Package 'tulpaMesh'

November 2, 2026

Title Constrained Delaunay Triangulation Meshes for Spatial 'SPDE' Models

Version 0.1.4

Description Generate constrained Delaunay triangulation meshes for use with stochastic partial differential equation (SPDE) spatial models (Lindgren, Rue and Lindstroem 2011 <doi:10.1111/j.1467-9868.2011.00777.x>). Provides automatic mesh generation from point coordinates with boundary constraints, Ruppert refinement for mesh quality, finite element method (FEM) matrix assembly (mass, stiffness, projection), barrier models, spherical meshes via icosahedral subdivision, and metric graph meshes for network geometries. Built on the 'CDT' header-only C++ library (Amirkhanov 2024 <https://github.com/artem-ogre/CDT>). Designed as the mesh backend for the 'tulpa' Bayesian hierarchical modelling engine but usable standalone for any spatial triangulation task.

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements C++17, GNU make

Depends R (>= 4.1.0)

Imports Rcpp (>= 1.0.12), Matrix (>= 1.5-0), RcppParallel

Suggests testthat (>= 3.0.0), sf, fmesher, ggplot2, svglite, knitr, rmarkdown

VignetteBuilder knitr

LinkingTo Rcpp, RcppParallel

Config/testthat/edition 3

URL https://gillescolling.com/tulpaMesh/, https://github.com/gcol33/tulpaMesh

BugReports https://github.com/gcol33/tulpaMesh/issues

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Gilles Colling [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-3070-6066>),
Artem Amirkhanov [ctb, cph] (CDT library (src/cdt/, MPL-2.0)),
William C. Lenthe [ctb, cph] (Geometric predicates
(src/cdt/predicates.h, BSD-3-Clause))

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-11-02 03:50:02 UTC

Contents

as_tulpa_mesh 2
barrier_triangles 3
fem_matrices 4
fem_matrices_nonstationary 5
fem_matrices_p2 5
mesh_components 6
mesh_crs 7
mesh_quality 7
mesh_summary 8
plot.tulpa_mesh 9
refine_mesh 10
subdivide_mesh 11
subset_mesh 11
tulpa_mesh 12
tulpa_mesh_1d 13
tulpa_mesh_graph 14
tulpa_mesh_sphere 15

Index **17**

`as_tulpa_mesh` *Convert to a tulpa_mesh Object*

Description

Generic function to convert mesh objects from other packages into `tulpa_mesh` objects. Currently supports `fm_mesh_2d` objects from the fmesher package.

Usage

```
as_tulpa_mesh(x, ...)

## S3 method for class 'fm_mesh_2d'
as_tulpa_mesh(x, ...)
```

```
## S3 method for class 'inla.mesh'
as_tulpa_mesh(x, ...)
```

Arguments

x	Object to convert.
...	Additional arguments (currently unused).

Value

A tulpa_mesh object.

---

barrier_triangles *Identify Barrier Triangles*

Description

Determines which mesh triangles fall inside barrier regions (e.g., coastlines, rivers, lakes). A triangle is marked as a barrier if its centroid falls inside any barrier polygon.

Usage

```
barrier_triangles(mesh, barriers)
```

Arguments

mesh	A tulpa_mesh object.
barriers	An sf/sfc object defining barrier regions, or a list of N x 2 coordinate matrices (each defining a closed polygon).

Value

A logical vector of length n_triangles. TRUE indicates the triangle is inside a barrier region.

`fem_matrices` *Compute FEM Matrices from a Mesh*

Description

Computes the finite element mass (C) and stiffness (G) matrices from a triangular mesh, plus the projection matrix (A) that maps mesh vertices to observation locations.

