Package 'rsgeo'

September 8, 2023

Title An Interface to Rust's 'geo' Library

Version 0.1.6

Description An R interface to the GeoRust crates 'geo' and 'geo-types' providing access to geometry primitives and algorithms.

URL https://github.com/JosiahParry/rsgeo, https://josiahparry.r-universe.dev/rsgeo, https://rsgeo.josiahparry.com/

License MIT + file LICENSE

Encoding UTF-8

Language en

RoxygenNote 7.2.3

Imports rlang, vctrs

Suggests sf, testthat (>= 3.0.0), wk

Config/rextendr/version 0.3.1.9000

SystemRequirements Cargo (Rust's package manager), rustc

Config/testthat/edition 3

Config/Needs/website rmarkdown

NeedsCompilation yes

Author Josiah Parry [aut, cre] (<https://orcid.org/0000-0001-9910-865X>)

Maintainer Josiah Parry <josiah.parry@gmail.com>

Repository CRAN

Date/Publication 2023-09-08 21:50:06 UTC

R topics documented:

`as_rsgeo` *Convert to an* `rsgeo` *vector*

Description

Given an vector of geometries, cast it as an `rsgeo` class object.

Usage

```
as_rsgeo(x)
```

Arguments

`x`	a geometry vector

Value

an object of class `rsgeo`

Examples

```
x <- sf::st_sfc(sf::st_point(c(0,0)))
as_rsgeo(x)
```

bearing_geodesic *Calculate Bearing*

Description

Calculates the bearing between two point geometries.

Usage

```
bearing_geodesic(x, y)

bearing_haversine(x, y)
```

Arguments

x	an object of class rs_POINT
y	an object of class rs_POINT

Value

A vector of doubles of the calculated bearing for between x and y

Examples

```
x <- geom_point(runif(10, 0, 90), rnorm(10, 1, 90))
y <- geom_point(runif(10, 0, 90), rnorm(10, 1, 90))
bearing_geodesic(x, y)
bearing_haversine(x, y)
```

bounding_boxes *Compute Geometric Boundaries*

Description

From a vector of geometries identify different types of boundaries.

Usage

```
bounding_boxes(x)

bounding_rect(x)

minimum_rotated_rect(x)

convex_hull(x)
```

```
concave_hull(x, concavity)

extreme_coords(x)

bounding_box(x)
```

Arguments

x	an object of class rsgeo
concavity	a value between 0 and 1 specifying the concavity of the convex hull

Details

Note that if you want a convex or concave hull over an entire vector of geometries you must first union or combine them using either combine_geoms() or union_geoms()

Value

- bounding_box() returns a named vector of xmin, ymin, xmax, and ymax
- bounding_boxes() returns a list of bounding box numeric vectors for each geometry
- bounding_rect() returns an rs_POLYGON of the bounding rectangle of each geometry
- convex_hull() returns an rs_POLYGON of the convex hull for each geometry
- concave_hull() returns an rs_POLYGON of the specified concavity for each geometry
- extreme_coords() returns the extreme coordinates of each geometry as a list where each element is a named vector of xmin, ymin, xmax, and ymax where each element is a Point geometry of the extreme value
- minimum_rotated_rect() returns the minimum rotated rectangle covering a geometry as an rs_POLYGON

Examples

```
lns <- geom_linestring(
  1:20,
  runif(20, -5, 5),
  rep.int(1:5, 4)
)
bounding_box(lns)
bounding_boxes(lns)
minimum_rotated_rect(lns)
convex_hull(lns)
concave_hull(lns, 0.5)
extreme_coords(lns)
```

cast_geoms *Cast geometries to another type*

Description

Cast geometries to another type

Usage

```
cast_geoms(x, to)
```

Arguments

x	an object of class rsgeo
to	a character scalar of the target geometry type. Must be one of "point", "multipoint", "polygon", "multipolygon", "linestring", or "multilinestring".

Details

The below conversions are made available. The left hand column indicates the originating vector class and the right hand column indicates the class that it will can be cast to.

Note that correctness of conversions will not be checked or verified. If you cast an rs_MULTIPOINT to an rs_POLYGON, the validity of the polygon cannot be guaranteed.

Conversions from an rs_POLYGON into an rs_LINESTRING will result in only the exterior ring of the polygon ignoring any interior rings if there are any.

