Package ‘tessellation’

October 14, 2022

Type      Package
Title     Delaunay and Voronoï Tessellations
Version   2.1.0
Maintainer Stéphane Laurent <laurent_step@outlook.fr>
Description Delaunay and Voronoï tessellations, with emphasis on the two-dimensional and the three-dimensional cases (the package provides functions to plot the tessellations for these cases). Delaunay tessellations are computed in C with the help of the 'Qhull' library <http://www.qhull.org/>.
License   GPL-3
Encoding  UTF-8
Imports   hash, rgl, randomcoloR, utils, R6, sets, cxhull, graphics, scales, english, interp, Rvcg
Suggests  rmarkdown, knitr, uniformly, viridisLite, paletteer
URL       https://github.com/stla/tessellation,
           https://stla.github.io/tessellation/
BugReports https://github.com/stla/tessellation/issues
RoxygenNote 7.1.2
NeedsCompilation yes
Author     Stéphane Laurent [aut, cre],
           C. B. Barber [cph] (author of the Qhull library),
           The Geometry Center [cph]
Repository  CRAN
Date/Publication  2022-04-05 15:00:02 UTC

R topics documented:
cellVertices .......................................................... 2
centricCuboctahedron ............................................. 3
delaunay .............................................................. 3
Edge2 ............................................................... 6
cellVertices

Vertices of a bounded cell

Description

Get all vertices of a bounded cell, without duplicates.

Usage

cellVertices(cell, check.bounded = TRUE)

Arguments

cell a bounded Voronoï cell

check.bounded Boolean, whether to check that the cell is bounded; set to FALSE for a small speed gain if you know that the cell is bounded

Value

A matrix, each row represents a vertex.

Examples

library(tessellation)
d <- delaunay(centricCuboctahedron())
v <- voronoi(d)
cell13 <- v[[13]]
isBoundedCell(cell13) # TRUE
library(rgl)
open3d(windowRect = c(50, 50, 562, 562))
invisible(lapply(cell13[['cell']], function(edge){

