Package 'geos'

November 30, 2023

Title Open Source Geometry Engine ('GEOS') R API

Version 0.2.4

Description Provides an R API to the Open Source Geometry Engine ('GEOS') library (<https://libgeos.org/>) and a vector format with which to efficiently store 'GEOS' geometries. High-performance functions to extract information from, calculate relationships between, and transform geometries are provided. Finally, facilities to import and export geometry vectors to other spatial formats are provided.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Suggests testthat (>= 3.0.0), vctrs, sf

Imports libgeos (>= 3.8.1-4), wk (>= 0.4.1)

URL https://paleolimbot.github.io/geos/, https://github.com/paleolimbot/geos/

BugReports https://github.com/paleolimbot/geos/issues

LinkingTo libgeos, wk

Config/testthat/edition 3

NeedsCompilation yes

Author Dewey Dunnington [aut, cre] (<https://orcid.org/0000-0002-9415-4582>), Edzer Pebesma [aut] (<https://orcid.org/0000-0001-8049-7069>)

Maintainer Dewey Dunnington <dewey@fishandwhistle.net>

Repository CRAN

Date/Publication 2023-11-30 05:50:03 UTC

R topics documented:

`as_geos_geometry.wk_xy` *Create GEOS Geometry Vectors*

Description

Create GEOS Geometry Vectors

Usage

```
## S3 method for class 'wk_xy'
as_geos_geometry(x, ...)

## S3 method for class 'wk_xyz'
as_geos_geometry(x, ...)

as_geos_geometry(x, ...)
```

```
## S3 method for class 'geos_geometry'
as_geos_geometry(x, ...)

## Default S3 method:
as_geos_geometry(x, ...)

## S3 method for class 'character'
as_geos_geometry(x, ..., crs = NULL)

## S3 method for class 'blob'
as_geos_geometry(x, ..., crs = NULL)

## S3 method for class 'WKB'
as_geos_geometry(x, ..., crs = NULL)

geos_geometry(crs = wk::wk_crs_inherit())
```

Arguments

x	An object to be coerced to a geometry vector
...	Unused
crs	An object that can be interpreted as a CRS. See wk::wk_crs().

Value

A geos geometry vector

Examples

```
as_geos_geometry("LINESTRING (0 1, 3 9)")
```

---

geos_area *Extract information from a GEOS geometry*

Description

Note that geos_x(), geos_y(), and geos_z() do not handle empty points (use geos_write_xy() if you need to handle this case). Similarly, the min/max functions will error on empty geometries.

Usage

```
geos_area(geom)

geos_length(geom)

geos_x(geom)
```

```
geos_y(geom)

geos_z(geom)

geos_xmin(geom)

geos_ymin(geom)

geos_xmax(geom)

geos_ymax(geom)

geos_minimum_clearance(geom)

geos_is_empty(geom)

geos_is_simple(geom)

geos_is_ring(geom)

geos_has_z(geom)

geos_is_closed(geom)

geos_type_id(geom)

geos_type(geom)

geos_precision(geom)

geos_srid(geom)

geos_num_coordinates(geom)

geos_num_geometries(geom)

geos_num_interior_rings(geom)

geos_num_rings(geom)

geos_dimension(geom)

geos_coordinate_dimension(geom)

geos_is_clockwise(geom)

geos_hilbert_code(geom, extent = wk::wk_bbox(geom), level = 15)
```

Arguments

geom	A GEOS geometry vector
extent	A geometry describing the extent of geom within which Hilbert codes should be computed. Defaults to wk::wk_bbox() of geom.
level	The Hilbert level of precision (between 0 and 15).

Value

A vector of length geom

Examples

```
geos_area("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))")
geos_length("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))")
geos_x("POINT Z (1 2 3)")
geos_y("POINT Z (1 2 3)")
geos_z("POINT Z (1 2 3)")
geos_xmin("LINESTRING (0 1, 2 3)")
geos_ymin("LINESTRING (0 1, 2 3)")
geos_xmax("LINESTRING (0 1, 2 3)")
geos_ymax("LINESTRING (0 1, 2 3)")
geos_minimum_clearance("POLYGON ((0 0, 10 0, 10 10, 3 5, 0 10, 0 0))")

geos_is_empty(c("POINT EMPTY", "POINT (0 1)"))
geos_is_simple(c("LINESTRING (0 0, 1 1)", "LINESTRING (0 0, 1 1, 1 0, 0 1)"))
geos_is_ring(
  c(
    "LINESTRING (0 0, 1 0, 1 1, 0 1, 0 0)",
    "LINESTRING (0 0, 1 0, 1 1, 0 1)"
   )
)
geos_is_closed(
  c(
    "LINESTRING (0 0, 1 0, 1 1, 0 1, 0 0)",
    "LINESTRING (0 0, 1 0, 1 1, 0 1)"
   )
)
geos_has_z(c("POINT Z (1 2 3)", "POINT (1 2)"))

geos_type_id(c("POINT (0 0)", "LINESTRING (0 0, 1 1)"))
geos_srid(wk::as_wkb(c("SRID=1234;POINT (0 0)", "POINT (0 0)")))
geos_num_coordinates(c("POINT (0 0)", "MULTIPOINT (0 0, 1 1)"))
geos_num_geometries(c("POINT (0 0)", "MULTIPOINT (0 0, 1 1)"))
geos_num_interior_rings("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))")
geos_dimension(c("POINT (0 0)", "LINESTRING (0 0, 1 1)"))
geos_coordinate_dimension(c("POINT (0 0)", "POINT Z (0 0 1)"))
```