Usage

```
fem_matrices(
  mesh,
  obs_coords = NULL,
  barrier = NULL,
  parallel = FALSE,
  lumped = FALSE
)
```

Arguments

`mesh`	A `tulpa_mesh` object.
`obs_coords`	Observation coordinates (N x 2 matrix). If NULL, the projection matrix A is the identity (observations at mesh nodes).
`barrier`	Optional logical vector of length `n_triangles`. Triangles marked `TRUE` are treated as physical barriers (coastlines, rivers): their stiffness contributions are zeroed so the spatial field cannot smooth across them. Based on Bakka et al. (2019).
`parallel`	Logical. If `TRUE`, uses parallel FEM assembly via RcppParallel (chunk-local triplet accumulation merged under a mutex). Beneficial for meshes with >50K triangles. Default `FALSE`.
`lumped`	Logical. If `TRUE`, returns a diagonal lumped mass matrix C0 (vertex areas) in addition to the consistent mass matrix C. The lumped mass inverse is trivial and needed for the SPDE Q-builder. Default `FALSE`.

Value

A list with sparse matrices (dgCMatrix class from Matrix package):

- `C`: consistent mass matrix (n_vertices x n_vertices)
- `G`: stiffness matrix (n_vertices x n_vertices)
- `A`: projection matrix (n_obs x n_vertices)
- `n_mesh`: number of mesh vertices
- `C0`: (only if `lumped = TRUE`) diagonal lumped mass matrix
- `va`: (only if `lumped = TRUE`) vertex areas (numeric vector)
- `ta`: (only if `lumped = TRUE`) triangle areas (numeric vector)

fem_matrices_nonstationary *Non-stationary FEM Matrices with Spatially Varying Parameters*

Description

Computes weighted FEM matrices for non-stationary SPDE models where the range and variance parameters vary spatially. The weights are per-vertex kappa(s) and tau(s) values, interpolated within each triangle using the element-average approximation.

Usage

```
fem_matrices_nonstationary(mesh, kappa, tau)
```

Arguments

mesh	A tulpa_mesh object (2D only).
kappa	Numeric vector of length n_vertices giving the spatial scale parameter kappa(s) = sqrt(8*nu) / range(s).
tau	Numeric vector of length n_vertices giving the precision scaling tau(s).

Value

A list with sparse matrices:

- Ck: kappa²-weighted mass matrix
- Gk: kappa²-weighted stiffness matrix
- Ct: tau²-weighted mass matrix
- C: unweighted consistent mass matrix
- G: unweighted stiffness matrix
- C0: lumped (diagonal) mass matrix
- n_mesh: number of mesh vertices

fem_matrices_p2 *Compute P2 (Quadratic) FEM Matrices*

Description

Computes finite element matrices using 6-node quadratic triangular elements. Each triangle edge gets a midpoint node, giving 6 basis functions per triangle instead of 3. Quadratic elements provide better approximation accuracy with fewer mesh nodes.

Usage

```
fem_matrices_p2(mesh)
```

Arguments

mesh	A tulpa_mesh object (2D only).

Value

A list with:

- C: consistent mass matrix (n_total x n_total sparse)
- G: stiffness matrix (n_total x n_total sparse)
- n_mesh: total number of nodes (vertices + midpoints)
- n_vertices: number of original mesh vertices
- n_midpoints: number of added midpoint nodes
- vertices: N x 2 matrix of all node coordinates
- triangles6: M x 6 connectivity matrix (columns: v0, v1, v2, mid01, mid12, mid20)

---

mesh_components *Identify Disconnected Mesh Components*

Description

Finds connected components of a mesh via triangle adjacency (two triangles are connected if they share an edge).

Usage

```
mesh_components(mesh)
```

Arguments

mesh	A tulpa_mesh object.

Value

An integer vector of length n_triangles giving the component ID (1, 2, ...) for each triangle.

mesh_crs *Get or Set the CRS of a Mesh*

Description

Access or assign a coordinate reference system to a `tulpa_mesh` or `tulpa_mesh_graph` object. The CRS is stored as metadata and propagated through mesh operations.

Usage

```
mesh_crs(x)

set_crs(x, value)
```

Arguments

`x`	A `tulpa_mesh`, `tulpa_mesh_graph`, or `tulpa_mesh_1d` object.
`value`	A CRS specification: an integer EPSG code, a PROJ string, a WKT string, an `sf::st_crs()` object, or `NULL` to remove.

Value

`mesh_crs()` returns the CRS (an `sf::crs` object or `NULL`). `set_crs()` returns the mesh with CRS attached.

mesh_quality *Per-Triangle Mesh Quality Metrics*

Description

Computes quality metrics for each triangle in a mesh: minimum angle, maximum angle, aspect ratio, and area.

