

# Package ‘geos’

June 1, 2023

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

**Version** 0.2.3

**Description** Provides an R API to the Open Source Geometry Engine ('GEOS') library ([<https://libgeos.org/>](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** 2023-05-31 22:40:02 UTC

## R topics documented:

as_geos_geometry.wk_xy . . . . .	2
geos_area . . . . .	3

geos_basic_strtree . . . . .	6
geos_buffer . . . . .	7
geos_centroid . . . . .	8
geos_create_rectangle . . . . .	11
geos_delaunay_triangles . . . . .	12
geos_disjoint . . . . .	13
geos_disjoint_matrix . . . . .	14
geos_distance . . . . .	15
geos_empty . . . . .	16
geos_geometry_n . . . . .	17
geos_inner_join . . . . .	18
geos_intersection . . . . .	19
geos_is_valid . . . . .	21
geos_largest_empty_circle_spec . . . . .	21
geos_make_point . . . . .	22
geos_nearest . . . . .	23
geos_polygonize . . . . .	24
geos_project . . . . .	25
geos_read_wkt . . . . .	26
geos_relate . . . . .	27
geos_segment_intersection . . . . .	28
geos_strtree . . . . .	29
geos_unnest . . . . .	30
geos_version . . . . .	31
plot.geos_geometry . . . . .	31
vctrs-methods . . . . .	32
wk-methods . . . . .	33

## Index 34

---

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 <a href="#">wk::wk_crs()</a> .

### 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 <a href="#">GEOS geometry vector</a>
extent	A geometry describing the extent of geom within which Hilbert codes should be computed. Defaults to <code>wk::wk_bbox()</code> 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

<code>items</code>	Items to add to the tree index
<code>node_capacity</code>	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.
<code>tree</code>	A <code>geos_basic_strtree()</code>
<code>query</code>	Items with which to query the tree
<code>limit</code>	The maximum number of matches in the tree to return
<code>fill</code>	If TRUE, always returns <code>limit</code> matches per item in query padded with NA if fewer than <code>limit</code> matches are found.
<code>tree_geom</code>	A vctr coercible to <code>geos_geometry()</code> whose indices align with <code>tree</code> .
<code>fun</code>	A vectorized binary predicate (e.g. <code>geos_intersects()</code> ) that will be called with the tree geometry, the query geometry and any ... args passed.
<code>...</code>	Passed to <code>fun</code> .
<code>.chunk_size</code>	The approximate number of comparisons to pass to <code>fun</code> .

**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**

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

---

**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 <a href="#">GEOS geometry vector</a>
grid_size	For <code>_prec()</code> variants, the grid size such that all vertices of the resulting geometry will lie on the grid.
make_valid_params	A <a href="#">geos_make_valid_params()</a> object.
keep_collapsed	Should items that become EMPTY due to rounding be kept in the output?
method	The method to use for <a href="#">geos_make_valid()</a> . One of: <ul style="list-style-type: none"> <li>"make_valid_linework" combines all rings into a set of noded lines and then extracts valid polygons from that linework.</li> <li>"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.</li> </ul>
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 <a href="#">wk transform</a> 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 <a href="#">wk::wk_crs()</a> .

**Value**

A [geos\\_geometry\(\)](#) consisting of a polygon

---

geos\_delaunay\_triangles

*Delaunay triangulations 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 <a href="#">GEOS geometry vector</a> 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	<i>Binary predicates</i>
---------------	--------------------------

---

**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 <a href="#">GEOS geometry vector</a>
tree	A <a href="#">geos_strtree()</a>
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.

**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	<a href="#">GEOS geometry vectors</a> , 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 <a href="#">geos_is_within_distance()</a> and <a href="#">geos_prepared_is_within_distance()</a> will return TRUE.

**Value**

A numeric vector along the recycled length of geom1 and geom2

---

geos_empty	<i>Create empty geometries</i>
------------	--------------------------------

---

**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	<i>Access child geometries</i>
-----------------	--------------------------------

---

### 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 <code>wk::wk_handle()</code> method.
predicate	One of: <ul style="list-style-type: none"> <li>• intersects</li> <li>• contains</li> <li>• contains_properly</li> <li>• covered_by</li> <li>• covers</li> <li>• crosses</li> <li>• equals</li> <li>• equals_exact</li> <li>• intersects</li> <li>• within_distance</li> <li>• overlaps</li> <li>• touches</li> </ul>
distance	Passed to <code>geos_is_within_distance()</code> when predicate is "within_distance"; passed to <code>geos_equals_exact()</code> 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	<a href="#">GEOS geometry vectors</a> , recycled to a common length.
grid_size	For <code>_prec()</code> 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	<i>Geometry validity</i>
---------------	--------------------------

---

**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	<i>Circular approximations</i>
--------------------------------	--------------------------------

---

**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 <a href="#">GEOS geometry vector</a>
boundary	An outer boundary for the largest empty circle algorithm.
tolerance	Threshold for considering circles to be touching a boundary.

