Package 'weightedVoronoi'

April 8, 2026

Title Weighted Spatial Tessellations in Constrained Polygon Domains

Version 1.1.1

Depends R (>= 4.1.0)

Description Provides tools for weighted spatial tessellation using Euclidean and geodesic distances within constrained polygonal domains. The package can generate complete and connected spatial partitions that respect complex boundaries, heterogeneous point weights, and optional resistance or terrain effects. The methods extend weighted Voronoi tessellations to constrained domains and graph-based cost-distance surfaces. For background see Aurenhammer (1991) <doi:10.1145/116873.116880> and van Etten (2017) <doi:10.18637/jss.v076.i13>.

URL https://HarriRaven.github.io/weightedVoronoi/, https://github.com/HarriRaven/weightedVoronoi

BugReports https://github.com/HarriRaven/weightedVoronoi/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Imports methods, sf, stats, terra, dplyr, gdistance, raster, Matrix

NeedsCompilation no

Author Harri Ravenscroft [aut, cre]

Maintainer Harri Ravenscroft <harri.ravenscroft@hotmail.co.uk>

Repository CRAN

Date/Publication 2026-04-08 14:20:03 UTC

Contents

`add_barriers` *Add barriers to a resistance surface*

Description

Modifies a resistance raster by applying semi-permeable or impermeable barriers provided as a raster mask (values > 0 treated as barrier) or as vector features (sf / SpatVector) rasterised onto the resistance grid.

Usage

```
add_barriers(
  resistance,
  barriers,
  permeability = c("semi", "impermeable", "permeable"),
  cost_multiplier = 10,
  width = 0
)
```

Arguments

`resistance`	terra::SpatRaster of strictly positive movement resistance.
`barriers`	A terra::SpatRaster mask (values > 0 treated as barrier), or an sf/SpatVector LINESTRING/POLYGON object.
`permeability`	One of "semi", "impermeable", "permeable".
`cost_multiplier`	Numeric > 0. Multiplier applied where barrier present (for "semi" and "permeable").
`width`	Buffer distance (CRS units) applied to vector barriers before rasterising.

Value

A terra::SpatRaster resistance surface with barrier effects applied.

Examples

```
r <- terra::rast(nrows = 5, ncols = 5, xmin = 0, xmax = 5, ymin = 0, ymax = 5)
terra::values(r) <- 1

b <- r
terra::values(b) <- 0
terra::values(b)[c(8, 13, 18)] <- 1

semi <- add_barriers(r, b, permeability = "semi", cost_multiplier = 10)
imp  <- add_barriers(r, b, permeability = "impermeable")

terra::global(semi, "max", na.rm = TRUE)
any(is.infinite(terra::values(imp)))
```

---

compose_resistance *Compose a resistance surface from multiple raster layers*

Description

Aligns one or more `terra::SpatRaster` layers to a common template (CRS, resolution, extent) and combines them into a single resistance raster using a specified rule. Intended for building `resistance_rast` inputs for geodesic tessellations.

Usage

```
compose_resistance(
  ...,
  template = NULL,
  mask = NULL,
  method = c("multiply", "add", "max"),
  resample_method = c("bilinear", "near"),
  na_policy = c("propagate", "ignore")
)
```

Arguments

`...`	One or more `terra::SpatRaster` layers. All layers must represent strictly positive resistance values.
`template`	Optional `terra::SpatRaster` defining the target grid (CRS, resolution, extent). Defaults to the first layer.
`mask`	Optional mask applied after alignment. Can be a `terra::SpatRaster`, `terra::SpatVector`, or `sf` polygon.
`method`	How to combine layers: `"multiply"` (default), `"add"`, or `"max"`.
`resample_method`	Resampling method used when aligning layers: `"bilinear"` for continuous resistance, `"near"` for categorical rasters.
`na_policy`	How to treat NA values: `"propagate"` makes any NA propagate to the output; `"ignore"` drops NAs (neutral element for the chosen method).

Value

A `terra::SpatRaster` resistance surface on the template grid.