Computes per-triangle quality metrics: minimum angle, maximum angle, aspect ratio, and area. Useful for identifying degenerate triangles that may cause numerical issues in SPDE models.

Usage

```
mesh_quality(mesh)

mesh_quality(mesh)
```

Arguments

`mesh`	A `tulpa_mesh` object.

Value

A data.frame with one row per triangle and columns:

- `min_angle`: minimum interior angle (degrees)
- `max_angle`: maximum interior angle (degrees)
- `aspect_ratio`: ratio of circumradius to twice the inradius (1 for equilateral, larger for worse quality)
- `area`: triangle area

A data.frame with one row per triangle and columns:

- `min_angle`: minimum interior angle (degrees)
- `max_angle`: maximum interior angle (degrees)
- `aspect_ratio`: longest edge / shortest edge (1 = equilateral)
- `area`: triangle area

Examples

```
set.seed(42)
mesh <- tulpa_mesh(cbind(runif(50), runif(50)))
q <- mesh_quality(mesh)
summary(q)
```

`mesh_summary` *Print Mesh Quality Summary*

Description

Prints min/median/max of mesh quality metrics.

Prints a summary of mesh quality metrics.

Usage

```
mesh_summary(mesh)

mesh_summary(mesh)
```

Arguments

`mesh`	A `tulpa_mesh` object.

Value

Invisible data.frame of per-triangle quality metrics.

Invisibly returns the quality data.frame.

`plot.tulpa_mesh` *Plot a Triangular Mesh*

Description

Draws the mesh using base R graphics: edges as line segments, vertices as points. Optionally colors triangles by a quality metric.

Usage

```
## S3 method for class 'tulpa_mesh'
plot(
  x,
  color = NULL,
  border = "grey50",
  vertex_col = NULL,
  vertex_cex = 0.5,
  palette = grDevices::hcl.colors,
  n_colors = 100L,
  main = "tulpa_mesh",
  ...
)
```

Arguments

`x`	A `tulpa_mesh` object.
`color`	Optional per-triangle numeric vector to color triangles (e.g., output of `mesh_quality()$min_angle`). If `"quality"`, uses minimum angle. If `NULL`, draws edges only.
`border`	Edge color. Default `"grey50"`.
`vertex_col`	Vertex point color. Default `NULL` (no vertices drawn).
`vertex_cex`	Vertex point size. Default 0.5.
`palette`	Color palette function for triangle fill. Default `grDevices::hcl.colors`.
`n_colors`	Number of colors in palette. Default 100.
`main`	Plot title.
`...`	Additional arguments passed to `plot.default()`.

Value

The `tulpa_mesh` object `x`, returned invisibly. Called for the side effect of producing a plot.

refine_mesh *Adaptively Refine a Mesh Based on Error Indicators*

Description

Refines triangles where error indicators exceed a threshold by inserting their centroids as new vertices and re-triangulating. Designed for iterative solve-refine-re-solve workflows where error indicators come from an SPDE solver (e.g., tulpa's posterior variance).

Usage

```
refine_mesh(
  mesh,
  indicators,
  threshold = NULL,
  fraction = NULL,
  max_iter = 1L,
  min_area = 0
)
```

Arguments

mesh	A tulpa_mesh object (2D only).
indicators	Numeric vector of per-triangle error indicators (length n_triangles). Higher values trigger refinement.
threshold	Triangles with indicator above this value are refined. Default: median(indicators) (refine worst half).
fraction	Alternative to threshold: refine this fraction of triangles with the highest indicators. Default NULL (use threshold).
max_iter	Maximum number of refine-retriangulate iterations. Default 1 (single refinement pass).
min_area	Minimum triangle area below which triangles are never refined, regardless of indicator. Default 0 (no limit).

Value

A new tulpa_mesh object with refined triangulation.

subdivide_mesh *Subdivide a Mesh*

Description

Splits each triangle into 4 sub-triangles by inserting edge midpoints, then re-triangulates. Useful for multi-resolution workflows where a coarser mesh needs uniform refinement.

Usage

```
subdivide_mesh(mesh)
```

Arguments

mesh	A tulpa_mesh object (2D only).