}})
centricCuboctahedron

```r

edge$plot(edgeAsTube = TRUE, tubeRadius = 0.025, tubeColor = "yellow")
}

)'

} )

centricCuboctahedron <- cellVertices(cell13)
spheres3d(cellvertices, radius = 0.1, color = "green")

delaunay

delaunay(points, atinfinity = FALSE, degenerate = FALSE, exteriorEdges = FALSE, elevation = FALSE)

Arguments

points the points given as a matrix, one point per row
atinfinity Boolean, whether to include a point at infinity
degenerate Boolean, whether to include degenerate tiles
exteriorEdges Boolean, for dimension 3 only, whether to return the exterior edges (see below)
elevation Boolean, only for three-dimensional points; if TRUE, the function performs an elevate Delaunay triangulation (also called 2.5D Delaunay triangulation), using the third coordinate of a point as its elevation; see the example

---

centricCuboctahedron  Centric cuboctahedron

Description

A cuboctahedron (12 vertices), with a point added at its center.

Usage

centricCuboctahedron()

Value

A numeric matrix with 13 rows and 3 columns.

---

delaunay  Delaunay triangulation

Description

Delaunay triangulation (or tessellation) of a set of points.

Usage

delaunay(points, atinfinity = FALSE, degenerate = FALSE, exteriorEdges = FALSE, elevation = FALSE)

Arguments

points the points given as a matrix, one point per row
atinfinity Boolean, whether to include a point at infinity
degenerate Boolean, whether to include degenerate tiles
exteriorEdges Boolean, for dimension 3 only, whether to return the exterior edges (see below)
elevation Boolean, only for three-dimensional points; if TRUE, the function performs an elevated Delaunay triangulation (also called 2.5D Delaunay triangulation), using the third coordinate of a point as its elevation; see the example
Value

If the function performs an elevated Delaunay tessellation, then the returned value is a list with four fields: `mesh`, `edges`, `volume`, and `surface`. The mesh field is an object of class `mesh3d`, ready for plotting with the `rgl` package. The edges field is an integer matrix which provides the indices of the vertices of the edges, and an indicator of whether an edge is a border edge; this matrix is obtained with `vcgGetEdge`. The volume field provides 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 the Delaunay triangles, thus this is an approximate value of the area of the surface that is triangulated. The elevated Delaunay tessellation is built with the help of the `interp` package.

Otherwise, the function returns the Delaunay tessellation with many details, in a list. This list contains five fields:

- `vertices` the vertices (or sites) of the tessellation; these are the points passed to the function
- `tiles` the tiles of the tessellation (triangles in dimension 2, tetrahedra in dimension 3)
- `tilefacets` the facets of the tiles of the tessellation
- `mesh` a `rgl` mesh (`mesh3d` object)
- `edges` a two-columns integer matrix representing the edges, each row represents an edge; the two integers of a row are the indices of the two points which form the edge.

In dimension 3, the list contains an additional field `exteriorEdges` if you set `exteriorEdges = TRUE`. This is the list of the exterior edges, represented as `Edge3` objects. This field is involved in the function `plotDelaunay3D`.

The `vertices` field is a list with the following fields:

- `id` the id of the vertex; this is nothing but the index of the corresponding point passed to the function
- `neighvertices` the ids of the vertices of the tessellation connected to this vertex by an edge
- `neightilefacets` the ids of the tile facets this vertex belongs to
- `neightiles` the ids of the tiles this vertex belongs to

The `tiles` field is a list with the following fields:

- `id` the id of the tile
- `simplex` a list describing the simplex (that is, the tile); this list contains four fields: `vertices`, a hash giving the simplex vertices and their id, `circumcenter`, the circumcenter of the simplex, `circumradius`, the circumradius of the simplex, and `volume`, the volume of the simplex
- `facets` the ids of the facets of this tile
- `neighbors` the ids of the tiles adjacent to this tile
- `family` two tiles have the same family if they share the same circumcenter; in this case the family is an integer, and the family is `NA` for tiles which do not share their circumcenter with any other tile
- `orientation` 1 or -1, an indicator of the orientation of the tile

The `tilefacets` field is a list with the following fields:

- `id` the id of this tile facet
**subsimplex**  a list describing the subsimplex (that is, the tile facet); this list is similar to the *simplex* list of *tiles*

**facetOf**  one or two ids, the id(s) of the tile this facet belongs to

**normal**  a vector, the normal of the tile facet

**offset**  a number, the offset of the tile facet

Note

The package provides the functions `plotDelaunay2D` to plot a 2D Delaunay tessellation and `plotDelaunay3D` to plot a 3D Delaunay tessellation. But there is no function to plot an elevated Delaunay tessellation; the examples show how to plot such a Delaunay tessellation.

See Also

`getDelaunaySimplicies`

Examples

```r
library(tessellation)
points <- rbind(
  c(0.5,0.5,0.5),
  c(0,0,0),
  c(0,0,1),
  c(0,1,0),
  c(0,1,1),
  c(1,0,0),
  c(1,0,1),
  c(1,1,0),
  c(1,1,1))
del <- delaunay(points)
del$vertices[[1]]
del$tiles[[1]]
del$tilefacets[[1]]