geos_basic_strtree *Create a basic GEOS STRTree*

Description

An experimental alternative to the geos_strtree() that provides a more flexible interface and potentially faster spatial joins. Notably, geos_basic_strtree_insert() uses wk::wk_envelope() instead of as_geos_geometry() and does not keep the underlying geometry in memory. For object types like wk::xy() with an optimized wk::wk_envelope() method, this is very efficient.

Usage

```
geos_basic_strtree(items = NULL, node_capacity = 10L)

geos_basic_strtree_size(tree)

geos_basic_strtree_finalized(tree)

geos_basic_strtree_insert(tree, items)

geos_basic_strtree_query(tree, query, limit = NA, fill = FALSE)

geos_basic_strtree_query_filtered(
  tree,
  query,
  tree_geom,
  fun,
  ...,
  .chunk_size = 65536
)
```

Arguments

items	Items to add to the tree index
node_capacity	The maximum number of child nodes that a node may have. The minimum recommended capacity value is 4. If unsure, use a default node capacity of 10.
tree	A geos_basic_strtree()
query	Items with which to query the tree
limit	The maximum number of matches in the tree to return
fill	If TRUE, always returns limit matches per item in query padded with NA if fewer than limit matches are found.
tree_geom	A vctr coercible to geos_geometry() whose indices align with tree.
fun	A vectorized binary predicate (e.g. geos_intersects()) that will be called with the tree geometry, the query geometry and any ... args passed.
...	Passed to fun.
.chunk_size	The approximate number of comparisons to pass to fun.

Value

A geos_basic_strtree object

Examples

```
tree <- geos_basic_strtree(wk::xy(1:5, 1:5))
geos_basic_strtree_size(tree)
(geos_basic_strtree_insert(tree, wk::xy(6:10, 6:10)))
geos_basic_strtree_query(tree, as_geos_geometry("LINESTRING (3 0, 0 3)"))
```

---

geos_buffer *Buffer a geometry*

Description

- geos_buffer() returns a polygon or multipolygon geometry.
- geos_offset_curve() returns a linestring offset to the left by distance.

Usage

```
geos_buffer(geom, distance, params = geos_buffer_params())

geos_offset_curve(geom, distance, params = geos_buffer_params())

geos_buffer_params(
  quad_segs = 30,
  end_cap_style = c("round", "flat", "square"),
  join_style = c("round", "mitre", "bevel"),
  mitre_limit = 1,
  single_sided = FALSE
)
```

Arguments

geom	A GEOS geometry vector
distance	The buffer distance. Can be negative to buffer or offset on the righthand side of the geometry.
params	A geos_buffer_params()
quad_segs	The number of segments per quadrant. A higher number here will increase the apparent resolution of the resulting polygon.
end_cap_style	One of "round", "flat", or "square".
join_style	One of "round", "mitre", or "bevel".
mitre_limit	If join_style is "mitre", the relative extent (from zero to one) of the join.
single_sided	Use TRUE to buffer on only the right side of the geometry. This does not apply to geos_offset_curve(), which is always one-sided.

Value

A GEOS geometry vector along the recycled length of `geom` and `distance`.

Examples

```
geos_buffer("POINT (0 0)", 1)
geos_offset_curve("LINESTRING (0 0, 0 10, 10 0)", 1)
```

