Package: geomattR (via r-universe)

October 19, 2026

Type Package

Title Calculate Geometric Attributes of Spatial Polygons

Version 0.3.1

Description Calculate geometric attributes of spatial polygons including area, perimeter, compactness, elongation, orientation (bearing), fractal dimension, and various shape indices. The package relies heavily on 'terra' methods and vector data objects <https://github.com/rspatial/terra>.

License MIT + file LICENSE

URL https://github.com/gortegasolis/geomattR

BugReports https://github.com/gortegasolis/geomattR/issues

Encoding UTF-8

Depends R (>= 4.1)

Roxygen list(markdown = TRUE)

Imports terra (>= 1.8.70), geosphere (>= 1.5.20), methods, parallel, stats, cli, rlang

Suggests testthat (>= 3.2.3), sf, knitr (>= 1.50), rmarkdown (>= 2.30)

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libproj-dev libsqlite3-dev

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

Date/Publication 2026-10-08 12:57:05 UTC

RemoteUrl https://github.com/gortegasolis/geomattr

RemoteRef HEAD

RemoteSha 628fdf1fb01500b71b0d10e55aa2acc757bb23a7

Contents

geomattR-package 2
calc_elongation 4
calc_extent 5
calculate_geometric_attributes 7
get_distant_points 11

Index **13**

`geomattR-package` *geomattR: Calculate Geometric Attributes of Spatial Polygons*

Description

Calculate geometric attributes of spatial polygons. Computes area, perimeter, compactness, elongation, orientation, fractal dimension, and shape indices suitable for geospatial analysis, urban planning, and environmental science applications.

By default, measurements use geodesic calculations for accuracy across large study regions (i.e. continental scale studies). The package supports the `"geo"`, `"haversine"`, and `"cosine"` methods and automatically handles geographic (lon/lat) and projected CRS appropriately.

Details

Main Functions:

- `calculate_geometric_attributes()`: Calculate metrics for one or more polygons. Runs sequentially by default; pass a `cl` argument for parallel execution.

Helper Functions:

- `get_distant_points()`: Find most distant points on convex hull
- `calc_elongation()`: Calculate elongation ratio
- `calc_extent()`: Calculate east-west and north-south extents

Geodesic Measurements:

All calculations prioritize accuracy:

- **Area & Perimeter**: Geodesic by default; projected input is handled transparently for methods that need geographic coordinates
- **Distances**: Support `method = "geo"`, `"haversine"`, and `"cosine"`. All three are lon/lat great-circle methods; on a projected CRS, `terra::distance()` ignores `method` and computes Cartesian distances
- **Bearing**: Computed from geographic coordinates when needed
- **Automatic Projection**: Non-geographic CRS are handled transparently

Precise definitions of each reported metric are documented in `calculate_geometric_attributes()`.

Example:

```
library(geomattR)
library(terra)

# Create sample polygon in WGS84
coords <- cbind(c(0, 0, 1, 1, 0), c(0, 1, 1, 0, 0))
polygon <- vect(coords, type = "polygon", crs = "EPSG:4326")

# Calculate all attributes (geodesic by default)
result <- calculate_geometric_attributes(polygon)

# Calculate specific metrics
result <- calculate_geometric_attributes(polygon,
  metrics = c("area", "perimeter", "compactness"))

# Parallel processing
cl <- parallel::makeCluster(2)
result <- calculate_geometric_attributes(polygon, cl = cl)
parallel::stopCluster(cl)
```

Acknowledgment

This package began as a side project during my time as a technician on BEAST – BiodivErsity dynamics Across a continuum of Space, Time, and their scales, led by Petr Keil, and was therefore partially funded by the European Union (ERC, BEAST, grant no. 101044740). Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency; neither the European Union nor the granting authority can be held responsible for them.

Discussions with P. Keil on spatial and geometric problems inspired the package. G. Ortega-Solís developed all core functions; Claude Code was used to double-check the code and to help write unit tests.

Author(s)

Maintainer: Gabriel Ortega-Solís <g.ortega.solis@geomattr.org> (ORCID) [copyright holder]

Authors:

- Gabriel Ortega-Solís <g.ortega.solis@geomattr.org> (ORCID) [copyright holder]

See Also

Useful links:

- https://github.com/gortegasolis/geomattR
- Report bugs at https://github.com/gortegasolis/geomattR/issues

`calc_elongation` *Calculate Elongation Ratio from Minimum Bounding Rectangle*

Description

Calculates the elongation ratio of a polygon based on its minimum bounding rectangle. The minimum bounding rectangle is computed from the convex hull of the input polygon.