Examples

```
r1 <- terra::rast(nrows = 5, ncols = 5, xmin = 0, xmax = 5, ymin = 0, ymax = 5)
terra::values(r1) <- 1

r2 <- r1
terra::values(r2) <- seq(1, 25) / 25 + 0.5

R_mult <- compose_resistance(r1, r2, method = "multiply")
R_add  <- compose_resistance(r1, r2, method = "add")

terra::global(R_mult, "min", na.rm = TRUE)
terra::global(R_add, "min", na.rm = TRUE)


boundary_sf <- sf::st_sf(
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(
      c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
    )))
  ),
  crs = 32636
)

pts <- sf::st_sf(
  w = c(2, 5),
  geometry = sf::st_sfc(
    sf::st_point(c(20, 20)),
    sf::st_point(c(80, 80))
  ),
  crs = 32636
)

template <- terra::rast(xmin = 0, xmax = 100, ymin = 0, ymax = 100,
                        resolution = 20, crs = "EPSG:32636")
terra::values(template) <- 1

R <- compose_resistance(template, template * 2, method = "multiply")
out <- weighted_voronoi_domain(
  points_sf = pts,
  weight_col = "w",
  boundary_sf = boundary_sf,
  distance = "geodesic",
  resistance_rast = R,
  res = 20,
  verbose = FALSE
)
```

prepare_geodesic_context *Prepare a geodesic context for repeated runs*

Description

Precomputes the domain mask, aligned resistance handling, transition object, and multisource graph representation (when applicable) for repeated geodesic tessellation workflows.

Usage

```
prepare_geodesic_context(
  boundary_sf,
  res = 20,
  close_mask = TRUE,
  close_iters = 1,
  resistance_rast = NULL,
  dem_rast = NULL,
  use_tobler = TRUE,
  tobler_v0_kmh = 6,
  tobler_a = 3.5,
  tobler_b = 0.05,
  min_speed_kmh = 0.25,
  anisotropy = c("none", "terrain"),
  uphill_factor = 1,
  downhill_factor = 1,
  geodesic_engine = c("classic", "multisource")
)
```

Arguments

boundary_sf	An sf POLYGON/MULTIPOLYGON defining the domain.
res	Numeric. Raster resolution in CRS units (e.g. metres).
close_mask	Logical. If TRUE, applies a morphological closing to the raster mask.
close_iters	Integer. Number of closing iterations.
resistance_rast	Optional SpatRaster giving movement resistance (>0).
dem_rast	Optional SpatRaster providing elevation or resistance surface.
use_tobler	Logical; if TRUE, apply Tobler's hiking function to convert slope into isotropic movement cost.
tobler_v0_kmh	Base walking speed on flat terrain (km/h).
tobler_a	Tobler exponential slope coefficient.
tobler_b	Tobler slope multiplier.
min_speed_kmh	Minimum allowed speed to avoid infinite costs.

anisotropy Character. One of "none" or "terrain".

uphill_factor Numeric > 0. Additional uphill movement penalty when anisotropy = "terrain".

downhill_factor
Numeric > 0. Relative ease of downhill movement when anisotropy = "terrain".

geodesic_engine
Character. One of "classic" or "multisource".

Value

A prepared geodesic context object for repeated geodesic allocation.

Examples

```
boundary_sf <- sf::st_sf(
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(
      c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
    )))
  ),
  crs = 32636
)

prep <- prepare_geodesic_context(
  boundary_sf = boundary_sf,
  res = 20,
  geodesic_engine = "classic"
)

names(prep)
inherits(prep$mask_r, "SpatRaster")


prep2 <- prepare_geodesic_context(
  boundary_sf = boundary_sf,
  res = 20,
  geodesic_engine = "multisource"
)
```

---

weighted_voronoi *Weighted Euclidean tessellation (core)*

Description

Internal/core function used by weighted_voronoi_domain() to compute a weighted Euclidean tessellation on a rasterised domain.