---

geos_make_point	<i>Create geometries from vectors of coordinates</i>
-----------------	--

---

**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**

<code>x, y, z</code>	Vectors of coordinate values
<code>crs</code>	An object that can be interpreted as a CRS. See <a href="#">wk::wk_crs()</a> .
<code>feature_id, ring_id</code>	Vectors for which a change in sequential values indicates a new feature or ring. Use <a href="#">factor()</a> to convert from a character vector.
<code>geom</code>	A <a href="#">GEOS geometry vector</a>
<code>type_id</code>	The numeric type identifier for which an empty should be returned, an object from which one can be extracted using <a href="#">as_geos_type_id()</a> (default to calling <a href="#">geos_type_id()</a> ). 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)")
```

---

<code>geos_nearest</code>	<i>Find the closest feature</i>
---------------------------	---------------------------------

---

**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**

<code>geom</code>	A <a href="#">GEOS geometry vector</a>
<code>tree</code>	A <a href="#">geos_strtree()</a>
<code>densify</code>	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	<i>Create polygons from noded edges</i>
-----------------	---

---

**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	<i>Linear referencing</i>
--------------	---------------------------

---

## 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

<code>geom1, geom2</code>	GEOS geometry vectors, recycled to a common length.
<code>geom</code>	A GEOS geometry vector
<code>distance</code>	Distance along the linestring to interpolate
<code>distance_normalized</code>	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 <code>wk::wk_crs()</code> .
geom	A <a href="#">GEOS geometry vector</a>
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 in WKT output.
trim	Trim unnecessary zeroes in the output?
geojson	A <code>character()</code> vector for 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 <code>list()</code> of <code>raw()</code> 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 hexadecimal representation of well-known binary
point	A <code>list()</code> representing points in the form <code>list(x, y)</code> .

### 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 <a href="#">GEOS geometry vector</a>
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 <a href="#">geos_strtree()</a>
x	An object to convert to a <a href="#">geos_strtree()</a>
...	Unused

**Value**

A `geos_str_tree` object

---

geos_unnest	<i>Unnest nested geometries</i>
-------------	---------------------------------

---

**Description**

This 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 <a href="#">GEOS geometry vector</a>
keep_empty	If TRUE, EMPTY geometries 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	<i>GEOS version information</i>
--------------	---------------------------------

---

**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 <a href="#">libgeos package</a> 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	<i>Plot GEOS geometries</i>
--------------------	-----------------------------

---

**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**

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



**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., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined.
handler	A <code>wk_handler</code> object.
...	Passed to the <code>wk_handle()</code> method.

**Value**

The result of the handler

**Examples**

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