# an elevated Delaunay tessellation ####
f <- function(x, y){
  dnorm(x) * dnorm(y)
}
x <- y <- seq(-5, 5, length.out = 50)
grd <- expand.grid(x = x, y = y) # grid on the xy-plane
points <- as.matrix(transform( # data (x_i, y_i, z_i)
  grd, z = f(x, y))
)
del <- delaunay(points, elevation = TRUE)
del["volume"] # close to 1, as expected
# plotting
library(rgl)
mesh <- del["mesh"]
open3d(windowRect = c(100, 100, 612, 356), zoom = 0.6)
aspect3d(1, 1, 20)
```

shade3d(mesh, color = "limegreen")
wire3d(mesh)

# another elevated Delaunay triangulation, to check the correctness of
# the calculated surface ####
library(Rvcg)
library(rgl)
cap <- vcgSphericalCap(angleRad = pi/2, subdivision = 3)
open3d(windowRect = c(100, 100, 612, 356), zoom = 0.6)
shade3d(cap, color = "lawngreen")
wire3d(cap)

# exact value of the surface of the spherical cap:
R <- 1
h <- R * (1 - sin(pi/2/2))
2 * pi * R * h

# our approximation:
points <- t(cap$vb[-4, ]) # the points on the spherical cap
del <- delaunay(points, elevation = TRUE)
del[["surface"]]

# try to increase 'subdivision' in 'vcgSphericalCap' to get a
# better approximation of the true value
# note that 'Rvcg' returns the same result as ours:
vcgArea(cap)

# let's check the volume as well:
pi * h^2 * (R - h/3) # true value
del[["volume"]]

# there's a warning with 'Rvcg':
tryCatch(vcgVolume(cap), warning = function(w) message(w))
suppressWarnings({vcgVolume(cap)})

---

**Edge2** *R6 class representing an edge in dimension 2.*

**Description**

An edge is given by two vertices in the 2D space, named A and B. This is for example an edge of a Voronoi cell of a 2D Delaunay tessellation.

**Active bindings**

- A get or set the vertex A
- B get or set the vertex B

**Methods**

Public methods:

- Edge2$new()
- Edge2$print()
- Edge2$plot()
• `Edge2$stack()`
• `Edge2$clone()`

**Method new():** Create a new Edge2 object.

**Usage:**
`Edge2$new(A, B)`

**Arguments:**
A the vertex A
B the vertex B

**Returns:** A new Edge2 object.

**Examples:**
```r
dge <- Edge2$new(c(1, 1), c(2, 3))
dge
dge$A
dge$A <- c(1, 0)
dge
```

**Method print():** Show instance of an Edge2 object.

**Usage:**
`Edge2$print(...)`

**Arguments:**
... ignored

**Examples:**
```r
Edge2$new(c(2, 0), c(3, -1))
```

**Method plot():** Plot an Edge2 object.

**Usage:**
`Edge2$plot(color = "black", ...)`

**Arguments:**

color the color of the edge
... graphical parameters such as lty or lwd

**Examples:**
```r
library(tessellation)
centricSquare <- rbind(
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1), c(0, 0)
)
d <- delaunay(centricSquare)
v <- voronoi(d)
cell5 <- v[[5]] # the cell of the point (0, 0), at the center
isBoundedCell(cell5) # TRUE
plot(centricSquare, type = "n")
invisible(lapply(cell5[["cell"]], function(edge) edge$plot()))
```

**Method stack():** Stack the two vertices of the edge (this is for internal purpose).
Usage:
Edge2$stack()

Method clone(): The objects of this class are cloneable with this method.

Usage:
Edge2$clone(deep = FALSE)

Arguments:
  deep  Whether to make a deep clone.

Examples

```r
## -------------------------------------------
## Method `Edge2$new`
## -------------------------------------------

d <- Edge2$new(c(1, 1), c(2, 3))
d
d$A
  d$A <- c(1, 0)
d
## -------------------------------------------
## Method `Edge2$print`
## -------------------------------------------

d$print()