From	To
rs_POINT	rs_MULTIPOINT
rs_MULTIPOINT	rs_POLYGON, rs_MULTIPOLYGON, rs_LINESTRING, rs_MULTILINESTRING
rs_POLYGON	rs_MULTIPOINT, rs_MULTIPOLYGON, rs_LINESTRING, rs_MULTILINESTRING
rs_MULTIPOLYGON	rs_MULTIPOINT, rs_MULTILINESTRING
rs_LINESTRING	rs_MULTIPOINT, rs_MULTILINESTRING, rs_POLYGON
rs_MULTILINESTRING	rs_MULTIPOINT, rs_MULTIPOLYGON

Value

An object of class rsgeo

Examples

```
ply <- geom_polygon(c(0, 1, 1, 0, 0), c(0, 0, 1, 1, 0))
cast_geoms(ply, "linestring")
cast_geoms(ply, "multipoint")
```

centroids *Extract Centroids*

Description

Given a vector of geometries, extract their centroids.

Usage

```
centroids(x)
```

Arguments

x	an object of class rsgeo

Value

an object of class rs_POINT

Examples

```
lns <- geom_linestring(1:100, runif(100, -10, 10), rep.int(1:5, 20))
centroids(lns)
```

closest_point *Find Closest Point*

Description

For a given geometry, find the closest point on that geometry to a point. The closest point may be an intersection, a single point, or unable to be determined.

Usage

```
closest_point(x, y)

closest_point_haversine(x, y)
```

Arguments

x	an object of class rsgeo
y	an object of class rs_POINT

Value

An rs_POINT vector

Examples

```
x <- geom_linestring(1:100, runif(100, 0, 90), rep.int(1:10, 10))
y <- geom_point(runif(10, 0, 90), rnorm(10, 1, 90))
closest_point(x, y)
closest_point_haversine(x, y)
```

`combine_geoms` *Combine geometries*

Description

Given a vector of geometries combine them into a single geometry.

Usage

```
combine_geoms(x)

union_geoms(x)
```

Arguments

`x`	an object of class `rsgeo`

Details

`combine_geoms()`:

`combine_geoms()` combines a vector of geometries into a vector of length one their `MULTI` counterpart.

- `rs_POINT` and `rs_MULTIPOINT` -> `rs_MULTIPOINT`
- `rs_LINESTRING` and `rs_MULTILINESTRING` -> `rs_MULTILINESTRING`
- `rs_POLYGON` and `rs_MULTIPOLYGON` -> `rs_MULTIPOLYGON`
- `rs_GEOMETRYCOLLECTION` is not supported

`union_geoms()`:

`union_geoms()` creates a union of all geometries removing repeated points or dissolving shared boundaries.

- `rs_POINT` - combines and removes repeated points
- `rs_MULTIPOINT` - combines removes repeated points
- `rs_LINESTRING` - combines and removes duplicated points
- `rs_MULTILINESTRING` - combines and removes duplicated points
- `rs_POLYGON` - unions geometries into a single geometry
- `rs_MULTIPOLYGON` - unions geometries into a single geometry

Value

An object of class `rsgeo` of length one.

Examples

```
pnts <- geom_point(runif(10), runif(10))
combine_geoms(pnts)

lns <- geom_linestring(1:100, runif(100, -10, 10), rep.int(1:5, 20))
union_geoms(lns)

x <- c(0, 1, 1, 0, 0)
y <- c(0, 0, 1, 1, 0)

p1 <- geom_polygon(x, y)
p2 <- geom_polygon(x - 1, y + 0.5)

z <- c(p1, p2)

res <- union_geoms(z)
res

if (rlang::is_installed(c("sf", "wk"))) {
  plot(z)
  plot(res, lty = 3, border = "blue", add = TRUE, lwd = 4)
}
```

---

coords *Extract Coordinates*

Description

Given an rsgeo class object, extract the object's coordinates as a data frame. Empty or missing geometries are ignored.

Usage

```
coords(x)
```

Arguments

x	an object of class rsgeo

Value

A data.frame with columns x, y. Additional columns are returned based on the geometry type. Additional columns are:

- id
- line_id: refers to the LineString ID for rs_LINESTRING, or the component LineString in a MultiLineString, or as the ring ID for a Polygon.
- multilinestring_id
- polygon_id
- multipolygon_id

Examples

```
pnt <- geom_point(3, 0.14)
mpnt <- geom_multipoint(1:10, 10:1)
ln <- geom_linestring(1:10, 10:1)
ply <- geom_polygon(c(0, 1, 1, 0, 0), c(0, 0, 1, 1, 0))

coords(pnt)
coords(mpnt)
coords(ln)
coords(union_geoms(rep(ln, 2)))
coords(ply)
coords(union_geoms(rep(ply, 2)))
```

---

distance_euclidean_pairwise

Calculate Distances

Description

Calculates distances between two vectors of geometries. There are a number of different distance methods that can be utilized.