`geos_centroid` *Geometry transformers*

Description

Geometry transformers

Usage

```
geos_centroid(geom)

geos_boundary(geom)

geos_minimum_width(geom)

geos_minimum_clearance_line(geom)

geos_minimum_rotated_rectangle(geom)

geos_unary_union(geom)

geos_unary_union_prec(geom, grid_size)

geos_coverage_union(geom)

geos_point_on_surface(geom)

geos_node(geom)

geos_make_valid(geom, make_valid_params = geos_make_valid_params())

geos_make_valid_params(
  keep_collapsed = TRUE,
  method = c("make_valid_linework", "make_valid_structure")
)

geos_unique_points(geom)
```

```
geos_reverse(geom)

geos_merge_lines(geom)

geos_build_area(geom)

geos_envelope(geom)

geos_envelope_rct(geom)

geos_extent(geom)

geos_convex_hull(geom)

geos_concave_hull(geom, ratio, allow_holes = FALSE)

geos_concave_hull_of_polygons(
  geom,
  ratio,
  is_tight = TRUE,
  allow_holes = FALSE
)

geos_polygon_hull_simplify(
  geom,
  ratio,
  hull_type = c("outer", "inner"),
  ratio_mode = c("vertex", "area")
)

geos_point_start(geom)

geos_point_end(geom)

geos_line_merge(geom)

geos_line_merge_directed(geom)

geos_transform_xy(geom, trans)

geos_clone(geom)

geos_set_srid(geom, srid)

geos_point_n(geom, index)

geos_simplify(geom, tolerance)
```

```
geos_remove_repeated_points(geom, tolerance)

geos_simplify_preserve_topology(geom, tolerance)

geos_set_precision(
  geom,
  grid_size,
  preserve_topology = TRUE,
  keep_collapsed = FALSE
)

geos_normalize(geom)

geos_densify(geom, tolerance)

geos_clip_by_rect(geom, rect)
```

Arguments

geom	A GEOS geometry vector
grid_size	For _prec() variants, the grid size such that all vertices of the resulting geometry will lie on the grid.
make_valid_params	A geos_make_valid_params() object.
keep_collapsed	Should items that become EMPTY due to rounding be kept in the output?
method	The method to use for geos_make_valid(). One of: • "make_valid_linework" combines all rings into a set of noded lines and then extracts valid polygons from that linework. • "make_valid_structure" Structured method, first makes all rings valid then merges shells and subtracts holes from shells to generate valid result. Assumes that holes and shells are correctly categorized.
ratio	The normalized ratio between the shape of the concave hull and the area of the return value. Use 1 for the convex hull; use 0 for maximum concave-ness.
allow_holes	Use TRUE to allow the concave hull to contain holes
is_tight	Use FALSE to allow concave hull to expand beyond the convex hull.
hull_type	One of "outer" or "inner".
ratio_mode	One of "vertex" or "area", describing the normalized proportion type for which ratio represents.
trans	A wk transform object.
srid	An integer spatial reference identifier.
index	The index of the point or geometry to extract.
tolerance	A minimum distance to use for simplification or densification. Use a higher value for more simplification (or less densification).
preserve_topology	Should topology internal to each feature be preserved?

`rect` A `list()` representing rectangles in the form `list(xmin, ymin, xmax, ymax)`. List items with length 1 will be recycled to the length of the longest item.

Value

A GEOS geometry vector of length `geom`

Examples

```
geos_centroid(c("POINT (0 1)", "LINESTRING (0 0, 1 1)"))
geos_boundary(c("POLYGON ((0 0, 1 0, 0 1, 0 0))", "LINESTRING (0 0, 1 1)"))
geos_minimum_width("POLYGON ((0 0, 1 0, 0 1, 0 0))")
geos_minimum_clearance_line("POLYGON ((0 0, 10 0, 10 10, 3 5, 0 10, 0 0))")
geos_minimum_rotated_rectangle("POLYGON ((0 0, 1 0, 0.5 0.5, 0 0))")
geos_minimum_bounding_circle("LINESTRING (-1 -1, 1 1)")
geos_unary_union("MULTIPOINT (0 1, 0 1)")
geos_point_on_surface("LINESTRING (0 1, 0.2 3, 10 10)")
geos_node("POLYGON ((0 0, 1 0, 0 1, 0 0))")
geos_make_valid("POLYGON ((0 0, 1 1, 1 0, 0 1, 0 0))")
geos_unique_points("POLYGON ((0 0, 1 0, 0 1, 0 0))")
geos_reverse("LINESTRING (0 0, 1 1)")
geos_merge_lines(
  "MULTILINESTRING ((0 0, 0.5 0.5, 2 2), (0.5 0.5, 2 2))"
)
geos_build_area("LINESTRING (0 0, 1 0, 0 1, 0 0)")
geos_envelope("LINESTRING (0 0, 1 2)")
geos_convex_hull("MULTIPOINT (0 0, 1 0, 0 2, 0 0)")
geos_point_start("LINESTRING (0 0, 1 1)")
geos_point_end("LINESTRING (0 0, 1 1)")

geos_simplify("LINESTRING (0 0, 0 1, 0 2)", 0.1)
geos_simplify_preserve_topology("LINESTRING (0 0, 0 1, 0 2)", 0.1)
```

---

geos_create_rectangle *Create rectangles from bounds*

Description

Create rectangles from bounds

Usage

```
geos_create_rectangle(xmin, ymin, xmax, ymax, crs = NULL)
```

Arguments

xmin	Left bound of envelope
ymin	Lower bound of envelope
xmax	Right bound of envelope
ymax	Upper bound of envelope
crs	An object that can be interpreted as a CRS. See wk::wk_crs().

Value

A geos_geometry() consisting of a polygon

---

geos_delaunay_triangles

Delaunay triagulations and Voronoi diagrams

Description

These functions return one triangulation/diagram per feature as a multi geometry. These functions are not vectorized along their parameters.