Usage

```
calc_elongation(
  v,
  isHull = FALSE,
  output = "value",
  method = "geo",
  by_feature = FALSE
)
```

Arguments

`v`	A SpatVector object representing a polygon, or a pre-computed convex hull if `isHull = TRUE`.
`isHull`	Logical. If `TRUE`, `v` is treated as a pre-computed convex hull. If `FALSE` (default), the convex hull is computed internally.
`output`	Character string. Either `"value"` (default) to return the elongation ratio, or `"polygon"` to return the input geometry with `elongation_rectangle` added as an attribute.
`method`	Character string passed to `terra::distance()`. Defaults to `"geo"` (recommended). Other supported options are `"haversine"` and `"cosine"`. All three are lon/lat great-circle methods; on a projected CRS, `method` is ignored and distances are Cartesian. See `distance` for more information.
`by_feature`	Logical. If `FALSE` (default), compute a single response from the convex hull of the whole input set. If `TRUE`, compute one response per polygon feature.

Details

In the current implementation, elongation is computed as the ratio of the mean of the two largest side lengths to the mean of the two shortest side lengths: $E = \frac{\text{mean(long sides)}}{\text{mean(short sides)}}$

The minimum bounding rectangle is a planar (GEOS) construction. For lon/lat input it is computed in a local Lambert azimuthal equal-area projection centred on the feature, so that the *minimum* rectangle is not distorted by the varying length of a degree of longitude; the rectangle is then measured back in the input's CRS with the selected `method`.

This approximation assumes the minimum bounding rectangle is unique, which may not always be the case. Consider this with caution.

Value

If `output = "value"`, a numeric value representing the elongation ratio of the input polygon from the current minimum-bounding-rectangle approximation. If `output = "polygon"`, the input geometry with `elongation_rectangle` added. When `by_feature = TRUE`, value output is returned as a numeric vector (one value per feature). Higher values indicate more elongated shapes.

References

Dražić, Slobodan, Nebojša Ralević, and Joviša Žunić. Shape Elongation from Optimal Encasing Rectangles. *Computers & Mathematics with Applications 60, no. 7 (2010): 2035–42*. doi:10.1016/j.camwa.2010.07.043.

Examples

```
library(terra)
coords <- cbind(c(0, 0, 1, 1, 0), c(0, 1, 1, 0, 0))
polygon <- vect(coords, type = "polygon", crs = "EPSG:4326")
elongation <- calc_elongation(polygon)

# Using a pre-computed convex hull
hull_geom <- terra::hull(polygon, type = "convex")
elongation_hull <- calc_elongation(hull_geom, isHull = TRUE)
```

`calc_extent` *Calculate Polygon Extent*

Description

Calculates east-west and/or north-south extent of a polygon based on its geographic bounding box corners.

Usage

```
calc_extent(
  v,
  isHull = FALSE,
  direction = c("ew", "ns"),
  output = "value",
  method = "geo",
  by_feature = FALSE
)
```

Arguments

`v`	A SpatVector object representing a polygon, or a pre-computed convex hull if `isHull = TRUE`.
`isHull`	Logical. If `TRUE`, `v` is treated as a pre-computed convex hull. If `FALSE` (default), the convex hull is computed internally.

direction	Character vector indicating which extent(s) to return when output = "value" or output = "polygon". Valid values are "ew" and "ns". Default is both.
output	Character string. Either "value" (default) to return extent values, or "polygon" to return the input geometry with the requested extent columns added.
method	Character string passed to terra::distance(). Defaults to "geo" (recommended). Other supported options are "haversine" and "cosine". All three are lon/lat great-circle methods; on a projected CRS, method is ignored and distances are Cartesian. See distance for more information.
by_feature	Logical. If FALSE (default), compute a single response from the convex hull of the whole input set. If TRUE, compute one response per polygon feature.

Details

Extent is computed from the bounding box of the polygon's convex hull. The Southwest (SW), Southeast (SE), Northwest (NW), and Northeast (NE) corners of this box are evaluated using terra::distance() with the selected method.

The metric values are calculated as:

- **East-West Extent**: The average length of the top and bottom bounding box edges:

$$\text{EW} = \frac{d(SW, SE) + d(NW, NE)}{2}$$

- **North-South Extent**: The average length of the left and right bounding box edges:

$$\text{NS} = \frac{d(SW, NW) + d(SE, NE)}{2}$$

When method is "geo" or "haversine", the hull is projected to EPSG:4326 when needed before distance calculations.