Usage

```
distance_euclidean_pairwise(x, y)

distance_hausdorff_pairwise(x, y)

distance_vicenty_pairwise(x, y)

distance_geodesic_pairwise(x, y)

distance_haversine_pairwise(x, y)

distance_euclidean_matrix(x, y)

distance_hausdorff_matrix(x, y)

distance_vicenty_matrix(x, y)

distance_geodesic_matrix(x, y)

distance_haversine_matrix(x, y)
```

Arguments

x	and object of class rsgeo
y	and object of class rsgeo

Details

There are `_pairwise()` and `_matrix()` suffixed functions to generate distances pairwise or as a dense matrix respectively. The pairwise functions calculate distances between the ith element of each vector. Whereas the matrix functions calculate the distance between each and every geometry.

Euclidean distance should be used for planar geometries. Haversine, Geodesic, and Vicenty are all methods of calculating distance based on spherical geometries. There is no concept of spherical geometries in rsgeo, so choose your distance measure appropriately.

Notes:

- Hausdorff distance is calculated using Euclidean distance.
- Haversine, Geodesic, and Vicenty distances only work with `rs_POINT` geometries.

Value

For `_matrix` functions, returns a dense matrix of distances whereas `_pairwise` functions return a numeric vector.

Examples

```
set.seed(1)
x <- geom_point(runif(5, -1, 1), runif(5, -1, 1))
y <- rev(x)

distance_euclidean_matrix(x, y)
distance_hausdorff_matrix(x, y)
distance_vicenty_matrix(x, y)
distance_geodesic_matrix(x, y)
distance_haversine_matrix(x, y)

distance_euclidean_pairwise(x, y)
distance_hausdorff_pairwise(x, y)
distance_vicenty_pairwise(x, y)
distance_geodesic_pairwise(x, y)
distance_haversine_pairwise(x, y)
```

---

expand_geoms *Expand Geometries*

Description

Expands geometries into a list of vectors of their components.

Usage

```
expand_geoms(x)
```

Arguments

`x`	an object of class `rsgeo`

Details

- rs_MULTIPOINT expands into a vector of points
- rs_LINESTRING expands into a vector points
- rs_MULTILINESTRING expands into a vector of linestrings
- rs_POLYGON expands into a vector of linestrings
- rs_MULTIPOLYGON expands into a vector of polygons

If you wish to have a single vector returned, pass the results into flatten_geoms().

Value

A list of rsgeo vectors containing each original geometry's components as a new vector.

Examples

```
mpnts <- geom_multipoint(runif(10), runif(10), rep.int(1:5, 2))
expand_geoms(mpnts)
```

---

flatten_geoms *Flatten a list of rsgeo vectors*

Description

Flatten a list of rsgeo vectors

Usage

```
flatten_geoms(x)
```

Arguments

x	list object where each element is an object of class rsgeo

Value

Returns an object of class rsgeo

Examples

```
pnts <- replicate(
  10,
  geom_point(runif(1), runif(1)),
  simplify = FALSE
)

flatten_geoms(pnts)
```

frechet_distance *Calculate Frechet Distance*

Description

Given two LineStrings compare thier similarity by calculating the Fréchet distance.

Usage

```
frechet_distance(x, y)
```

Arguments

x	an object of class rs_LINESTRING
y	an object of class rs_LINESTRING

Value

A numeric vector

Examples

```
x <- geom_linestring(1:10, runif(10, -1, 1))
y <- geom_linestring(1:10, runif(10, -3, 3))
frechet_distance(x, y)
```

geom_point *Construct Geometries*

Description

Constructs geometries from numeric vectors.

Usage

```
geom_point(x, y)

geom_multipoint(x, y, id = 1)

geom_linestring(x, y, id = 1)

geom_polygon(x, y, id = 1, ring = 1)
```

Arguments

x	a vector of x coordinates
y	a vector of y coordinates
id	the feature identifier
ring	the id of the polygon ring

Value

an object of class rsgeo

Examples

```
geom_point(3, 0.14)
geom_multipoint(1:10, 10:1)
geom_linestring(1:10, 10:1)
geom_polygon(c(0, 1, 1, 0, 0), c(0, 0, 1, 1, 0))
```

---

haversine_destination *Identify a destination point*

Description

Given a vector of point geometries, bearings, and distances, identify a destination location.