Usage

```
geos_delaunay_triangles(geom, tolerance = 0)

geos_constrained_delaunay_triangles(geom)

geos_delaunay_edges(geom, tolerance = 0)

geos_voronoi_polygons(geom, env = NULL, tolerance = 0)

geos_voronoi_edges(geom, env = NULL, tolerance = 0)
```

Arguments

geom	A GEOS geometry vector whose nodes will be used as input.
tolerance	A snapping tolerance or 0 to disable snapping
env	A boundary for the diagram, or NULL to construct one based on the input

Value

A GEOS geometry vector of length geom

Examples

```
geos_delaunay_triangles("MULTIPOINT (0 0, 1 0, 0 1)")
geos_delaunay_edges("MULTIPOINT (0 0, 1 0, 0 1)")

geos_voronoi_polygons("MULTIPOINT (0 0, 1 0, 0 1)")
geos_voronoi_edges("MULTIPOINT (0 0, 1 0, 0 1)")
```

---

geos_disjoint *Binary predicates*

Description

Binary predicates

Usage

```
geos_disjoint(geom1, geom2)

geos_touches(geom1, geom2)

geos_intersects(geom1, geom2)

geos_crosses(geom1, geom2)

geos_within(geom1, geom2)

geos_contains(geom1, geom2)

geos_overlaps(geom1, geom2)

geos_equals(geom1, geom2)

geos_equals_exact(geom1, geom2, tolerance = .Machine$double.eps^2)

geos_covers(geom1, geom2)

geos_covered_by(geom1, geom2)

geos_prepared_disjoint(geom1, geom2)

geos_prepared_touches(geom1, geom2)

geos_prepared_intersects(geom1, geom2)

geos_prepared_crosses(geom1, geom2)
```

```
geos_prepared_within(geom1, geom2)

geos_prepared_contains(geom1, geom2)

geos_prepared_contains_properly(geom1, geom2)

geos_prepared_overlaps(geom1, geom2)

geos_prepared_covers(geom1, geom2)

geos_prepared_covered_by(geom1, geom2)
```

Arguments

geom1, geom2	GEOS geometry vectors, recycled to a common length.
tolerance	The maximum separation of vertices that should be considered equal.

Value

A logical vector along the recycled length of geom1 and geom2

`geos_disjoint_matrix` *Matrix predicates*

Description

Matrix predicates

Usage

```
geos_disjoint_matrix(geom, tree)

geos_touches_matrix(geom, tree)

geos_intersects_matrix(geom, tree)

geos_crosses_matrix(geom, tree)

geos_within_matrix(geom, tree)

geos_contains_matrix(geom, tree)

geos_contains_properly_matrix(geom, tree)

geos_overlaps_matrix(geom, tree)

geos_equals_matrix(geom, tree)
```

```
geos_equals_exact_matrix(geom, tree, tolerance = .Machine$double.eps^2)

geos_covers_matrix(geom, tree)

geos_covered_by_matrix(geom, tree)

geos_disjoint_any(geom, tree)

geos_touches_any(geom, tree)

geos_intersects_any(geom, tree)

geos_crosses_any(geom, tree)

geos_within_any(geom, tree)

geos_contains_any(geom, tree)

geos_contains_properly_any(geom, tree)

geos_overlaps_any(geom, tree)

geos_equals_any(geom, tree)

geos_equals_exact_any(geom, tree, tolerance = .Machine$double.eps^2)

geos_covers_any(geom, tree)

geos_covered_by_any(geom, tree)
```

Arguments

geom	A GEOS geometry vector
tree	A geos_strtree()
tolerance	The maximum separation of vertices that should be considered equal.

Value

A list() of integer vectors containing the indices of tree for which the predicate would return TRUE.

---

geos_distance *Distance calculations*

Description

Distance calculations

Usage

```
geos_distance(geom1, geom2)

geos_prepared_distance(geom1, geom2)

geos_distance_indexed(geom1, geom2)

geos_distance_hausdorff(geom1, geom2, densify = NULL)

geos_distance_frechet(geom1, geom2, densify = NULL)

geos_is_within_distance(geom1, geom2, distance)

geos_prepared_is_within_distance(geom1, geom2, distance)
```

Arguments

geom1, geom2	GEOS geometry vectors, recycled to a common length.
densify	A fraction between 0 and 1 denoting the degree to which edges should be subdivided (smaller value means more subdivisions). Use NULL to calculate the distance as-is.
distance	A threshold distance, below which geos_is_within_distance() and geos_prepared_is_within_dist will return TRUE.

