `axes=FALSE`, etc

Value

No value, just renders a 2D plot.

See Also

The `mesh` field in the output of `delaunay` for an interactive plot. Other examples of `plotDelaunay` are given in the examples of `delaunay`.

Examples

```
library(RCDT)
# random points in a square ####
```

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

# a constrained Delaunay triangulation: pentagram ####
# vertices
R <- sqrt((5-sqrt(5))/10)     # outer circumradius
r <- sqrt((25-11*sqrt(5))/10) # circumradius of the inner pentagon
k <- pi/180 # factor to convert degrees to radians
X <- R * vapply(0L:4L, function(i) cos(k * (90+72*i)), numeric(1L))
Y <- R * vapply(0L:4L, function(i) sin(k * (90+72*i)), numeric(1L))
x <- r * vapply(0L:4L, function(i) cos(k * (126+72*i)), numeric(1L))
y <- r * vapply(0L:4L, function(i) sin(k * (126+72*i)), numeric(1L))
vertices <- rbind(
  c(X[1L], Y[1L]),
  c(x[1L], y[1L]),
  c(X[2L], Y[2L]),
  c(x[2L], y[2L]),
  c(X[3L], Y[3L]),
  c(x[3L], y[3L]),
  c(X[4L], Y[4L]),
  c(x[4L], y[4L]),
  c(X[5L], Y[5L]),
  c(x[5L], y[5L])
)
# constraint edge indices (= boundary)
edges <- cbind(1L:10L, c(2L:10L, 1L))
# constrained Delaunay triangulation
del <- delaunay(vertices, edges)
# plot
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay(
  del, type = "n", asp = 1, fillcolor = "distinct", lwd_borders = 3,
  xlab = NA, ylab = NA, axes = FALSE
)
par(opar)
# interactive plot with 'rgl'
mesh <- del[["mesh"]]
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 - we could find them in `del[["edges"]]`
  # but we use the 'rgl' function `getBoundary3d` instead
open3d(windowRect = c(100, 100, 612, 612))
shade3d(mesh, color = "darkred", specular = "firebrick")
shade3d(getBoundary3d(mesh), lwd = 3)

# an example where `fillcolor` is a vector of colors ####
n <- 50L # number of sides of the outer polygon
angles1 <- head(seq(0, 2*pi, length.out = n + 1L), -1L)
outer_points <- cbind(cos(angles1), sin(angles1))
m <- 5L # number of sides of the inner polygon
angles2 <- head(seq(0, 2*pi, length.out = m + 1L), -1L)
phi <- (1+sqrt(5))/2 # the ratio  2-phi  will yield a perfect pentagram
inner_points <- (2-phi) * cbind(cos(angles2), sin(angles2))
points <- rbind(outer_points, inner_points)
# constraint edges
indices <- 1L:n
edges_outer <- cbind(indices, c(indices[-1L], indices[1L]))
indices <- n + 1L:m
edges_inner <- cbind(indices, c(indices[-1L], indices[1L]))
edges <- rbind(edges_outer, edges_inner)
# constrained Delaunay triangulation
del <- delaunay(points, edges)
# there are 55 triangles:
del[["mesh"]]
# we make a cyclic palette of colors:
colors <- viridisLite::turbo(28)
colors <- c(colors, rev(colors[-1L]))
# plot
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay(
  del, type = "n", asp = 1, lwd_borders = 3, col_borders = "black",
  fillcolor = colors, col_edges = "black", lwd_edges = 1.5,
  axes = FALSE, xlab = NA, ylab = NA
)
par(opar)
```

Index