# Index

`as_geos_geometry`  
    (`as_geos_geometry.wk_xy`), 2  
`as_geos_geometry()`, 6  
`as_geos_geometry.wk_xy`, 2  
`as_geos_strtree` (`geos_strtree`), 29  
`as_geos_type_id` (`geos_empty`), 16  
`as_geos_type_id()`, 17, 23

`factor()`, 23

GEOS geometry vector, 5, 7, 8, 10–12, 15, 17, 20–23, 25, 27, 30, 32  
GEOS geometry vectors, 14, 16, 20, 25, 28  
`geos_area`, 3  
`geos_basic_strtree`, 6  
`geos_basic_strtree()`, 6, 18  
`geos_basic_strtree_finalized`  
    (`geos_basic_strtree`), 6  
`geos_basic_strtree_insert`  
    (`geos_basic_strtree`), 6  
`geos_basic_strtree_insert()`, 6  
`geos_basic_strtree_query`  
    (`geos_basic_strtree`), 6  
`geos_basic_strtree_query_filtered`  
    (`geos_basic_strtree`), 6  
`geos_basic_strtree_size`  
    (`geos_basic_strtree`), 6  
`geos_boundary` (`geos_centroid`), 8  
`geos_buffer`, 7  
`geos_buffer()`, 7  
`geos_buffer_params` (`geos_buffer`), 7  
`geos_buffer_params()`, 7  
`geos_build_area` (`geos_centroid`), 8  
`geos_centroid`, 8  
`geos_clearance_line_between`  
    (`geos_intersection`), 19  
`geos_clip_by_rect` (`geos_centroid`), 8  
`geos_clone` (`geos_centroid`), 8  
`geos_concave_hull` (`geos_centroid`), 8  
`geos_concave_hull_of_polygons`  
    (`geos_centroid`), 8  
`geos_constrained_delaunay_triangles`  
    (`geos_delaunay_triangles`), 12  
`geos_contains` (`geos_disjoint`), 13  
`geos_contains_any`  
    (`geos_disjoint_matrix`), 14  
`geos_contains_matrix`  
    (`geos_disjoint_matrix`), 14  
`geos_contains_properly_any`  
    (`geos_disjoint_matrix`), 14  
`geos_contains_properly_matrix`  
    (`geos_disjoint_matrix`), 14  
`geos_convex_hull` (`geos_centroid`), 8  
`geos_coordinate_dimension` (`geos_area`), 3  
`geos_coverage_union` (`geos_centroid`), 8  
`geos_covered_by` (`geos_disjoint`), 13  
`geos_covered_by_any`  
    (`geos_disjoint_matrix`), 14  
`geos_covered_by_matrix`  
    (`geos_disjoint_matrix`), 14  
`geos_covers` (`geos_disjoint`), 13  
`geos_covers_any` (`geos_disjoint_matrix`), 14  
`geos_covers_matrix`  
    (`geos_disjoint_matrix`), 14  
`geos_create_rectangle`, 11  
`geos_crosses` (`geos_disjoint`), 13  
`geos_crosses_any`  
    (`geos_disjoint_matrix`), 14  
`geos_crosses_matrix`  
    (`geos_disjoint_matrix`), 14  
`geos_delaunay_edges`  
    (`geos_delaunay_triangles`), 12  
`geos_delaunay_triangles`, 12  
`geos_densify` (`geos_centroid`), 8  
`geos_difference` (`geos_intersection`), 19  
`geos_difference()`, 19  
`geos_difference_prec`

- (geos\_intersection), 19
- geos\_dimension (geos\_area), 3
- geos\_disjoint, 13
- geos\_disjoint\_any
  - (geos\_disjoint\_matrix), 14
- geos\_disjoint\_matrix, 14
- geos\_distance, 15
- geos\_distance\_frechet (geos\_distance), 15
- geos\_distance\_hausdorff
  - (geos\_distance), 15
- geos\_distance\_indexed (geos\_distance), 15
- geos\_empty, 16
- geos\_envelope (geos\_centroid), 8
- geos\_envelope\_rct (geos\_centroid), 8
- geos\_equals (geos\_disjoint), 13
- geos\_equals\_any (geos\_disjoint\_matrix), 14
- geos\_equals\_exact (geos\_disjoint), 13
- geos\_equals\_exact(), 18
- geos\_equals\_exact\_any
  - (geos\_disjoint\_matrix), 14
- geos\_equals\_exact\_matrix
  - (geos\_disjoint\_matrix), 14
- geos\_equals\_matrix
  - (geos\_disjoint\_matrix), 14
- geos\_extent (geos\_centroid), 8
- geos\_geometry, 21
- geos\_geometry (as\_geos\_geometry.wk\_xy), 2
- geos\_geometry(), 6, 12
- geos\_geometry\_n, 17
- geos\_geometry\_n(), 30
- geos\_geometry\_writer (wk-methods), 33
- geos\_has\_z (geos\_area), 3
- geos\_hilbert\_code (geos\_area), 3
- geos\_inner\_join, 18
- geos\_inner\_join\_keys (geos\_inner\_join), 18