Value

A new tulpa_mesh object with approximately 4x as many triangles.

subset_mesh *Extract a Submesh from Triangle Indices*

Description

Creates a new tulpa_mesh containing only the specified triangles, with vertex indices remapped.

Usage

```
subset_mesh(mesh, triangles)
```

Arguments

mesh	A tulpa_mesh object.
triangles	Integer vector of triangle indices to keep, or a logical vector of length n_triangles.

Value

A new tulpa_mesh object containing only the selected triangles and their vertices.

tulpa_mesh *Create a Triangular Mesh for SPDE Spatial Models*

Description

Generates a constrained Delaunay triangulation from point coordinates, optionally with boundary constraints. The resulting mesh can be used directly with tulpa's SPDE spatial fields.

Usage

```
tulpa_mesh(
  coords,
  data = NULL,
  boundary = NULL,
  max_edge = NULL,
  cutoff = 0,
  extend = 0.1,
  min_angle = NULL,
  max_area = NULL,
  max_steiner = 10000L
)

## S3 method for class 'tulpa_mesh'
print(x, ...)
```

Arguments

coords	A matrix or data.frame with columns x and y, or a formula like ~ x + y evaluated in data.
data	Optional data.frame for formula evaluation.
boundary	Optional boundary specification: a matrix of boundary vertex coordinates (N x 2), an sf polygon, or NULL for convex hull.
max_edge	Maximum edge length. A single value or a vector of two values c(inner, outer) where inner sets the lattice spacing inside the study area (the boundary, or the convex hull of coords) and outer sets the lattice spacing and boundary segment length in the extension region. A single value is used for both.
cutoff	Minimum distance between mesh vertices. Points closer than this are merged. Default 0 (no merging).
extend	Width of the extension region around the convex hull of coords, as a fraction of the domain diameter (the largest distance between two points of coords). Default 0.1 (10% of domain diameter). The extended boundary is the outward buffer of the convex hull, with rounded corners. Set to 0 for no extension. Ignored when boundary is supplied.

min_angle	Minimum angle (degrees) for Ruppert refinement. If specified, Steiner points are inserted at circumcenters of triangles with angles below this threshold. Theoretical maximum is ~20.7 degrees; values up to 30 usually work. Default NULL (no refinement).
max_area	Maximum triangle area for refinement. Triangles larger than this are refined regardless of angle quality. Default NULL (no area constraint).
max_steiner	Maximum number of Steiner points to insert during Ruppert refinement. Default 10000.
x	A tulpa_mesh object.
...	Additional arguments (ignored).

Value

A tulpa_mesh object with components:

- vertices: N x 2 matrix of vertex coordinates. Every vertex belongs to at least one triangle.
- triangles: M x 3 integer matrix of vertex indices (1-based)
- edges: K x 2 integer matrix of edge vertex indices (1-based)
- n_vertices: number of mesh vertices
- n_triangles: number of triangles
- n_edges: number of edges
- n_input_points: number of original input points

The tulpa_mesh object x, returned invisibly.

Examples

```
# Simple mesh from random points
set.seed(42)
coords <- cbind(runif(50), runif(50))
mesh <- tulpa_mesh(coords)
print(mesh)

# Mesh with Ruppert refinement (min angle 25 degrees)
mesh_refined <- tulpa_mesh(coords, min_angle = 25)
```

---

tulpa_mesh_1d *Create a 1D Mesh for Temporal SPDE Models*

Description

Generates a 1D mesh (interval partition) for temporal components of spatio-temporal SPDE models. Returns 1D FEM matrices (tridiagonal mass and stiffness) that can be combined with 2D spatial FEM matrices via Kronecker products in tulpa's space-time Q-builder.

Usage

```
tulpa_mesh_1d(knots, boundary = NULL, n_extend = 3L)

## S3 method for class 'tulpa_mesh_1d'
print(x, ...)
```

Arguments

knots	Numeric vector of mesh knot locations (e.g., time points). Will be sorted and deduplicated.
boundary	Two-element numeric vector c(lower, upper) defining the mesh domain. Defaults to range(knots).
n_extend	Number of extra knots to add beyond each boundary, spaced at the median knot interval. Default 3.
x	A tulpa_mesh_1d object.
...	Additional arguments (ignored).