## -------------------------------------------
## Method `Edge2$plot`
## -------------------------------------------

library(tessellation)
centricSquare <- rbind(
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1), c(0, 0)
)
d <- delaunay(centricSquare)
v <- voronoi(d)
cell5 <- v[[5]]  # the cell of the point (0, 0), at the center
isBoundedCell(cell5)  # TRUE
plot(criticalSquare, type = "n")
invisible(lapply(cell5["cell"], function(edge) edge$plot()))
```

---

**Edge3**

R6 class representing an edge in dimension 3.

**Description**

An edge is given by two vertices in the 3D space, named A and B. This is for example an edge of a Voronoi cell of a 3D Delaunay tessellation.
Active bindings

A  get or set the vertex A
B  get or set the vertex B
idA get or set the id of vertex A
idB get or set the id of vertex B

Methods

Public methods:

• Edge3$new()
• Edge3$print()
• Edge3$plot()
• Edge3$stack()
• Edge3$clone()

Method new(): Create a new Edge3 object.

Usage:
Edge3$new(A, B, idA, idB)

Arguments:
A  the vertex A
B  the vertex B
idA the id of vertex A, an integer; can be missing
idB the id of vertex B, an integer; can be missing

Returns: A new Edge3 object.

Examples:
edge <- Edge3$new(c(1, 1, 1), c(1, 2, 3))
edge
edge$A
edge$A <- c(1, 0, 0)
edge

Method print(): Show instance of an Edge3 object.

Usage:
Edge3$print(...)

Arguments:
... ignored

Examples:
Edge3$new(c(2, 0, 0), c(3, -1, 4))

Method plot(): Plot an Edge3 object.

Usage:
Edge3$plot(edgeAsTube = FALSE, tubeRadius, tubeColor)
Arguments:
edgeAsTube  Boolean, whether to plot the edge as a tube
tubeRadius  the radius of the tube
tubeColor   the color of the tube

Examples:
library(tessellation)
d <- delaunay(cenricCuboctahedron())
v <- voronoi(d)
cell13 <- v[[13]]  # the point (0, 0, 0), at the center
isBoundedCell(cell13)  # TRUE
library(rgl)
open3d(windowRect = c(50, 50, 562, 562))
invisible(lapply(cell13["cell"], function(edge) edge$plot()))

Method stack(): Stack the two vertices of the edge (this is for internal purpose).
Usage:
Edge3$stack()

Method clone(): The objects of this class are cloneable with this method.
Usage:
Edge3$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Examples

## Method `Edge3$new`
## ----------------------------------
edge <- Edge3$new(c(1, 1, 1), c(1, 2, 3))
edge
edge$A
edge$A <- c(1, 0, 0)
edge

## Method `Edge3$print`
## ----------------------------------
Edge3$new(c(2, 0, 0), c(3, -1, 4))

## Method `Edge3$plot`
## ----------------------------------
library(tessellation)
getDelaunaySimplicies

```
d <- delaunay(centricCuboctahedron())
v <- voronoi(d)
cell13 <- v[[13]] # the point (0, 0, 0), at the center
isBoundedCell(cell13) # TRUE
library(rgl)
open3d(windowRect = c(50, 50, 562, 562))
invisible(lapply(cell13[["cell"]], function(edge) edge$plot()))
```

---

**getDelaunaySimplicies**  *Delaunay simplicies*

**Description**
Get Delaunay simplicies (tiles).

**Usage**

```
getDelaunaySimplicies(tessellation, hashes = FALSE)
```

**Arguments**

- `tessellation` the output of `delaunay`
- `hashes` Boolean, whether to return the simplicies as hash maps

**Value**
The list of simplicies of the Delaunay tessellation.

**Examples**

```
library(tessellation)
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)
getDelaunaySimplicies(tess)
```
Description

A semi-infinite edge is given by a vertex, its origin, and a vector, its direction. Voronoi diagrams possibly have such edges.

Active bindings

O get or set the vertex 0
direction get or set the vector direction

Methods

Public methods:

• IEdge2$new()
• IEdge2$print()
• IEdge2$clone()

Method new(): Create a new IEdge2 object.

Usage:
IEdge2$new(O, direction)

Arguments:
O the vertex 0 (origin)
direction the vector direction

Returns: A new IEdge2 object.

Examples:
iedge <- IEdge2$new(c(1, 1), c(2, 3))
iedge
iedge$O
iedge$O <- c(1, 0)
iedge

Method print(): Show instance of an IEdge2 object.

Usage:
IEdge2$print(...)

Arguments:
... ignored

Examples:
IEdge2$new(c(2, 0), c(3, -1))

Method clone(): The objects of this class are cloneable with this method.
**Usage:**

```r
IEdge2$clone(deep = FALSE)
```

**Arguments:**

- `deep` Whether to make a deep clone.

**Examples**

```r
# Method `IEdge3$new`

iedge <- IEdge3$new(c(1, 1), c(2, 3))
iedge
iedge$O
iedge$O <- c(1, 0)
iedge

# Method `IEdge3$print`

IEdge3$new(c(2, 0), c(3, -1))
```

---

**IEdge3**  
*R6 class representing a semi-infinite edge in dimension 3*

**Description**

A semi-infinite edge is given by a vertex, its origin, and a vector, its direction. Voronoi diagrams possibly have such edges.

**Active bindings**

- `O` get or set the vertex 0
- `direction` get or set the vector direction

**Methods**

**Public methods:**

- `IEdge3$new()`
- `IEdge3$print()`
- `IEdge3$clone()`

**Method new():** Create a new IEdge3 object.

**Usage:**