Usage

```
weighted_voronoi(
  points_sf,
  weight_col,
  boundary = NULL,
  template_rast = NULL,
  res = NULL,
  weight_transform = function(w) w,
  weight_model = c("multiplicative", "power", "additive"),
  weight_power = 1,
  method = c("argmin", "partition"),
  max_dist = NULL,
  verbose = TRUE,
  island_min_cells = 5,
  island_fill_iter = 50
)
```

Arguments

points_sf	An sf POINT object containing generator locations and attributes.
weight_col	Character. Name of the weight column in points_sf.
boundary	Optional sf polygon defining the tessellation domain. Used when template_rast is NULL.
template_rast	Optional terra::SpatRaster template raster. Provide this instead of boundary + res.
res	Numeric. Raster resolution in CRS units.
weight_transform	Function used to transform weights before allocation. Must return finite, strictly positive values.
weight_model	Character. One of "multiplicative", "power", or "additive". Controls how distances and weights combine into effective cost.
weight_power	Numeric > 0. Only used when weight_model = "power". Controls the distance exponent.
method	Character. Allocation method; one of "argmin" or "partition".
max_dist	Optional numeric. Maximum Euclidean distance to consider (euclidean only).
verbose	Logical. If TRUE, prints progress.
island_min_cells	Integer. Minimum patch size used in island removal.
island_fill_iter	Integer. Maximum iterations for filling reassigned cells.

Value

A list containing polygon output (if requested), allocation raster, and weights.

`weighted_voronoi_domain` *Weighted tessellation in a constrained polygon domain*

Description

Creates a complete, connected tessellation of a polygonal domain using either weighted Euclidean distance or weighted geodesic (domain-constrained) distance. Weights are supplied as an attribute of generator points and can be transformed by a user-defined function prior to allocation.

Usage

```
weighted_voronoi_domain(
  points_sf,
  weight_col,
  boundary_sf,
  res = 20,
  weight_transform = function(w) w,
  weight_model = c("multiplicative", "power", "additive"),
  weight_power = 1,
  distance = c("euclidean", "geodesic"),
  max_dist = NULL,
  island_min_cells = 5,
  island_fill_iter = 50,
  clip_to_boundary = TRUE,
  close_mask = TRUE,
  close_iters = 1,
  resistance_rast = NULL,
  dem_rast = NULL,
  use_tobler = TRUE,
  tobler_v0_kmh = 6,
  tobler_a = 3.5,
  tobler_b = 0.05,
  min_speed_kmh = 0.25,
  anisotropy = c("none", "terrain"),
  uphill_factor = 1,
  downhill_factor = 1,
  geodesic_engine = c("classic", "multisource"),
  prepared = NULL,
  verbose = TRUE
)
```

Arguments

`points_sf`	An `sf` POINT object containing generator locations and attributes.
`weight_col`	Character. Name of the weight column in `points_sf`.

`boundary_sf` An `sf` POLYGON or MULTIPOLYGON defining the domain.

`res` Numeric. Raster resolution in CRS units.

`weight_transform`
Function used to transform weights before allocation. Must return finite, strictly positive values.

`weight_model` Character. One of `"multiplicative"`, `"power"`, or `"additive"`. Controls how distances and weights combine into effective cost.

`weight_power` Numeric greater than 0. Only used when `weight_model = "power"`. Controls the distance exponent.

`distance` Character. One of `"euclidean"` or `"geodesic"`.

`max_dist` Optional numeric. Maximum Euclidean distance to consider (euclidean only).

`island_min_cells`
Integer. Minimum patch size used in island removal.

`island_fill_iter`
Integer. Maximum iterations for filling reassigned cells.

`clip_to_boundary`
Logical. If `TRUE`, polygon output is intersected with the input boundary for exact edge matching (euclidean only).

`close_mask` Logical. If `TRUE`, applies a morphological closing to the raster mask (geodesic only).

`close_iters` Integer. Number of closing iterations (geodesic only).

`resistance_rast`
Optional `SpatRaster` giving movement resistance values greater than 0. Overrides `dem_rast` and Tobler-based resistance when provided.