Value

A numeric vector along the recycled length of geom1 and geom2

---

geos_empty *Create empty geometries*

Description

Create empty geometries

Usage

```
geos_empty(type_id = "geometrycollection", crs = wk::wk_crs_inherit())

as_geos_type_id(type_id)

## Default S3 method:
```

```
as_geos_type_id(type_id)

## S3 method for class 'character'
as_geos_type_id(type_id)

## S3 method for class 'numeric'
as_geos_type_id(type_id)
```

Arguments

type_id	The numeric type identifier for which an empty should be returned, an object from which one can be extracted using as_geos_type_id() (default to calling geos_type_id()). This is most usefully a character vector with the geometry type (e.g., point, linestring, polygon).
crs	An object that can be interpreted as a CRS. See wk::wk_crs().

Value

A GEOS geometry vector.

Examples

```
geos_empty(c("point", "linestring", "polygon"))
geos_empty(1:7)
geos_empty(geos_read_wkt(c("POINT (0 1)", "LINESTRING (0 0, 1 1)")))
```

---

geos_geometry_n *Access child geometries*

Description

Access child geometries

Usage

```
geos_geometry_n(geom, n)

geos_ring_n(geom, n)
```

Arguments

geom	A GEOS geometry vector
n	The (one-based) index of the child geometry

Value

A GEOS geometry vector along the recycled length of geom and i.

Examples

```
multipoint <- "MULTIPOINT (0 0, 1 1, 2 2)"
geos_geometry_n(multipoint, seq_len(geos_num_geometries(multipoint)))

poly <- "POLYGON ((0 0, 0 1, 1 0, 0 0), (0.1 0.1, 0.1 0.2, 0.2 0.1, 0.1 0.1))"
geos_ring_n(poly, seq_len(geos_num_rings(poly)))
```

---

geos_inner_join *Generate inner join keys based on a GEOS predicate*

Description

Experimental low-level spatial join infrastructure based on the `geos_basic_strtree()`.

Usage

```
geos_inner_join(
  x,
  y,
  predicate = "intersects",
  distance = NA,
  suffix = c(".x", ".y")
)

geos_inner_join_keys(x, y, predicate = "intersects", distance = NA)
```

Arguments

`x, y`	Geometry vectors with a `wk::wk_handle()` method.
`predicate`	One of: • intersects • contains • contains_properly • covered_by • covers • crosses • equals • equals_exact • intersects • within_distance • overlaps • touches
`distance`	Passed to `geos_is_within_distance()` when `predicate` is "within_distance"; passed to `geos_equals_exact()` when `predicate` is "equals_exact.
`suffix`	A character vector of length 2 with suffixes for the left and right sides for output columns with duplicated names.

Value

A data.frame with columns x and y corresponding to the 1-based indices on pairs of x and y for which predicate is TRUE.

Examples

```
x <- data.frame(
  col_x = "a",
  geometry = as_geos_geometry("POINT (10 10)")
)

y <- data.frame(
  col_y = "a",
  geometry = as_geos_geometry("POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))")
)

geos_inner_join(x, y, "intersects")

geos_inner_join_keys(
  "POINT (5 5)",
  "POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))",
  "intersects"
)
```

---

geos_intersection *Binary geometry operators*

Description

- geos_intersection() returns the set of points common to both x and y.
- geos_difference() returns the set of points from x that are not contained by y.
- geos_sym_difference() returns the set of points that are *not* common to x and y.
- geos_union() returns the set of points contained by either x or y.
- geos_shared_paths() returns a GEOMETRYCOLLECTION containing two MULTILINESTRINGS: the first containing paths in the same direction, the second containing common paths in the opposite direction.
- geos_snap() snaps the vertices of x within tolerance of y to y.

Usage

```
geos_intersection(geom1, geom2)

geos_difference(geom1, geom2)

geos_sym_difference(geom1, geom2)
```

```
geos_union(geom1, geom2)

geos_intersection_prec(geom1, geom2, grid_size)

geos_difference_prec(geom1, geom2, grid_size)

geos_sym_difference_prec(geom1, geom2, grid_size)

geos_union_prec(geom1, geom2, grid_size)

geos_shared_paths(geom1, geom2)

geos_snap(geom1, geom2, tolerance = .Machine$double.eps^2)

geos_clearance_line_between(geom1, geom2, prepare = FALSE)
```

Arguments

geom1, geom2	GEOS geometry vectors, recycled to a common length.
grid_size	For _prec() variants, the grid size such that all vertices of the resulting geometry will lie on the grid.
tolerance	The maximum separation of vertices that should be considered equal.
prepare	Use prepared geometries to calculate clearance line

Value

A GEOS geometry vector along the recycled length of geom1 and geom2.

Examples

```
poly1 <- "POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))"
poly2 <- "POLYGON ((5 5, 5 15, 15 15, 15 5, 5 5))"

geos_intersection(poly1, poly2)
geos_difference(poly1, poly2)
geos_sym_difference(poly1, poly2)
geos_union(poly1, poly2)

line <- "LINESTRING (11 0, 11 10)"
geos_snap(poly1, line, tolerance = 2)

geos_shared_paths("LINESTRING (0 0, 1 1, 2 2)", "LINESTRING (3 3, 2 2, 1 1)")
```

geos_is_valid *Geometry validity*

Description

- `geos_is_valid()` returns a logical vector denoting if each feature is a valid geometry.
- `geos_is_valid_detail()` returns a data frame with columns `is_valid` (logical), `reason` (character), and `location` (geos_geometry).