```r
IEdge3$new(0, direction)
```
**Arguments:**
0 the vertex 0 (origin)
direction the vector direction

**Returns:** A new IEdge3 object.

**Examples:**
iedge <- IEdge3$new(c(1, 1, 1), c(1, 2, 3))
iedge
iedge$O
iedge$O <- c(1, 0, 0)
iedge

**Method print():** Show instance of an IEdge3 object.

**Usage:**
IEdge3$print(...)

**Arguments:**
... ignored

**Examples:**
IEdge3$new(c(2, 0, 0), c(3, -1, 4))

**Method clone():** The objects of this class are cloneable with this method.

**Usage:**
IEdge3$clone(deep = FALSE)

**Arguments:**
depth Whether to make a deep clone.

**Examples**

```r
# -----------------------------
# Method 'IEdge3$new'
# -----------------------------

iedge <- IEdge3$new(c(1, 1, 1), c(1, 2, 3))
iedge
iedge$O
iedge$O <- c(1, 0, 0)
iedge

# -----------------------------
# Method 'IEdge3$print'
# -----------------------------

IEdge3$new(c(2, 0, 0), c(3, -1, 4))
```
isBoundedCell  Is this cell bounded?

Description
Check whether a Voronoi cell is bounded, i.e. contains only finite edges.

Usage
isBoundedCell(cell)

Arguments
cell a Voronoi cell

Value
A Boolean value, whether the cell is bounded.

plotBoundedCell2D  Plot a bounded Voronoi 2D cell

Description
Plot a bounded Voronoi 2D cell.

Usage
plotBoundedCell2D(
cell,  
border = "black",  
color = NA,  
check.bounded = TRUE,  
...  
)

Arguments
cell a bounded Voronoi 2D cell
border color of the borders of the cell; NA for no color
color color of the cell; NA for no color
check.bounded Boolean, whether to check that the cell is bounded; set to FALSE for a small speed gain if you know that the cell is bounded
... graphical parameters for the borders
Value
No value, this function just plots the cell (more precisely, it adds the plot of the cell to the current plot).

Examples

library(tessellation)
centricSquare <- rbind(
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1), c(0, 0)
)
d <- delaunay(centricSquare)
v <- voronoi(d)
cell5 <- v[[5]]
isBoundedCell(cell5) # TRUE
plot(centricSquare, type = "n", asp = 1, xlab = "x", ylab = "y")
plotBoundedCell2D(cell5, color = "pink")

plotBoundedCell3D

Plot a bounded Voronoï 3D cell

Description
Plot a bounded Voronoï 3D cell with rgl.

Usage

plotBoundedCell3D(
  cell,
  edgesAsTubes = FALSE,
  tubeRadius,
  tubeColor,
  facetsColor = NA,
  alpha = 1,
  check.bounded = TRUE
)

Arguments

cell
  a bounded Voronoï 3D cell
edgesAsTubes
  Boolean, whether to plot edges as tubes or as lines
tubeRadius
  radius of the tubes if edgesAsTubes = TRUE
tubeColor
  color of the tubes if edgesAsTubes = TRUE
facetsColor
  color of the facets; NA for no color
alpha
  opacity of the facets, a number between 0 and 1
check.bounded
  Boolean, whether to check that the cell is bounded; set to FALSE for a small speed gain if you know that the cell is bounded
plotDelaunay2D

Value

No value, this function just plots the cell.

Examples

```r
library(tessellation)
d <- delaunay(centricCuboctahedron())
v <- voronoi(d)
cell13 <- v[[13]]
isBoundedCell(cell13) # TRUE
library(rgl)
open3d(windowRect = c(50, 50, 562, 562))
plotBoundedCell3D(
  cell13, edgesAsTubes = TRUE, tubeRadius = 0.03, tubeColor = "yellow",
  facetsColor = "navy", alpha = 0.7
)
```

Description

Plot a 2D Delaunay tessellation.

Usage

