Package: cdtr (via r-universe)

October 21, 2026

Title Constrained Delaunay Triangulation with Refinement

Version 0.1.0

Description Binding to the 'CDT' header-only C++ library: constrained and conforming Delaunay triangulation, Ruppert refinement by area or minimum angle (Ruppert, 1995, <doi:10.1006/jagm.1995.1021>), resolution of crossing constraints, and topological region depth per triangle. Aimed at replacing 'RTriangle' in 'anglr'.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

LinkingTo cpp11

Suggests bench, RTriangle, sf, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

URL https://github.com/hypertidy/cdtr, https://hypertidy.github.io/cdtr/

BugReports https://github.com/hypertidy/cdtr/issues

NeedsCompilation yes

Author Michael Sumner [aut, cre], Leica Geosystems Technology AB [cph] (Copyright holder of the bundled 'CDT' C++ library), CDT Contributors [ctb, cph] ('CDT' C++ library, https://github.com/artem-ogre/CDT), William C. Lenthe [ctb, cph] (Robust geometric predicates (predicates.h))

Maintainer Michael Sumner <mdsumner@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-21 15:20:09 UTC

RemoteUrl https://github.com/cran/cdtr

RemoteRef HEAD

RemoteSha 1a8c62e52636e9d600adf4d28382d60a37ad2792

Contents

cdt_edges 2
cdt_interpolate 3
cdt_pslg 3
cdt_triangulate 4
cdt_triangulate_attr 6
default_min_edge_length 7

Index **9**

`cdt_edges` *Unique edges of a triangulation*

Description

The index pairs of every triangle edge once, for plotting with `graphics::segments()` or for building an edge table.

Usage

```
cdt_edges(x)
```

Arguments

x	result of `cdt_triangulate()`

Value

integer matrix (2 columns) of 1-based vertex indices

Examples

```
r <- cdt_triangulate(c(0, 1, 1, 0, 0.5), c(0, 0, 1, 1, 0.5))
e <- cdt_edges(r)
plot(r$P, asp = 1)
segments(r$P[e[, 1], 1], r$P[e[, 1], 2], r$P[e[, 2], 1], r$P[e[, 2], 2])
```

cdt_interpolate *Interpolate per-vertex attributes onto points*

Description

Linear (barycentric) interpolation of a matrix of per-vertex values from a triangle mesh onto query points. Points in no triangle take the nearest vertex's value. Used to carry attributes such as z onto Steiner vertices after refinement, matching the `PA` behaviour of RTriangle.

Usage

```
cdt_interpolate(P, T, A, xq, yq)
```

Arguments

`P`	vertex coordinates (n x 2)
`T`	triangle vertex indices (m x 3, 1-based)
`A`	per-vertex attribute matrix (n x k) or vector
`xq, yq`	query coordinates

Value

a length(xq) x k matrix (column names kept from `A`)

Examples

```
## two triangles on the unit square, z = x + y at the corners
P <- cbind(c(0, 1, 1, 0), c(0, 0, 1, 1))
T <- rbind(c(1, 2, 3), c(1, 3, 4))
A <- cbind(z = P[, 1] + P[, 2])
## a linear field is reproduced exactly inside; (2, 2) is outside
cdt_interpolate(P, T, A, xq = c(0.5, 0.25, 2), yq = c(0.5, 0.1, 2))
```

cdt_pslg *Triangulate an RTriangle pslg with the CDT backend*

Description

Convenience for input in the shape of `RTriangle::pslg()`: a list with `P`, a two-column matrix of vertex coordinates, and `S`, a two-column matrix of 1-based segment vertex indices.

Usage

```
cdt_pslg(p, ...)
```

Arguments

p	a pslg (list with P and S)
...	passed to cdt_triangulate()

Value

the list returned by cdt_triangulate()

Examples

```
p <- list(P = cbind(c(0, 2, 2, 1, 0), c(0, 0, 2, 1, 2)),
          S = cbind(1:5, c(2:5, 1)))
r <- cdt_pslg(p, max_area = 0.05)
nrow(r$T)
plot(r$P, asp = 1, pch = ".")
polygon(r$P[t(cbind(r$T, NA)), ])
```

---

cdt_triangulate *Constrained Delaunay triangulation with refinement (CDT backend)*

Description

Thin binding to the artem-ogre/CDT library. Input mirrors RTriangle::pslg: vertices as x/y and constraint segments as index pairs into the vertices.

Usage

```
cdt_triangulate(
  x,
  y,
  s0 = NULL,
  s1 = NULL,
  max_area = NULL,
  min_angle = NULL,
  max_steiner = Inf,
  min_edge_length = NULL,
  min_edge_frac = 0.3,
  area_edge_frac = 0.25,
  conforming = FALSE,
  erase = c("outer", "hull", "holes"),
  intersect = c("resolve", "error", "ignore"),
  angle_first = FALSE,
  depth_distinct = FALSE
)
```

Arguments

`x, y`	vertex coordinates
`s0, s1`	1-based vertex indices of constraint segment start/end (may be NULL)
`max_area`	maximum triangle area (Ruppert refinement by area), NULL for none
`min_angle`	minimum triangle angle in degrees, NULL for none
`max_steiner`	budget of Steiner points to insert (Inf for unlimited)
`min_edge_length`	refinement does not split edges or triangles already shorter than this. NULL (the default) chooses a value from the input and the targets (see `default_min_edge_length()`): a fraction of the median constraint segment length, capped so that it never blocks the `max_area` target. This stops the refiner cascading into sharp input corners. Use 0 to never give up.
`min_edge_frac`	fraction of the median constraint segment length used when `min_edge_length` is NULL
`area_edge_frac`	fraction of `sqrt(max_area)` that caps the default floor
`conforming`	if TRUE use conforming (Steiner on segments) instead of constrained
`erase`	one of "outer" (drop triangles outside the segment-bounded region, as RTriangle's `-p` does), "hull" (keep the convex hull), "holes" (also drop even-depth regions). With no segments there is no boundary, so "hull" is used.
`intersect`	how crossing constraints are handled: "resolve" inserts the crossing point, "error" fails, "ignore" skips the check
`angle_first`	when both criteria are given, refine by angle before area (default is area then angle; the two orderings differ little)
`depth_distinct`	also return `depth_distinct`, a per-triangle depth that counts a doubled (overlapping) boundary once rather than once per copy. The default `depth` counts per copy; the two differ only where the only path from outside a triangle crosses a shared, doubled boundary.