Usage

```
geos_is_valid(geom)

geos_is_valid_detail(geom, allow_self_touching_ring_forming_hole = FALSE)
```

Arguments

`geom`	A GEOS geometry vector
`allow_self_touching_ring_forming_hole`	It's all in the name

Examples

```
geos_is_valid(
  c(
    "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
    "POLYGON ((0 0, 1 1, 1 0, 0 1, 0 0))"
  )
)

geos_is_valid_detail(
  c(
    "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
    "POLYGON ((0 0, 1 1, 1 0, 0 1, 0 0))"
  )
)
```

geos_largest_empty_circle_spec *Circular approximations*

Description

Circular approximations

Usage

```
geos_largest_empty_circle_spec(geom, boundary, tolerance)

geos_largest_empty_crc(geom, boundary, tolerance)

geos_minimum_bounding_circle(geom)

geos_minimum_bounding_crc(geom)

geos_maximum_inscribed_circle_spec(geom, tolerance)

geos_maximum_inscribed_crc(geom, tolerance)
```

Arguments

geom	A GEOS geometry vector
boundary	An outer boundary for the largest empty circle algorithm.
tolerance	Threshold for considering circles to be touching a boundary.

---

geos_make_point *Create geometries from vectors of coordinates*

Description

These functions transform raw coordinates into point, line, polygon, features, or nest a vector of geometries into a MULTI* type or GEOMETRYCOLLECTION. See `wk::wk_coords()`, `geos_unnest()`, or `wk::wk_flatten()` to perform inverse operations; see `wk::xy()`, `wk::wk_linestring()`, `wk::wk_polygon()`, or `wk::wk_collection()` for generic versions that work with non-GEOS types.

Usage

```
geos_make_point(x, y, z = NA_real_, crs = NULL)

geos_make_linestring(x, y, z = NA_real_, feature_id = 1L, crs = NULL)

geos_make_polygon(
  x,
  y,
  z = NA_real_,
  feature_id = 1L,
  ring_id = 1L,
  crs = NULL
)

geos_make_collection(geom, type_id = "geometrycollection", feature_id = 1L)
```

Arguments

x, y, z	Vectors of coordinate values
crs	An object that can be interpreted as a CRS. See wk::wk_crs().
feature_id, ring_id	Vectors for which a change in sequential values indicates a new feature or ring. Use factor() to convert from a character vector.
geom	A GEOS geometry vector
type_id	The numeric type identifier for which an empty should be returned, an object from which one can be extracted using as_geos_type_id() (default to calling geos_type_id()). This is most usefully a character vector with the geometry type (e.g., point, linestring, polygon).

Value

A GEOS geometry vector

Examples

```
geos_make_point(1:3, 1:3)
geos_make_linestring(1:3, 1:3)
geos_make_polygon(c(0, 1, 0), c(0, 0, 1))
geos_make_collection("POINT (1 1)")
```

---

geos_nearest *Find the closest feature*

Description

Finds the closest item index in tree to geom, vectorized along geom.

Usage

```
geos_nearest(geom, tree)

geos_nearest_indexed(geom, tree)

geos_nearest_hausdorff(geom, tree, densify = NULL)

geos_nearest_frechet(geom, tree, densify = NULL)
```

Arguments

geom	A GEOS geometry vector
tree	A geos_strtree()
densify	A fraction between 0 and 1 denoting the degree to which edges should be subdivided (smaller value means more subdivisions). Use NULL to calculate the distance as-is.

Value

An integer vector of length `geom` containing the index of `tree` that is closest to each feature in `geom`.

`geos_polygonize` *Create polygons from noded edges*

Description

Create polygons from noded edges

Usage

```
geos_polygonize(collection)

geos_polygonize_valid(collection)

geos_polygonize_cut_edges(collection)

geos_polygonize_full(collection)
```

Arguments

`collection`	A GEOMETRYCOLLECTION or MULTILINESTRING of edges that meet at their endpoints.

Value

A GEOMETRYCOLLECTION of polygons

Examples

```
geos_polygonize("MULTILINESTRING ((0 0, 0 1), (0 1, 1 0), (1 0, 0 0))")
geos_polygonize_valid("MULTILINESTRING ((0 0, 0 1), (0 1, 1 0), (1 0, 0 0))")
geos_polygonize_cut_edges("MULTILINESTRING ((0 0, 0 1), (0 1, 1 0), (1 0, 0 0))")
```

geos_project *Linear referencing*

Description

- geos_project() and geos_project_normalized() return the distance of point geom2 projected on geom1 from the origin of geom1, which must be a lineal geometry.
- geos_interpolate() performs an inverse operation, returning the point along geom representing the given distance from the origin along the geometry.
- _normalized() variants use a distance normalized to the geos_length() of the geometry.