For output = "value":

- direction = "ew": returns east-west extent (numeric scalar)
- direction = "ns": returns north-south extent (numeric scalar)
- direction = c("ew", "ns") (any order): returns both values as a named numeric vector

For output = "polygon", only the requested extent columns are added as attributes.

For interpretation and relation to other shape metrics, see calculate_geometric_attributes.

Value

If output = "value" and one direction is requested, returns a numeric scalar. If both directions are requested, returns a named numeric vector with ew_length and ns_length. If output = "polygon", returns the input geometry with the requested extent columns added. When by_feature = TRUE, value output is returned per feature (numeric vector for one direction, data.frame for both directions).

Examples

```
library(terra)
coords <- cbind(c(0, 0, 1, 1, 0), c(0, 1, 1, 0, 0))
polygon <- vect(coords, type = "polygon", crs = "EPSG:4326")

# Get one extent value
ew_extent <- calc_extent(polygon, direction = "ew")

# Get both extent values
both_extents <- calc_extent(polygon, direction = c("ns", "ew"))

# Return polygon with only requested extent columns
polygon_with_ew <- calc_extent(polygon, direction = "ew", output = "polygon")

# Return polygon with both extent columns
polygon_with_extents <- calc_extent(polygon, output = "polygon")

# Use a custom distance method supported by terra::distance()
extents_planar <- calc_extent(polygon, direction = c("ew", "ns"), method = "cosine")
```

---

calculate_geometric_attributes

Calculate Geometric Attributes of Spatial Polygons

Description

Calculates a comprehensive set of geometric attributes for spatial polygon features. Supports both sequential and parallel execution through an optional cluster argument.

Usage

```
calculate_geometric_attributes(
  v,
  metrics = "all",
  cl = NULL,
  method = "geo",
  by_feature = TRUE
)
```

Arguments

`v`	A SpatVector object representing one or more polygons.
`metrics`	Character string or vector specifying which metrics to calculate. Options are: • `"all"` (default): Calculate all available metrics • A single metric name as a string: Calculate only that metric • A character vector of metric names: Calculate specified metrics

Available metric names: "area", "perimeter", "compactness", "reock", "elongation_rectangle", "num_holes", "hole_area", "hole_area_pct", "num_polygons", "ew_length", "ns_length", "maxlength", "bearing", "northerness", "fractaldimension", "sinuosity", "shape_index", "circularity_ratio", "decimallongitude", "decimallatitude"

`cl` A cluster object created by `makeCluster`, or `NULL` (default). When `NULL` features are processed sequentially. Passing a cluster enables parallel processing across the cluster workers.

`method` Character string passed to `terra::distance()` for distance-based metrics. One of `"geo"` (default, recommended), `"haversine"`, or `"cosine"`. All three are lon/lat great-circle methods; on a projected CRS, `method` is ignored and distances are Cartesian.

`by_feature` Logical. If `TRUE` (default), compute metrics for each feature independently. If `FALSE`, non-spatial columns are dropped, features are dissolved into a single geometry, and one set of metrics is returned.

Details

Each polygon is processed independently via the internal function `.calculate_geometric_attributes_single`, which handle a single feature at a time, but it is optimized to ensure that intermediate objects required for multiple metrics (convex hull, minimum circle, etc.) are computed only once per feature.

Sequential vs. Parallel:

- When `cl = NULL` (default), features are processed sequentially with `lapply`.
- When `cl` is a valid cluster, features are distributed across workers via `parLapply`. The function handles library loading and function export to the cluster automatically.

Geodesic Measurements: The function supports the methods `"geo"` (default), `"haversine"` and `"cosine"` (less precise but faster great-circle approximations). All three are lon/lat great-circle methods in `terra::distance()`; on a projected CRS `method` is ignored and distances are Cartesian. Inputs are internally projected to EPSG:4326 when a metric requires geographic coordinates and restored to the original CRS on return when applicable.

Scope of the `method` parameter: The `method` argument is passed to `terra::distance()` for distance-based metrics (extent, maxlength, and elongation). Area (`terra::expanse()`) and perimeter (`terra::perim()`) use terra's built-in CRS-dependent calculations: geodesic for geographic (lon/lat) CRS, Cartesian for projected CRS. When `method = "geo"` or `"haversine"` and the input has a projected CRS, the geometry is temporarily reprojected to EPSG:4326 where required by downstream metrics.