`dem_rast` Optional `SpatRaster` providing elevation or movement-cost information. Must align with the tessellation domain and resolution.

`use_tobler` Logical. If `TRUE`, applies Tobler's hiking function to convert slope into isotropic movement cost.

`tobler_v0_kmh` Base walking speed on flat terrain, in km/h.

`tobler_a` Tobler exponential slope coefficient.

`tobler_b` Tobler slope multiplier.

`min_speed_kmh` Minimum allowed speed to avoid infinite costs.

`anisotropy` Character. Directional cost model for geodesic distance.

`"none"` Standard isotropic geodesic distance (default).

`"terrain"` Direction-dependent movement based on terrain slope (`dem_rast` required).

`uphill_factor` Numeric greater than 0. Multiplier controlling the additional cost of uphill movement when `anisotropy = "terrain"`. Values greater than 1 penalise uphill movement more strongly.

`downhill_factor`
Numeric greater than 0. Multiplier controlling the ease of downhill movement when `anisotropy = "terrain"`. Values greater than 1 make downhill travel easier.

`geodesic_engine`
: Character. Geodesic allocation engine to use when `distance = "geodesic"`.

  "classic" Per-generator accumulated-cost allocation. Supports all current geodesic modes and weight models.

  "multisource" Single-pass multisource allocation. Currently supported only for `weight_model = "additive"` and `anisotropy = "none"`.

`prepared`
: Optional prepared geodesic context created by `prepare_geodesic_context()` for repeated compatible geodesic runs.

`verbose`
: Logical. If `TRUE`, prints progress.

Details

When `distance = "geodesic"`, distances are computed as shortest paths constrained to the spatial domain. If `dem_rast` is supplied and `use_tobler = TRUE`, movement cost between adjacent raster cells is modified using Tobler's hiking function so that steeper slopes increase effective distance. This allows elevation or resistance surfaces to influence spatial allocation while preserving a complete tessellation.

When `distance = "geodesic"` and `anisotropy = "terrain"`, movement costs are computed using a direction-dependent extension of a Tobler-like hiking function. Movement between raster cells becomes asymmetric, so uphill and downhill transitions have different costs.

Currently, anisotropic terrain mode:

- requires a `dem_rast` input
- does not combine with a user-supplied `resistance_rast`
- uses 8-directional neighbourhood transitions

For geodesic allocation, `geodesic_engine = "classic"` computes one accumulated-cost surface per generator and assigns each raster cell to the minimum effective cost. This is the reference implementation and supports all current geodesic modes.

`geodesic_engine = "multisource"` provides a scalable alternative for additive-weight isotropic geodesic tessellations. It uses a single multisource shortest-path propagation and is currently available only when `weight_model = "additive"` and `anisotropy = "none"`.

Value

A list with elements including:

polygons An `sf` object with one polygon per generator.

allocation A `terra::SpatRaster` assigning each cell to a generator.

summary A generator-level summary table.

diagnostics A list of diagnostic metrics and settings.

Examples

```
boundary_sf <- sf::st_sf(
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(
      c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
    )))
  ),
  crs = 32636
)

points_sf <- sf::st_sf(
  population = c(50, 200, 1000),
  geometry = sf::st_sfc(
    sf::st_point(c(20, 20)),
    sf::st_point(c(80, 25)),
    sf::st_point(c(50, 50))
  ),
  crs = 32636
)

out <- weighted_voronoi_domain(
  points_sf = points_sf,
  weight_col = "population",
  boundary_sf = boundary_sf,
  res = 10,
  weight_transform = log10,
  distance = "euclidean",
  verbose = FALSE
)

names(out)
out$summary


boundary_sf <- sf::st_sf(
  id = 1,
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(
      c(0, 0), c(1000, 0), c(1000, 1000), c(0, 1000), c(0, 0)
    )))
  ),
  crs = 3857
)

points_sf <- sf::st_sf(
  population = c(1, 1),
  geometry = sf::st_sfc(
    sf::st_point(c(200, 500)),
    sf::st_point(c(800, 500))
  ),
  crs = 3857
)
```

```
dem_rast <- terra::rast(
  xmin = 0, xmax = 1000, ymin = 0, ymax = 1000,
  resolution = 50,
  crs = "EPSG:3857"
)

xy <- terra::crds(dem_rast, df = TRUE)
terra::values(dem_rast) <- xy$x * 20

out <- weighted_voronoi_domain(
  points_sf = points_sf,
  weight_col = "population",
  boundary_sf = boundary_sf,
  distance = "geodesic",
  dem_rast = dem_rast,
  anisotropy = "terrain",
  uphill_factor = 3,
  downhill_factor = 1.2,
  res = 50,
  verbose = FALSE
)
```