- geos\_interpolate (geos\_project), 25
- geos\_interpolate(), 25
- geos\_interpolate\_normalized
  - (geos\_project), 25
- geos\_intersection, 19
- geos\_intersection(), 19
- geos\_intersection\_prec
  - (geos\_intersection), 19
- geos\_intersects (geos\_disjoint), 13
- geos\_intersects(), 6
- geos\_intersects\_any
  - (geos\_disjoint\_matrix), 14
- geos\_intersects\_matrix
  - (geos\_disjoint\_matrix), 14
- geos\_is\_clockwise (geos\_area), 3
- geos\_is\_closed (geos\_area), 3
- geos\_is\_empty (geos\_area), 3
- geos\_is\_ring (geos\_area), 3
- geos\_is\_simple (geos\_area), 3
- geos\_is\_valid, 21
- geos\_is\_valid(), 21
- geos\_is\_valid\_detail (geos\_is\_valid), 21
- geos\_is\_valid\_detail(), 21
- geos\_is\_within\_distance
  - (geos\_distance), 15
- geos\_is\_within\_distance(), 16, 18
- geos\_largest\_empty\_circle\_spec, 21
- geos\_largest\_empty\_crc
  - (geos\_largest\_empty\_circle\_spec), 21
- geos\_length (geos\_area), 3
- geos\_length(), 25
- geos\_line\_merge (geos\_centroid), 8
- geos\_line\_merge\_directed
  - (geos\_centroid), 8
- geos\_make\_collection (geos\_make\_point), 22
- geos\_make\_collection(), 30
- geos\_make\_linestring (geos\_make\_point), 22
- geos\_make\_point, 22
- geos\_make\_polygon (geos\_make\_point), 22
- geos\_make\_valid (geos\_centroid), 8
- geos\_make\_valid(), 10
- geos\_make\_valid\_params (geos\_centroid), 8
- geos\_make\_valid\_params(), 10
- geos\_maximum\_inscribed\_circle\_spec
  - (geos\_largest\_empty\_circle\_spec), 21
- geos\_maximum\_inscribed\_crc
  - (geos\_largest\_empty\_circle\_spec), 21
- geos\_merge\_lines (geos\_centroid), 8
- geos\_minimum\_bounding\_circle
  - (geos\_largest\_empty\_circle\_spec),

- 21
- geos\_minimum\_bounding\_crc  
(geos\_largest\_empty\_circle\_spec),  
21
- geos\_minimum\_clearance (geos\_area), 3
- geos\_minimum\_clearance\_line  
(geos\_centroid), 8
- geos\_minimum\_rotated\_rectangle  
(geos\_centroid), 8
- geos\_minimum\_width (geos\_centroid), 8
- geos\_nearest, 23
- geos\_nearest\_frechet (geos\_nearest), 23
- geos\_nearest\_hausdorff (geos\_nearest),  
23
- geos\_nearest\_indexed (geos\_nearest), 23
- geos\_node (geos\_centroid), 8
- geos\_normalize (geos\_centroid), 8
- geos\_num\_coordinates (geos\_area), 3
- geos\_num\_geometries (geos\_area), 3
- geos\_num\_interior\_rings (geos\_area), 3
- geos\_num\_rings (geos\_area), 3
- geos\_offset\_curve (geos\_buffer), 7
- geos\_offset\_curve(), 7
- geos\_orientation\_index  
(geos\_segment\_intersection), 28
- geos\_orientation\_index(), 29
- geos\_overlaps (geos\_disjoint), 13
- geos\_overlaps\_any  
(geos\_disjoint\_matrix), 14
- geos\_overlaps\_matrix  
(geos\_disjoint\_matrix), 14
- geos\_point\_end (geos\_centroid), 8
- geos\_point\_n (geos\_centroid), 8
- geos\_point\_on\_surface (geos\_centroid), 8
- geos\_point\_start (geos\_centroid), 8
- geos\_polygon\_hull\_simplify  
(geos\_centroid), 8
- geos\_polygonize, 24
- geos\_polygonize\_cut\_edges  
(geos\_polygonize), 24
- geos\_polygonize\_full (geos\_polygonize),  
24
- geos\_polygonize\_valid  
(geos\_polygonize), 24
- geos\_precision (geos\_area), 3
- geos\_prepared\_contains (geos\_disjoint),  
13
- geos\_prepared\_contains\_properly  
(geos\_disjoint), 13
- geos\_prepared\_covered\_by  
(geos\_disjoint), 13
- geos\_prepared\_covers (geos\_disjoint), 13
- geos\_prepared\_crosses (geos\_disjoint),  
13
- geos\_prepared\_disjoint (geos\_disjoint),  
13
- geos\_prepared\_distance (geos\_distance),  
15
- geos\_prepared\_intersects  
(geos\_disjoint), 13
- geos\_prepared\_is\_within\_distance  
(geos\_distance), 15
- geos\_prepared\_is\_within\_distance(), 16