```r
plotDelaunay2D(
  tessellation, border = "black",
  color = "distinct",
  hue = "random",
  luminosity = "light",
  lty = par("lty"),
  lwd = par("lwd"),
  ...
)
```

Arguments

tessellation the output of `delaunay`
border the color of the borders of the triangles; NULL for no borders
color controls the filling colors of the triangles, 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`
lty, lwd graphical parameters
... arguments passed to `plot`
Value

No value, just renders a 2D plot.

Examples

```r
# random points in a square
set.seed(314)
library(tessellation)
library(uniformly)
square <- rbind(
  c(-1, 1), c(1, 1), c(1, -1), c(-1, -1)
)
ptsin <- runif_in_cube(10L, d = 2L)
pts <- rbind(square, ptsin)
d <- delaunay(pts)
opar <- par(mar = c(0, 0, 0, 0))
plotDelaunay2D(
  d, xlab = NA, ylab = NA, asp = 1, color = "random", luminosity = "dark"
)
par(opar)
```

Description

Plot a 3D Delaunay tessellation with `rgl`.

Usage

```r
plotDelaunay3D(
  tessellation,
  color = "distinct",
  hue = "random",
  luminosity = "light",
  alpha = 0.3,
  exteriorEdgesAsTubes = FALSE,
  tubeRadius,
  tubeColor
)
```

Arguments

tessellation the output of `delaunay`
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`
**plotVoronoiDiagram**

alpha opacity, number between 0 and 1
exteriorEdgesAsTubes Boolean, whether to plot the exterior edges as tubes; in order to use this feature, you need to set exteriorEdges = TRUE in the delaunay function
tubeRadius if exteriorEdgesAsTubes = TRUE, the radius of the tubes
tubeColor if exteriorEdgesAsTubes = TRUE, the color of the tubes

**Value**
No value, just renders a 3D plot.

**Examples**

```r
library(tessellation)
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)
open3d(windowRect = c(50, 50, 562, 562))
plotDelaunay3D(
  tess, exteriorEdgesAsTubes = TRUE, tubeRadius = 0.3, tubeColor = "yellow"
)
```

---

**Description**
Plot all the bounded cells of a 2D or 3D Voronoi tessellation.

**Usage**

```r
plotVoronoiDiagram(
  v, 
  colors = "random", 
  hue = "random", 
  luminosity = "light", 
  alpha = 1, 
  ... 
)
```
Arguments

- **v**: an output of `voronoi`
- **colors**: this can be "random" to use random colors for the cells (with `randomColor`), "distinct" to use distinct colors with the help of `distinctColorPalette`, or this can be NA for no colors, or a vector of colors; the length of this vector of colors must match the number of bounded cells, which is displayed when you run the `voronoi` function and that you can also get by typing `attr(v, "nbounded")`
- **hue, luminosity** if colors = "random", these arguments are passed to `randomColor`
- **alpha**: opacity, a number between 0 and 1 (used when colors is not NA)
- **...**: arguments passed to `plotBoundedCell2D` or `plotBoundedCell3D`

Value

No returned value.

Note

Sometimes, it is necessary to set the option `degenerate=TRUE` in the `delaunay` function in order to get a correct Voronoï diagram with the `plotVoronoiDiagram` function (I don’t know why).

Examples

```r
library(tessellation)
# 2D example: Fermat spiral
theta <- seq(0, 100, length.out = 300L)
x <- sqrt(theta) * cos(theta)
y <- sqrt(theta) * sin(theta)
pts <- cbind(x,y)
opar <- par(mar = c(0, 0, 0, 0), bg = "black")
# Here is a Fermat spiral:
plot(pts, asp = 1, xlab = NA, ylab = NA, axes = FALSE, pch = 19, col = "white")
# And here is its Voronoï diagram:
plot(NULL, asp = 1, xlim = c(-15, 15), ylim = c(-15, 15),
    xlab = NA, ylab = NA, axes = FALSE)
del <- delaunay(pts)
v <- voronoi(del)
length(Filter(isBoundedCell, v)) # 281 bounded cells
plotVoronoiDiagram(v, colors = viridisLite::turbo(281L))
par(opar)