`weighted_voronoi_geodesic` *Weighted geodesic tessellation (core)*

Description

Computes a weighted tessellation using domain-constrained (geodesic) distances. Distances are calculated as shortest-path distances through a rasterised domain mask.

Usage

```
weighted_voronoi_geodesic(
  points_sf,
  weight_col,
  boundary_sf,
  res = 20,
  weight_transform = function(w) w,
  weight_model = c("multiplicative", "power", "additive"),
  weight_power = 1,
  close_mask = TRUE,
  close_iters = 1,
  resistance_rast = NULL,
  dem_rast = NULL,
  use_tobler = TRUE,
  tobler_v0_kmh = 6,
```

```
  tobler_a = 3.5,
  tobler_b = 0.05,
  min_speed_kmh = 0.25,
  anisotropy = c("none", "terrain"),
  uphill_factor = 1,
  downhill_factor = 1,
  island_min_cells = 5,
  island_fill_iter = 50,
  geodesic_engine = c("classic", "multisource"),
  return_polygons = TRUE,
  prepared = NULL,
  verbose = TRUE
)
```

Arguments

points_sf	An sf POINT object containing generator locations and attributes.
weight_col	Character. Name of the weight column in points_sf.
boundary_sf	An sf POLYGON or MULTIPOLYGON defining the domain.
res	Numeric. Raster resolution in CRS units.
weight_transform	Function used to transform weights before allocation. Must return finite, strictly positive values.
weight_model	Character. One of "multiplicative", "power", or "additive". Controls how distances and weights combine into effective cost.
weight_power	Numeric > 0. Only used when weight_model = "power". Controls the distance exponent.
close_mask	Logical. If TRUE, applies a morphological closing to the raster mask (geodesic only).
close_iters	Integer. Number of closing iterations (geodesic only).
resistance_rast	Optional SpatRaster giving movement resistance values greater than 0. Overrides dem_rast and Tobler-based resistance when provided.
dem_rast	Optional SpatRaster providing elevation or movement-cost information. Must align with the tessellation domain and resolution.
use_tobler	Logical. If TRUE, applies Tobler's hiking function to convert slope into isotropic movement cost.
tobler_v0_kmh	Base walking speed on flat terrain, in km/h.
tobler_a	Tobler exponential slope coefficient.
tobler_b	Tobler slope multiplier.
min_speed_kmh	Minimum allowed speed to avoid infinite costs.
anisotropy	Character. Directional cost model for geodesic distance.
	"none" Standard isotropic geodesic distance (default).

"terrain" Direction-dependent movement based on terrain slope (`dem_rast` required).

`uphill_factor` Numeric greater than 0. Multiplier controlling the additional cost of uphill movement when `anisotropy = "terrain"`. Values greater than 1 penalise uphill movement more strongly.

`downhill_factor`
Numeric greater than 0. Multiplier controlling the ease of downhill movement when `anisotropy = "terrain"`. Values greater than 1 make downhill travel easier.

`island_min_cells`
Integer. Minimum patch size used in island removal.

`island_fill_iter`
Integer. Maximum iterations for filling reassigned cells.

`geodesic_engine`
Character. Geodesic allocation engine; one of `"classic"` or `"multisource"`.

`return_polygons`
Logical. If `TRUE`, polygonise the cleaned allocation raster and attach point attributes. If `FALSE`, return allocation outputs only.