Usage

```
haversine_destination(x, bearing, distance)
```

Arguments

x	an object of class rs_POINT
bearing	a numeric vector specifying the degree of the direction where 0 is north
distance	a numeric vector specifying the distance to travel in the direction specified by bearing in meters

Value

an object of class rs_POINT

Examples

```
# create 10 points at the origin
pnts <- geom_point(rep(0, 10), rep(0, 10))

# set seed for reproducibiliy
set.seed(1)

# generate random bearings
bearings <- runif(10, 0, 360)

# generate random distances
distances <- runif(10, 10000, 100000)

# find the destinations
dests <- haversine_destination(pnts, bearings, distances)

# plot points
if (rlang::is_installed(c("sf", "wk"))) {
  plot(pnts, pch = 3)
  plot(dests, add = TRUE, pch = 17)
}
```

`haversine_intermediate`

Identifies a point between two points

Description

Identifies the location between two points on a great circle along a specified fraction of the distance.

Usage

```
haversine_intermediate(x, y, distance)
```

Arguments

`x`	an `rs_POINT` vector
`y`	an `rs_POINT` vector
`distance`	a numeric vector of either length 1 or the same length as x and y

Value

an object of class `rs_POINT`

Examples

```
x <- geom_point(1:10, rep(5, 10))
y <- geom_point(1:10, rep(0, 10))
res <- haversine_intermediate(x, y, 0.5)
if (rlang::is_installed(c("wk", "sf"))) {
  plot(
    c(x, y, res),
    col = sort(rep.int(c("red", "blue", "purple"), 10)),
    pch = 16
  )
}
```

`intersects_sparse` *Binary Predicates*

Description

Functions to ascertain the binary relationship between two geometry vectors. Binary predicates are provided both pairwise as a sparse matrix.

Usage

```
intersects_sparse(x, y)

intersects_pairwise(x, y)

contains_sparse(x, y)

contains_pairwise(x, y)

within_sparse(x, y)

within_pairwise(x, y)
```

Arguments

`x`	an object of class `rsgeo`
`y`	an object of class `rsgeo`

Value

- For `_sparse` a list of integer vectors containing the position of the geometry in `y`
- For `_pairwise` a logical vector

Examples

```
if (rlang::is_installed("sf")) {
    nc <- sf::st_read(
      system.file("shape/nc.shp", package = "sf"),
      quiet = TRUE
    )

    x <- as_rsgeo(nc$geometry[1:5])
    y <- rev(x)

    # intersects
    intersects_sparse(x, y)
    intersects_pairwise(x, y)
    # contains
    contains_sparse(x, y)
    contains_pairwise(x, y)
    # within
    within_sparse(x, y)
    within_pairwise(x, y)
}
```

`is_convex` *Determine the Convexity of a LineString*

Description

For a given `rs_LINESTRING` vector, test its convexity. Convexity can be tested strictly or strongly, as well as based on winding.

Usage

```
is_convex(x)

is_ccw_convex(x)

is_cw_convex(x)

is_strictly_convex(x)

is_strictly_ccw_convex(x)

is_strictly_cw_convex(x)
```

Arguments

`x`	an object of class `rs_LINESTRING` See geo docs for further details

Value

a logical vector

Examples

```
lns <- geom_linestring(
    1:20,
    runif(20, -5, 5),
    rep.int(1:5, 4)
  )

is_convex(lns)
is_cw_convex(lns)
is_ccw_convex(lns)
is_strictly_convex(lns)
is_strictly_cw_convex(lns)
is_strictly_ccw_convex(lns)
```

---

length_euclidean *Calculate LineString Length*

Description

For a given LineString or MultiLineString geometry, calculate its length. Other geometries will return a value of NA.

Usage

```
length_euclidean(x)

length_geodesic(x)

length_vincenty(x)

length_haversine(x)
```

Arguments

x	an object of class rsgeo

Details

Notes:

- Vicenty, Geodesic, and Haversine methods will return in units of meters.
- Geodesic length will always converge and is more accurate than the Vicenty methods.
- Haversine uses a mean earth radius of 6371.088 km.

See geo docs for more details.