# 3D example: tetrahedron surrounded by three circles
tetrahedron <-
    rbind(
        c(2*sqrt(2)/3, 0, -1/3),
        c(-sqrt(2)/3, sqrt(2/3), -1/3),
        c(-sqrt(2)/3, -sqrt(2/3), -1/3),
        c(0, 0, 1)
    )
```
angles <- seq(0, 2*pi, length.out = 91)[-1]
R <- 2.5
circle1 <- t(vapply(angles, function(a) R*c(cos(a), sin(a), 0), numeric(3L)))
circle2 <- t(vapply(angles, function(a) R*c(cos(a), 0, sin(a)), numeric(3L)))
circle3 <- t(vapply(angles, function(a) R*c(0, cos(a), sin(a)), numeric(3L)))
circles <- rbind(circle1, circle2, circle3)
pts <- rbind(tetrahedron, circles)
d <- delaunay(pts, degenerate = TRUE)
v <- voronoi(d)
library(rgl)
open3d(windowRect = c(50, 50, 562, 562))
material3d(lwd = 2)
plotVoronoiDiagram(v, luminosity = "bright")

---

**surface**

Tessellation surface

**Description**

Exterior surface of the Delaunay tessellation.

**Usage**

surface(tessellation)

**Arguments**

tessellation output of **delaunay**

**Value**

A number, the exterior surface of the Delaunay tessellation (perimeter in 2D).

**Note**

It is not guaranteed that this function provides the correct result for all cases. The exterior surface of the Delaunay tessellation is the exterior surface of the convex hull of the sites (the points), and you can get it with the **cxhull** package (by summing the volumes of the facets). Moreover, I encountered some cases for which I got a correct result only with the option degenerate=TRUE in the delaunay function. I will probably remove this function in the next version.

**See Also**

**volume**
teapot  

*Utah teapot*

**Description**

Vertices of the Utah teapot.

**Usage**

`teapot()`

**Value**

A matrix with 1976 rows and 3 columns.

---

tessellation-imports  

*Objects imported from other packages*

**Description**

These objects are imported from other packages. Follow the links to their documentation: `values`, `keys`.

---

volume  

*Tessellation volume*

**Description**

The volume of the Delaunay tessellation, that is, the volume of the convex hull of the sites.

**Usage**

`volume(tessellation)`

**Arguments**

- `tessellation`  
  output of `delaunay`

**Value**

A number, the volume of the Delaunay tessellation (area in 2D).

**See Also**

`surface`
voronoi

Voronoï tessellation

Description

Voronoï tessellation from Delaunay tessellation; this is a list of pairs made of a site (a vertex) and a list of edges.

Usage

```r
voronoi(tessellation)
```

Arguments

- `tessellation`: output of `delaunay`

Value

A list of pairs representing the Voronoï tessellation. Each pair is named: the first component is called "site", and the second component is called "cell".

See Also

- `isBoundedCell`, `cellVertices`, `plotBoundedCell2D`, `plotBoundedCell3D`

Examples

```r
library(tessellation)
d <- delaunay(cubicCuboctahedron())
v <- voronoi(d)
# the Voronoï diagram has 13 cells (one for each site):
length(v)
# there is only one bounded cell:
length(Filter(isBoundedCell, v)) # or attr(v, "nbounded")
```
Index

cellVertices, 2, 23
centricCuboctahedron, 3
delaunay, 3, 11, 17–23
distinctColorPalette, 17, 18, 20
Edge2, 6
Edge3, 4, 8
getDelaunaySimplicies, 5, 11
hash, 4
IEdge2, 12
IEdge3, 13
isBoundedCell, 15, 23
keys, 22
keys (tessellation-imports), 22
mesh3d, 4
pair, 23
plot, 17
plotBoundedCell2D, 15, 20, 23
plotBoundedCell3D, 16, 20, 23
plotDelaunay2D, 5, 17
plotDelaunay3D, 4, 5, 18
plotVoronoiDiagram, 19
randomColor, 17, 18, 20
surface, 21, 22
teapot, 22
tessellation-imports, 22
values, 22
values (tessellation-imports), 22
vcgGetEdge, 4
volume, 21, 22
voronoi, 20, 23