Value

list with P (vertices), T (1-based triangle indices, wound counter-clockwise / positive signed area), S (fixed edges), depth (per-triangle constraint layer depth: 0 outside, odd inside, even hole; a doubled shared boundary is counted once per copy), origin (a factor giving each vertex's provenance: "input", "crossing" or "steiner"), n_input (vertices after dedupe, before Steiner), n_crossing (vertices added resolving crossing constraints), input_map (input vertex -> deduped vertex index), min_edge_length (the value used), and unrefined, a data frame with one row per refinement pass counting what CDT could not refine (short edges, sharp fixed corners, and so on). When `depth_distinct = TRUE` the list also has depth_distinct (a doubled boundary counted once).

Examples

```
## unit square with a square hole: two closed rings of 4 vertices
x <- c(0, 1, 1, 0, 0.25, 0.75, 0.75, 0.25)
y <- c(0, 0, 1, 1, 0.25, 0.25, 0.75, 0.75)
s0 <- c(1, 2, 3, 4, 5, 6, 7, 8)
```

```
s1 <- c(2, 3, 4, 1, 6, 7, 8, 5)

## constrained triangulation, hole triangles kept with even depth
r <- cdt_triangulate(x, y, s0, s1)
table(r$depth)

## drop the hole and refine by area and minimum angle
r2 <- cdt_triangulate(x, y, s0, s1, erase = "holes",
                      max_area = 0.005, min_angle = 25)
table(r2$origin)
plot(r2$P, asp = 1, pch = ".")
polygon(r2$P[t(cbind(r2$T, NA)), ], col = "grey90")

## points only: Delaunay triangulation of the convex hull
set.seed(1)
r3 <- cdt_triangulate(runif(30), runif(30))
nrow(r3$T)
```

`cdt_triangulate_attr` *Triangulate with per-vertex attributes carried onto new vertices*

Description

`cdt_triangulate()` plus a `PA` matrix of per-input-vertex attributes (z, m, t ...), returned as `$PA` aligned with the output vertices. Input vertices keep their values; Steiner vertices get values interpolated from the unrefined constrained triangulation of the input.

Usage

```
cdt_triangulate_attr(x, y, s0 = NULL, s1 = NULL, PA = NULL, ...)
```

Arguments

`x, y`	vertex coordinates
`s0, s1`	1-based vertex indices of constraint segment start/end (may be NULL)
`PA`	numeric matrix with one row per input vertex (or NULL)
`...`	passed to `cdt_triangulate()`

Value

the list returned by `cdt_triangulate()`, plus `PA`, a matrix with one row per output vertex and the columns of the input `PA` (absent when `PA = NULL`)

Examples

```
x <- c(0, 1, 1, 0)
y <- c(0, 0, 1, 1)
## a sloping surface: z rises with x
r <- cdt_triangulate_attr(x, y, 1:4, c(2:4, 1),
                          PA = cbind(z = 10 * x), max_area = 0.02)
## Steiner vertices carry interpolated z
head(cbind(r$P, r$PA)[r$origin == "steiner", , drop = FALSE])
all.equal(r$PA[, "z"], 10 * r$P[, 1])
```

`default_min_edge_length`

Default refinement edge-length floor

Description

`min_edge_frac` times the median constraint segment length, which keeps the refiner from cascading into sharp input corners, capped at `area_edge_frac * sqrt(max_area)` so the floor never blocks the area target (a triangle of area A has edges of order sqrt(A)). With no segments only the area cap applies; with neither, 0.

Usage

```
default_min_edge_length(
  x,
  y,
  s0,
  s1,
  max_area = NULL,
  min_edge_frac = 0.3,
  area_edge_frac = 0.25
)
```

Arguments

`x, y`	vertex coordinates
`s0, s1`	1-based vertex indices of constraint segment start/end (may be NULL)
`max_area`	maximum triangle area (Ruppert refinement by area), NULL for none
`min_edge_frac`	fraction of the median constraint segment length used when `min_edge_length` is NULL
`area_edge_frac`	fraction of `sqrt(max_area)` that caps the default floor

Value

a single non-negative number, the edge-length floor that `cdt_triangulate()` uses when `min_edge_length = NULL`

Examples

```
x <- c(0, 1, 1, 0)
y <- c(0, 0, 1, 1)
s0 <- 1:4
s1 <- c(2:4, 1)
## 0.3 times the median segment length
default_min_edge_length(x, y, s0, s1)
## a small area target caps the floor at 0.25 * sqrt(max_area)
default_min_edge_length(x, y, s0, s1, max_area = 0.01)
## no segments and no area target: no floor
default_min_edge_length(x, y, integer(0), integer(0))
```

Index

cdt_edges, 2
cdt_interpolate, 3
cdt_pslg, 3
cdt_triangulate, 4
cdt_triangulate(), *2*, *4*, *6*, *7*
cdt_triangulate_attr, 6

default_min_edge_length, 7
default_min_edge_length(), *5*

graphics::segments(), *2*