Usage

```
geos_project(geom1, geom2)

geos_project_normalized(geom1, geom2)

geos_interpolate(geom, distance)

geos_interpolate_normalized(geom, distance_normalized)
```

Arguments

geom1, geom2	GEOS geometry vectors, recycled to a common length.
geom	A GEOS geometry vector
distance	Distance along the linestring to interpolate
distance_normalized	Distance along the linestring to interpolate relative to the length of the linestring.

Examples

```
geos_interpolate("LINESTRING (0 0, 1 1)", 1)
geos_interpolate_normalized("LINESTRING (0 0, 1 1)", 1)

geos_project("LINESTRING (0 0, 10 10)", "POINT (5 5)")
geos_project_normalized("LINESTRING (0 0, 10 10)", "POINT (5 5)")
```

geos_read_wkt *Read and write well-known text*

Description

Read and write well-known text

Usage

```
geos_read_wkt(wkt, fix_structure = FALSE, crs = NULL)

geos_write_wkt(geom, include_z = TRUE, precision = 16, trim = TRUE)

geos_read_geojson(geojson, crs = NULL)

geos_write_geojson(geom, indent = -1)

geos_read_wkb(wkb, fix_structure = FALSE, crs = NULL)

geos_write_wkb(
  geom,
  include_z = TRUE,
  include_srid = FALSE,
  endian = 1,
  flavor = c("extended", "iso")
)

geos_read_hex(hex, fix_structure = FALSE, crs = NULL)

geos_write_hex(
  geom,
  include_z = TRUE,
  include_srid = FALSE,
  endian = 1,
  flavor = c("extended", "iso")
)

geos_read_xy(point)

geos_write_xy(geom)
```

Arguments

wkt	a character() vector of well-known text
fix_structure	Set the reader to automatically repair structural errors in the input (currently just unclosed rings) while reading.

crs	An object that can be interpreted as a CRS. See wk::wk_crs().
geom	A GEOS geometry vector
include_z, include_srid	Include the values of the Z and M coordinates and/or SRID in the output? Use FALSE to omit, TRUE to include, or NA to include only if present. Note that using TRUE may result in an error if there is no value present in the original.
precision	The number of significant digits to include iin WKT output.
trim	Trim unnecessary zeroes in the output?
geojson	A character() vector fo GeoJSON features
indent	The number of spaces to use when indenting a formatted version of the output. Use -1 to indicate no formatting.
wkb	A list() of raw() vectors (or NULL representing an NA value).
endian	0 for big endian or 1 for little endian.
flavor	One of "extended" (i.e., EWKB) or "iso".
hex	A hexidecimal representation of well-known binary
point	A list() representing points in the form list(x, y).

Examples

```
geos_read_wkt("POINT (30 10)")
geos_write_wkt(geos_read_wkt("POINT (30 10)"))
```

---

geos_relate *Dimensionally extended 9 intersection model*

Description

See the Wikipedia entry on DE-9IM for how to interpret pattern, match, and the result of geos_relate() and/or geos_relate_pattern_create().

Usage

```
geos_relate(geom1, geom2, boundary_node_rule = "mod2")

geos_relate_pattern(geom1, geom2, pattern, boundary_node_rule = "mod2")

geos_relate_pattern_match(match, pattern)

geos_relate_pattern_create(
  II = "*",
  IB = "*",
  IE = "*",
  BI = "*",
```

```
  BB = "*",
  BE = "*",
  EI = "*",
  EB = "*",
  EE = "*"
)
```

Arguments

geom1, geom2	GEOS geometry vectors, recycled to a common length.
boundary_node_rule	One of "mod2", "endpoint", "multivalent_endpoint", or "monovalent_endpoint".
pattern, match	A character vector representing the match
II, IB, IE, BI, BB, BE, EI, EB, EE	One of "0", "1", "2", "T", "F", or "*" describing the dimension of the intersection between the interior (I), boundary (B), and exterior (E).

Examples

```
geos_relate_pattern_match("FF*FF1***", c(NA, "FF*FF****", "FF*FF***F"))
geos_relate("POINT (0 0)", "POINT (0 0)")
geos_relate_pattern("POINT (0 0)", "POINT (0 0)", "T********")
geos_relate_pattern_create(II = "T")
```

---

geos_segment_intersection

Segment operations

Description

Segment operations

Usage

```
geos_segment_intersection(a, b)

geos_orientation_index(a, point)
```

Arguments

a, b	A list() representing segments in the form list(x0, y0, x1, y1). List items with length 1 will be recycled to the length of the longest item.
point	A list() representing points in the form list(x, y).

Value

`geos_segment_intersection()` returns a `list(x, y)`; `geos_orientation_index()` returns -1, 0 or 1, depending if the `point` lies to the right of (-1), is colinear with (0) or lies to the left of (1) the segment (as judged from the start of the segment looking towards the end).