`prepared` Optional prepared geodesic context created by `prepare_geodesic_context()` for repeated compatible geodesic runs.

`verbose` Logical. If `TRUE`, prints progress.

Value

A list containing polygon output, allocation raster, and weights.

Examples

```
boundary_sf <- sf::st_sf(
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(
      c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
    )))
  ),
  crs = 32636
)

points_sf <- sf::st_sf(
  w = c(2, 5),
  geometry = sf::st_sfc(
    sf::st_point(c(20, 20)),
    sf::st_point(c(80, 80))
  ),
  crs = 32636
)

out <- weighted_voronoi_geodesic(
  points_sf = points_sf,
  weight_col = "w",
```

```
  boundary_sf = boundary_sf,
  res = 20,
  verbose = FALSE
)

names(out)
terra::freq(out$allocation)


prep <- prepare_geodesic_context(boundary_sf, res = 20, geodesic_engine = "classic")
out2 <- weighted_voronoi_geodesic(
  points_sf = points_sf,
  weight_col = "w",
  boundary_sf = boundary_sf,
  prepared = prep,
  res = 20,
  verbose = FALSE
)
```

`weighted_voronoi_time` *Temporal weighted tessellation*

Description

Runs weighted tessellation across a sequence of time-specific point datasets and returns a stack of allocation rasters, with optional polygons and summaries per time step.

Usage

```
weighted_voronoi_time(
  points_list,
  weight_col,
  boundary_sf,
  time_index = NULL,
  distance = c("euclidean", "geodesic"),
  geodesic_engine = c("multisource", "classic"),
  res = 20,
  resistance_list = NULL,
  dem_list = NULL,
  keep_polygons = FALSE,
  keep_summaries = TRUE,
  prepared = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

points_list	A non-empty list of sf POINT objects, one per time step.
weight_col	Character. Name of the weight column present in each element of points_list.
boundary_sf	An sf POLYGON/MULTIPOLYGON defining the domain.
time_index	Optional character vector of time labels. Defaults to names(points_list) if present, otherwise "t1", "t2", etc.
distance	Character. One of "euclidean" or "geodesic".
geodesic_engine	Character. Geodesic engine to use when distance = "geodesic".
res	Numeric. Raster resolution in CRS units (e.g. metres).
resistance_list	Optional list of resistance rasters, either length 1 (reused for all times) or the same length as points_list.
dem_list	Optional list of DEM rasters, either length 1 (reused for all times) or the same length as points_list.
keep_polygons	Logical. If TRUE, return polygons for each time step.
keep_summaries	Logical. If TRUE, return summaries for each time step.
prepared	Optional prepared geodesic context created by prepare_geodesic_context() for repeated compatible geodesic runs.
verbose	Logical. If TRUE, prints progress.
...	Additional arguments passed to weighted_voronoi_domain().

Details

This first implementation assumes a static boundary and runs each time step independently. Time-varying weights, point locations, and resistance/DEM surfaces are supported by supplying separate inputs for each time step.

Value

A list containing:

allocations A terra::SpatRaster with one allocation layer per time step.

time_index Character vector of time labels.

change_map_first_last A terra::SpatRaster indicating whether allocation changed between the first and last time step (1 = changed, 0 = unchanged).

persistence A terra::SpatRaster indicating whether each cell retained the same allocation across all time steps (1 = persistent, 0 = changed at least once).

polygons Optional list of sf polygon outputs by time.

summaries Optional list of summary tables by time.

Examples