Size metrics:

- Area: geodesic by default; handled according to the selected method.
- Perimeter: geodesic by default; handled according to the selected method.
- East-west extent: $\bar{D}_{EW} = \frac{d(NW,NE)+d(SW,SE)}{2}$.
- North-south extent: $\bar{D}_{NS} = \frac{d(SW,NW)+d(SE,NE)}{2}$.
- Maximum length (Feret diameter): $L_{\max} = \max_{p_i,p_j} d(p_i, p_j)$.

Shape metrics:

- Compactness (Polsby-Popper): area normalized by perimeter, relative to a circle: $C = \frac{4\pi A}{P^2}$.
- Reock compactness: $R = \frac{A}{A_{MEC}}$. The minimum enclosing circle is computed in a local Lambert azimuthal equal-area projection centred on the feature (GEOS hulls are planar), then measured back in the input's CRS.
- Shape index (perimeter relative to a circle of equal area): $SI = \frac{P}{2\sqrt{\pi A}}$.
- Circularity ratio: $CR = \frac{4A}{\pi L_{\max}^2}$.
- Fractal dimension (FRAGSTATS convention; approaches 1 for simple, square-like shapes and 2 for highly convoluted boundaries): $D = \frac{2\ln(0.25P)}{\ln(A)}$.
- Sinuosity: $S = \frac{P}{L_{\max}}$.

Orientation metrics:

- Bearing: orientation of the longest axis; computed from geographic coordinates when needed.
- Northerness: $N = \cos(\text{bearing} \times \frac{\pi}{180})$.
- Elongation: ratio of major to minor axis of the minimum bounding rectangle. Elongation is calculated as the ratio of the mean of the two largest side lengths to the mean of the two shortest side lengths: $E = \frac{\text{mean(long sides)}}{\text{mean(short sides)}}$.

Hole and part metrics:

- Hole area percentage, relative to the gross area (including holes): $HA\% = \frac{A_{\text{holes}}}{A+A_{\text{holes}}} \times 100$.
- `num_holes`, `hole_area`, and `num_polygons` are direct counts/areas derived from polygon topology.

Value

The input SpatVector with additional columns containing the requested geometric attributes:

- `area`: Total area in square meters. Wrapper around `terra::expanse()`.
- `perimeter`: Total perimeter length in meters. Wrapper around `terra::perim()`.
- `compactness`: Polsby-Popper compactness.
- `reock`: Reock compactness.
- `elongation_rectangle`: Elongation index from the minimum bounding rectangle.
- `num_holes`: Number of holes (interior rings) in the polygon.
- `hole_area`: Total area of holes in square meters.
- `hole_area_pct`: Percentage of total area occupied by holes.
- `num_polygons`: Number of separate polygon parts (multi-part count).
- `ew_length`: East-west extent in meters.
- `ns_length`: North-south extent in meters.
- `maxlength`: Maximum Feret distance across the convex hull.
- `bearing`: Axial orientation of the maximum length line: geographic bearing from the southernmost to the northernmost point in decimal degrees, within [-90, 90] (0 = north-south, -90/90 = east-west).

- `northerness`: Cosine of bearing, within [0, 1] (1 = north-south axis, 0 = east-west axis).
- `fractaldimension`: Boundary complexity index (FRAGSTATS convention).
- `sinuosity`: Perimeter-to-maximum-length ratio.
- `shape_index`: Dimensionless irregularity index.
- `circularity_ratio`: Circularity index based on area and maximum hull distance.
- `decimallongitude`: Centroid longitude in decimal degrees.
- `decimallatitude`: Centroid latitude in decimal degrees.

References

Reock, E. C. (1961). Measuring compactness as a requirement of legislative apportionment. *Midwest Journal of Political Science*, 5(1), 70-74. doi:10.2307/2109043

Dražić, Slobodan, Nebojša Ralević, and Joviša Žunić. Shape Elongation from Optimal Encasing Rectangles. *Computers & Mathematics with Applications 60, no. 7 (2010): 2035–42*. doi:10.1016/j.camwa.2010.07.043.