Examples

```
geos_segment_intersection(
  list(0, 0, 10, 10),
  list(10, 0, 0, 10)
)

geos_orientation_index(
  list(0, 0, 10, 10),
  list(15, c(12, 15, 17))
)
```

---

geos_strtree *Create a GEOS STRTree*

Description

Create a GEOS STRTree

Usage

```
geos_strtree(geom, node_capacity = 10L)

geos_strtree_query(tree, geom)

geos_strtree_data(tree)

as_geos_strtree(x, ...)

## Default S3 method:
as_geos_strtree(x, ...)

## S3 method for class 'geos_strtree'
as_geos_strtree(x, ...)

## S3 method for class 'geos_geometry'
as_geos_strtree(x, ...)
```

Arguments

geom	A GEOS geometry vector
node_capacity	The maximum number of child nodes that a node may have. The minimum recommended capacity value is 4. If unsure, use a default node capacity of 10.
tree	A geos_strtree()
x	An object to convert to a geos_strtree()
...	Unused

Value

A geos_str_tree object

---

geos_unnest *Unnest nested geometries*

Description

Ths function flattens nested geometries (i.e., multi or geometrycollection types) into a vector with the same or fewer levels of nesting. See geos_geometry_n() to access individual geometries within a collection; see wk::wk_flatten() for a version of this function that works with non-GEOS geometries; see geos_make_collection() and wk::wk_collection() for functions that perform the inverse operation.

Usage

```
geos_unnest(geom, keep_empty = FALSE, keep_multi = TRUE, max_depth = 1)
```

Arguments

geom	A GEOS geometry vector
keep_empty	If TRUE, EMPTY geomtries are left as-is rather than collapsing to length 0.
keep_multi	If TRUE, MULTI* geometries are not expanded to sub-features (i.e., only GEOMETRYCOLLECTIONs are).
max_depth	The maximum recursive GEOMETRYCOLLECTION depth to unnest.

Value

A GEOS geometry vector, with a length greater than or equal to geom with an attribute "lengths" that can be used to map elements of the result to the original item.

Examples

```
geos_unnest("GEOMETRYCOLLECTION (POINT (1 2), POINT (3 4))")
```

geos_version *GEOS version information*

Description

GEOS version information

Usage

```
geos_version(runtime = TRUE)
```

Arguments

runtime	Use FALSE to return the build-time GEOS version, which may be different than the runtime version if a different version of the libgeos package was used to build this package.

Examples

```
geos_version()
geos_version(runtime = FALSE)

# check for a minimum version of GEOS
geos_version() >= "3.8.1"
```

plot.geos_geometry *Plot GEOS geometries*

Description

Plot GEOS geometries

Usage

```
## S3 method for class 'geos_geometry'
plot(
  x,
  ...,
  asp = 1,
  bbox = NULL,
  xlab = "",
  ylab = "",
  rule = "evenodd",
  add = FALSE,
  simplify = 1,
  crop = TRUE
)
```

Arguments

x	A GEOS geometry vector
...	Passed to plotting functions for features: graphics::points() for point and multipoint geometries, graphics::lines() for linestring and multilinestring geometries, and graphics::polypath() for polygon and multipolygon geometries.
asp, xlab, ylab	Passed to graphics::plot()
bbox	The limits of the plot as a rct() or compatible object
rule	The rule to use for filling polygons (see graphics::polypath())
add	Should a new plot be created, or should handleable be added to the existing plot?
simplify	A relative tolerance to use for simplification of geometries. Use 0 to disable simplification; use a higher number to make simplification coarser.
crop	Use TRUE to crop the input to the extent of the plot.

Value

The input, invisibly

Examples

```
plot(as_geos_geometry("LINESTRING (0 0, 1 1)"))
plot(as_geos_geometry("POINT (0.5 0.4)"), add = TRUE)
```

---

vctrs-methods *Vctrs methods*

Description

Vctrs methods

Usage

```
vec_cast.geos_geometry(x, to, ...)

vec_ptype2.geos_geometry(x, y, ...)
```

Arguments

x, y, to, ...	See vctrs::vec_cast() and vctrs::vec_ptype2().

wk-methods *Compatibility with the wk package*

Description

Compatibility with the wk package

Usage

```
## S3 method for class 'geos_geometry'
wk_handle(handleable, handler, ...)

geos_geometry_writer()

## S3 method for class 'geos_geometry'
wk_writer(handleable, ...)
```

Arguments

handleable	A geometry vector (e.g., wkb(), wkt(), xy(), rct(), or sf::st_sfc()) for which wk_handle() is defined.
handler	A wk_handler object.
...	Passed to the wk_handle() method.

Value

The result of the handler

Examples

```
library(wk)
wk_handle(as_geos_geometry("POINT (1 2)"), wk::wkt_writer())
```

Index