Value

A numeric vector

Examples

```
set.seed(0)
y <- runif(25, -5, 5)
x <- 1:25

ln <- geom_linestring(x, y)

length_euclidean(ln)
length_geodesic(ln)
length_vincenty(ln)
length_haversine(ln)
```

---

line_interpolate_point

Interpolate a Point on a LineString

Description

Finds the point that lies a given fraction along a line.

Usage

```
line_interpolate_point(x, fraction)
```

Arguments

x	an object of class rs_LINESTRING
fraction	a numeric vector of length 1 or the same length as x. Must be a value between 0 and 1 inclusive.

Value

An object of class rs_POINT

Examples

```
x <- geom_linestring(c(-1, 0, 0), c(0, 0, 1))
line_interpolate_point(x, 0.5)
```

line_segmentize *Segments a LineString into* n *equal length LineStrings*

Description

Given a LineString, segment it into n equal length LineStrings. The n LineStrings are provided as a MultiLineString which can be expanded using expand_geoms() and consequently flattened using flatten_geoms() if desired.

Usage

```
line_segmentize(x, n)
```

Arguments

x	and object of class rs_LINESTRING
n	an integer vector determining the number of equal length LineStrings to create

Value

A vector of class rs_MULTILINESTRING

Examples

```
x <- geom_linestring(1:10, runif(10, -1, 1))

segs <- line_segmentize(x, 3)

flatten_geoms(
  expand_geoms(segs)
)
```

locate_point_on_line *Locate a Point on a LineString*

Description

Calculates the fraction of a LineString's length to a point that is closes to a corresponding point in y.

Usage

```
locate_point_on_line(x, y)
```

Arguments

x	an object of class rs_LINESTRING
y	an object of class rs_POINT

Value

A numeric vector containing the fraction of of the LineString that would need to be traveled to reach the closest point.

Examples

```
x <- geom_linestring(c(-1, 0, 0), c(0, 0, 1))
y <- geom_point(-0.5, 0)
locate_point_on_line(x, y)
```

---

plot.rsgeo *Plot Geometries*

Description

Plot Geometries

Usage

```
## S3 method for class 'rsgeo'
plot(x, ...)
```

Arguments

x	an object of class rsgeo
...	arguments passed to wk::wk_plot()

Details

Plotting geometries utilizes wk::wk_plot(). The rust geometries are handled by first converting to an sfc object in the wk::wk_handle() method thus requiring both packages for plotting.

Value

Nothing.

Examples

```
if (rlang::is_installed(c("sf", "wk"))) {
  plot(geom_linestring(1:10, runif(10)))
}
```

`signed_area` *Calculate the area of a polygon*

Description

Functions to calculate different types of area for polygons.

Usage

```
signed_area(x)

unsigned_area(x)

signed_area_cd(x)

unsigned_area_cd(x)

signed_area_geodesic(x)

unsigned_area_geodesic(x)
```

Arguments

`x`	an object of class `rsgeo`

Details

- functions assume counter clock-wise winding in accordance with the simple feature access standard
- functions ending in `_cd` use the Chamberlain-Duquette algorithm for spherical area
- Chamberlain-Duquette and Geodesic areas are returned in meters squared and assume non-planar geometries

See geo docs for more:

- GeodesicArea
- Area
- ChamberlainDuquetteArea

Value

a numeric vector of the area contained by the geometry

Examples

```
x <- c(0, 1, 1, 0, 0)
y <- c(0, 0, 1, 1, 0)
p <- geom_polygon(x, y)

signed_area(p)
unsigned_area(p)
signed_area_cd(p)
unsigned_area_cd(p)
signed_area_geodesic(p)
unsigned_area_geodesic(p)
```

`simplify_geoms` *Simplify Geometry*

Description

Simplifies LineStrings, Polygons, and their Multi- counterparts.

Usage

```
simplify_geoms(x, epsilon)

simplify_vw_geoms(x, epsilon)

simplify_vw_preserve_geoms(x, epsilon)
```

Arguments

`x`	an object of class of `rsgeo`
`epsilon`	a tolerance parameter. Cannot be equal to or less than 0.

Details

Simplify functions use the Ramer–Douglas–Peucker algorithm. Functions with `vw` use the Visvalingam-Whyatt algorithm.

For more see geo docs.

Value

an object of class `rsgeo`

Examples

```
x <- geom_linestring(1:100, runif(100, 5, 10))

simplify_geoms(x, 3)
simplify_vw_geoms(x, 2)
simplify_vw_preserve_geoms(x, 100)
```

Index