McGarigal, K., and B. J. Marks. FRAGSTATS: Spatial Pattern Analysis Program for Quantifying Landscape Structure. *Gen. Tech. Rep. PNW-GTR-351*. Portland, OR: USDA Forest Service, Pacific Northwest Research Station (1995). doi:10.2737/PNWGTR351

Examples

```
library(terra)

# Create a sample 1-degree square polygon near the equator
coords <- cbind(c(0, 0, 1, 1, 0), c(0, 1, 1, 0, 0))
polygons <- vect(coords, type = "polygon", crs = "EPSG:4326")

# Sequential processing (default)
result <- calculate_geometric_attributes(polygons)

# Calculate specific metrics only
result <- calculate_geometric_attributes(polygons,
  metrics = c("area", "perimeter", "compactness")
)


# Parallel processing with an explicit cluster. Features are distributed
# across workers, so the polygon is duplicated to give each worker one.
# Two workers are used here; adjust to your machine's resources.
two_polygons <- rbind(polygons, polygons)
cl <- parallel::makeCluster(2)
result <- calculate_geometric_attributes(two_polygons, cl = cl)
parallel::stopCluster(cl)
```

get_distant_points *Find largest distance between a pair of opposite vertices*

Description

Identifies the pair of most distant points on a polygon's convex hull and calculates their distance (using the selected `method`) and the bearing from the southernmost to the northernmost point.

Usage

```
get_distant_points(
  v,
  isHull = FALSE,
  distance = TRUE,
  bearing = TRUE,
  output = "points",
  by_feature = FALSE,
  method = "geo"
)
```

Arguments

`v`	A SpatVector object representing a polygon, or a pre-computed convex hull if `isHull = TRUE`.
`isHull`	Logical. If `TRUE`, `v` is treated as a pre-computed convex hull. If `FALSE` (default), the convex hull is computed internally.
`distance`	Logical indicating whether to return the maximum distance (default `TRUE`).
`bearing`	Logical indicating whether to return the bearing (default `TRUE`).
`output`	Character string. `"points"` (default) returns point geometry outputs. `"value"` returns numeric metric values only, and `"polygon"` returns the input geometry with requested metric columns added.
`by_feature`	Logical. If `FALSE` (default), compute a single response from the convex hull of the whole input set. If `TRUE`, compute one response per polygon feature.
`method`	Character string passed to `terra::distance()`. Defaults to `"geo"` (recommended). Other supported options are `"haversine"` and `"cosine"`. All three are lon/lat great-circle methods; on a projected CRS, `method` is ignored and distances are Cartesian. Bearing is computed from geographic coordinates when needed.

Details

Distances between **all** convex-hull vertex pairs are evaluated with `terra::distance()` using the selected `method`, so the returned pair is the true geodesic maximum (hulls are small, so the exhaustive search is cheap). An earlier rotating-calipers implementation searched antipodal pairs in planar coordinate space, which could miss the geodesic maximum on lon/lat data.

Tie-breaking & Point Ordering: If multiple antipodal pairs share the maximum distance, ties are broken by selecting the pair containing the southernmost (and then westernmost) coordinate. The points are ordered in the output such that:

- `south_point` (point 1): the southernmost (and westernmost, in case of a tie) point of the pair.
- `north_point` (point 2): the northernmost (and easternmost) point of the pair.

The geographic bearing is calculated from `south_point` to `north_point` using `geosphere::bearing()`. When needed, the geometry is internally transformed to geographic coordinates for bearing calculation.

Value

For `output = "points"`, a list containing:

`south_point`	SpatVector of the southernmost point
`north_point`	SpatVector of the northernmost point
`distance`	Maximum distance in meters (if `distance = TRUE`)
`bearing`	Axial orientation: geographic bearing from the southernmost to the northernmost point, in degrees within [-90, 90] (if `bearing = TRUE`). Because the pair is always ordered south to north, this describes the orientation of the longest axis, not a travel direction: -90/90 is east-west, 0 is north-south.

For `output = "value"`, returns a numeric scalar when one metric is requested, or a named numeric vector when both are requested. For `output = "polygon"`, returns the input geometry with the requested metric columns added. When `by_feature = TRUE`, returns per-feature outputs (list, numeric vector/data.frame, or polygon with columns, depending on `output`).

Examples

```
library(terra)
coords <- cbind(c(0, 0, 1, 1, 0), c(0, 1, 1, 0, 0))
polygon <- vect(coords, type = "polygon", crs = "EPSG:4326")
distant_pts <- get_distant_points(polygon)

# Using a pre-computed convex hull
hull_geom <- terra::hull(polygon, type = "convex")
distant_pts_hull <- get_distant_points(hull_geom, isHull = TRUE)
```

Index

calc_elongation, 4
calc_elongation(), *2*
calc_extent, 5
calc_extent(), *2*
calculate_geometric_attributes, *6*, 7
calculate_geometric_attributes(), *2*

distance, *4*, *6*

geomattR *(*geomattR-package*)*, 2
geomattR-package, 2
get_distant_points, 11
get_distant_points(), *2*

lapply, *8*

makeCluster, *8*

parLapply, *8*