```
boundary_sf <- sf::st_sf(
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(
      c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
    )))
  ),
  crs = 32636
)

pts1 <- sf::st_sf(
  w = c(2, 5),
  geometry = sf::st_sfc(
    sf::st_point(c(20, 20)),
    sf::st_point(c(80, 80))
  ),
  crs = 32636
)

pts2 <- sf::st_sf(
  w = c(3, 4),
  geometry = sf::st_sfc(
    sf::st_point(c(30, 20)),
    sf::st_point(c(70, 80))
  ),
  crs = 32636
)

out <- weighted_voronoi_time(
  points_list = list(t1 = pts1, t2 = pts2),
  weight_col = "w",
  boundary_sf = boundary_sf,
  distance = "euclidean",
  res = 20,
  verbose = FALSE
)

names(out)
names(out$allocations)


out_g <- weighted_voronoi_time(
  points_list = list(t1 = pts1, t2 = pts2),
  weight_col = "w",
  boundary_sf = boundary_sf,
  distance = "geodesic",
  geodesic_engine = "classic",
  res = 20,
  verbose = FALSE
)
```

`weighted_voronoi_uncertainty` *Uncertainty-aware weighted tessellation*

Description

Repeats weighted tessellation under stochastic perturbation of generator weights and summarises the results as per-cell membership probabilities, modal allocation, and entropy.

Usage

```
weighted_voronoi_uncertainty(
  points_sf,
  weight_col,
  boundary_sf,
  n_sim = 100,
  weight_sd = NULL,
  distance = c("euclidean", "geodesic"),
  geodesic_engine = c("multisource", "classic"),
  res = 20,
  keep_simulations = FALSE,
  seed = NULL,
  warn_zero_entropy = TRUE,
  prepared = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`points_sf`	An `sf` POINT object containing generator locations and attributes.
`weight_col`	Character. Name of the weight column in `points_sf`.
`boundary_sf`	An `sf` POLYGON/MULTIPOLYGON defining the domain.
`n_sim`	Integer. Number of simulation runs.
`weight_sd`	Optional numeric. Standard deviation of lognormal weight perturbation on the log scale. If `NULL`, no perturbation is applied and the same tessellation is repeated.
`distance`	Character. One of `"euclidean"` or `"geodesic"`.
`geodesic_engine`	Character. Geodesic engine to use when `distance = "geodesic"`. Defaults to `"multisource"`.
`res`	Numeric. Raster resolution in CRS units (e.g. metres).
`keep_simulations`	Logical. If `TRUE`, return the simulated allocation rasters as a stack.

seed	Optional integer random seed for reproducibility.
warn_zero_entropy	Logical. If TRUE, warn when all entropy values are zero across the domain.
prepared	Optional prepared geodesic context created by prepare_geodesic_context() for repeated compatible geodesic runs.
verbose	Logical. If TRUE, prints progress.
...	Additional arguments passed to weighted_voronoi_domain().

Details

This first implementation supports uncertainty in generator weights only. Weights are perturbed independently across simulations using a lognormal multiplicative model:

`w_sim = w * exp(N(0, weight_sd))`

The output includes:

probabilities A `terra::SpatRaster` with one layer per generator, containing the fraction of simulations in which each cell was assigned to that generator.

modal_allocation A `terra::SpatRaster` giving the most probable generator for each cell.

entropy A `terra::SpatRaster` showing per-cell uncertainty. Higher values indicate less stable allocation across simulations.

Value

A list with probability surfaces, modal allocation, entropy, and optionally the full simulation stack.

Examples

```
boundary_sf <- sf::st_sf(
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(
      c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
    )))
  ),
  crs = 32636
)

points_sf <- sf::st_sf(
  w = c(2, 5, 3),
  geometry = sf::st_sfc(
    sf::st_point(c(20, 20)),
    sf::st_point(c(80, 80)),
    sf::st_point(c(20, 80))
  ),
  crs = 32636
)

out <- weighted_voronoi_uncertainty(
  points_sf = points_sf,
  weight_col = "w",
```

```
  boundary_sf = boundary_sf,
  n_sim = 5,
  distance = "euclidean",
  res = 20,
  seed = 1,
  verbose = FALSE
)

names(out)


out_g <- weighted_voronoi_uncertainty(
  points_sf = points_sf,
  weight_col = "w",
  boundary_sf = boundary_sf,
  n_sim = 5,
  distance = "geodesic",
  geodesic_engine = "classic",
  res = 20,
  seed = 1,
  verbose = FALSE
)
```

Index