Value

A tulpa_mesh_1d object with components:

- knots: sorted numeric vector of mesh knot locations
- n: number of knots
- C: consistent mass matrix (tridiagonal, n x n sparse)
- G: stiffness matrix (tridiagonal, n x n sparse)
- C0: lumped (diagonal) mass matrix

The tulpa_mesh_1d object x, returned invisibly.

---

tulpa_mesh_graph *Create a Metric Graph Mesh*

Description

Builds a 1D FEM mesh along the edges of a spatial network (roads, rivers, coastlines). Each edge is discretized into segments with P1 linear elements. Junction nodes where edges meet are shared. Based on the Whittle-Matern SPDE formulation on metric graphs (Bolin, Simas & Wallin, 2024).

Usage

```
tulpa_mesh_graph(edges, max_edge = NULL, snap_tolerance = 1e-08)

## S3 method for class 'tulpa_mesh_graph'
print(x, ...)
```

Arguments

edges	A list of N x 2 coordinate matrices, each defining a polyline edge of the network. Or an sf object with LINESTRING geometries.
max_edge	Maximum segment length along edges. Edges longer than this are subdivided. Default NULL (use vertices as-is).
snap_tolerance	Distance below which endpoints are snapped to the same junction node. Default 1e-8.
x	A tulpa_mesh_graph object.
...	Additional arguments (ignored).

Value

A tulpa_mesh_graph object with components:

- vertices: N x 2 matrix of node coordinates
- segments: M x 2 integer matrix of segment connectivity (1-based)
- n_vertices, n_segments: counts
- C: consistent mass matrix (tridiagonal-block sparse)
- G: stiffness matrix (tridiagonal-block sparse)
- C0: lumped (diagonal) mass matrix
- degree: integer vector of node degrees (junction = degree > 2)

The tulpa_mesh_graph object x, returned invisibly.

---

tulpa_mesh_sphere *Create a Triangular Mesh on the Sphere*

Description

Generates a geodesic mesh on the unit sphere by recursive subdivision of an icosahedron. Optionally refines around data locations using stereographic projection and local Delaunay triangulation.

Usage

```
tulpa_mesh_sphere(subdivisions = 3L, coords = NULL, radius = 1)
```

Arguments

subdivisions	Number of recursive icosahedral subdivisions. Each level quadruples the triangle count: 0 = 20 triangles, 1 = 80, 2 = 320, 3 = 1280, 4 = 5120, 5 = 20480. Default 3.
coords	Optional matrix of lon/lat coordinates (degrees) to insert into the mesh. Points are projected to the sphere and added as additional vertices via local re-triangulation.
radius	Sphere radius. Default 1 (unit sphere). For Earth, use 6371 (km).

Value

A `tulpa_mesh` object with components:

- `vertices`: N x 3 matrix of (x, y, z) Cartesian coordinates on the sphere surface
- `triangles`: M x 3 integer matrix of vertex indices (1-based)
- `edges`: K x 2 integer matrix of edge vertex indices (1-based)
- `lonlat`: N x 2 matrix of (longitude, latitude) in degrees
- `n_vertices`, `n_triangles`, `n_edges`: counts
- `n_input_points`: number of original input points
- `sphere`: list with `radius` and `subdivisions`

Index

as_tulpa_mesh, 2

barrier_triangles, 3

fem_matrices, 4
fem_matrices_nonstationary, 5
fem_matrices_p2, 5

mesh_components, 6
mesh_crs, 7
mesh_quality, 7
mesh_summary, 8

plot.default(), *9*
plot.tulpa_mesh, 9
print.tulpa_mesh *(*tulpa_mesh*)*, 12
print.tulpa_mesh_1d *(*tulpa_mesh_1d*)*, 13
print.tulpa_mesh_graph
*(*tulpa_mesh_graph*)*, 14

refine_mesh, 10

set_crs *(*mesh_crs*)*, 7
subdivide_mesh, 11
subset_mesh, 11

tulpa_mesh, 12
tulpa_mesh_1d, 13
tulpa_mesh_graph, 14
tulpa_mesh_sphere, 15