- geos\_prepared\_overlaps (geos\_disjoint),  
13
- geos\_prepared\_touches (geos\_disjoint),  
13
- geos\_prepared\_within (geos\_disjoint), 13
- geos\_project, 25
- geos\_project(), 25
- geos\_project\_normalized (geos\_project),  
25
- geos\_project\_normalized(), 25
- geos\_read\_geojson (geos\_read\_wkt), 26
- geos\_read\_hex (geos\_read\_wkt), 26
- geos\_read\_wkb (geos\_read\_wkt), 26
- geos\_read\_wkt, 26
- geos\_read\_xy (geos\_read\_wkt), 26
- geos\_relate, 27
- geos\_relate(), 27
- geos\_relate\_pattern (geos\_relate), 27
- geos\_relate\_pattern\_create  
(geos\_relate), 27
- geos\_relate\_pattern\_create(), 27
- geos\_relate\_pattern\_match  
(geos\_relate), 27
- geos\_remove\_repeated\_points  
(geos\_centroid), 8
- geos\_reverse (geos\_centroid), 8
- geos\_ring\_n (geos\_geometry\_n), 17
- geos\_segment\_intersection, 28
- geos\_segment\_intersection(), 29
- geos\_set\_precision (geos\_centroid), 8
- geos\_set\_srid (geos\_centroid), 8
- geos\_shared\_paths (geos\_intersection),  
19

- geos\_shared\_paths(), 19
- geos\_simplify (geos\_centroid), 8
- geos\_simplify\_preserve\_topology (geos\_centroid), 8
- geos\_snap (geos\_intersection), 19
- geos\_snap(), 19
- geos\_srid (geos\_area), 3
- geos\_strtree, 29
- geos\_strtree(), 6, 15, 23, 30
- geos\_strtree\_data (geos\_strtree), 29
- geos\_strtree\_query (geos\_strtree), 29
- geos\_sym\_difference (geos\_intersection), 19
- geos\_sym\_difference(), 19
- geos\_sym\_difference\_prec (geos\_intersection), 19
- geos\_touches (geos\_disjoint), 13
- geos\_touches\_any (geos\_disjoint\_matrix), 14
- geos\_touches\_matrix (geos\_disjoint\_matrix), 14
- geos\_transform\_xy (geos\_centroid), 8
- geos\_type (geos\_area), 3
- geos\_type\_id (geos\_area), 3
- geos\_type\_id(), 17, 23
- geos\_unary\_union (geos\_centroid), 8
- geos\_unary\_union\_prec (geos\_centroid), 8
- geos\_union (geos\_intersection), 19
- geos\_union(), 19
- geos\_union\_prec (geos\_intersection), 19
- geos\_unique\_points (geos\_centroid), 8
- geos\_unnest, 30
- geos\_unnest(), 22
- geos\_version, 31
- geos\_voronoi\_edges (geos\_delaunay\_triangles), 12
- geos\_voronoi\_polygons (geos\_delaunay\_triangles), 12
- geos\_within (geos\_disjoint), 13
- geos\_within\_any (geos\_disjoint\_matrix), 14
- geos\_within\_matrix (geos\_disjoint\_matrix), 14
- geos\_write\_geojson (geos\_read\_wkt), 26
- geos\_write\_hex (geos\_read\_wkt), 26
- geos\_write\_wkb (geos\_read\_wkt), 26
- geos\_write\_wkt (geos\_read\_wkt), 26
- geos\_write\_xy (geos\_read\_wkt), 26
- geos\_write\_xy(), 3
- geos\_x (geos\_area), 3
- geos\_x(), 3
- geos\_xmax (geos\_area), 3
- geos\_xmin (geos\_area), 3
- geos\_y (geos\_area), 3
- geos\_y(), 3
- geos\_ymax (geos\_area), 3
- geos\_ymin (geos\_area), 3
- geos\_z (geos\_area), 3
- geos\_z(), 3
- graphics::lines(), 32
- graphics::plot(), 32
- graphics::points(), 32
- graphics::polypath(), 32
- libgeos package, 31
- plot.geos\_geometry, 31
- rct(), 32, 33
- sf::st\_sfc(), 33
- vctrs-methods, 32
- vctrs::vec\_cast(), 32
- vctrs::vec\_ptype2(), 32
- vec\_cast.geos\_geometry (vctrs-methods), 32
- vec\_ptype2.geos\_geometry (vctrs-methods), 32
- wk transform, 10
- wk-methods, 33
- wk::wk\_bbox(), 5
- wk::wk\_collection(), 22, 30
- wk::wk\_coords(), 22
- wk::wk\_crs(), 3, 12, 17, 23, 27
- wk::wk\_envelope(), 6
- wk::wk\_flatten(), 22, 30
- wk::wk\_handle(), 18
- wk::wk\_linestring(), 22
- wk::wk\_polygon(), 22
- wk::xy(), 6, 22
- wk\_handle(), 33
- wk\_handle.geos\_geometry (wk-methods), 33
- wk\_handler, 33
- wk\_writer.geos\_geometry (wk-methods), 33
- wkb(), 33
- wkt(), 33

`xy()`, [33](#)