

Package: osmnxr (via r-universe)

July 6, 2026

Title Download, Model and Analyze 'OpenStreetMap' Street Networks

Version 0.1.1

Description A 'tidyverse'-friendly toolkit, inspired by the 'OSMnx' 'Python' library, to download, model, simplify, analyze and visualize street networks and other geospatial features from 'OpenStreetMap'. Build routable graphs from a place name, address, point or bounding box; simplify topology; compute shortest paths, isochrones and urban metrics (intersection density, circuitry, street-orientation entropy, centrality); and export to 'sf', 'sfnetworks' and 'MapLibre'. Heavy graph computation is performed by a bundled 'Rust' core.

Language en-US

License MIT + file LICENSE

URL <https://github.com/StrategicProjects/osmnxr>,
<https://strategicprojects.github.io/osmnxr/>

BugReports <https://github.com/StrategicProjects/osmnxr/issues>

Encoding UTF-8

RoxygenNote 8.0.0

SystemRequirements Cargo (Rust's package manager), rustc

Depends R (>= 4.2)

Imports cli, glue, httr2 (>= 1.0.0), purrr, rlang (>= 1.1.0), sf,
tibble

Suggests dodgr, ggplot2, jsonlite, knitr, rmarkdown, sfnetworks,
testthat (>= 3.0.0), tidygraph, units, xml2

Config/testthat/edition 3

VignetteBuilder knitr

Config/rextendr/version 0.5.0

NeedsCompilation yes

Author Andre Leite [aut, cre], Marcos Wasilew [aut], Hugo Vasconcelos [aut], Carlos Amorin [aut], Diogo Bezerra [aut], StrategicProjects [cph, fnd], The extendr authors [cph] (Bundled Rust crates extendr-api, extendr-ffi, extendr-macros), David Tolnay [cph] (Bundled Rust crates proc-macro2, quote, syn, paste, readonly, unicode-ident), Alex Crichton [cph] (Bundled Rust crate proc-macro2), Marvin Loebel [cph] (Bundled Rust crate lazy_static), Aleksey Kladov [cph] (Bundled Rust crate once_cell), Unicode, Inc. [cph] (Bundled Rust crate unicode-ident (Unicode-3.0 data tables))

Maintainer Andre Leite <leite@castlab.org>

Config/pak/sysreqs libclang-dev

Repository <https://cran.r-universe.dev>

Date/Publication 2026-07-05 20:30:02 UTC

RemoteUrl <https://github.com/cran/osmnxr>

RemoteRef HEAD

RemoteSha 4f7d662d8712dfcbe39054ca4bd47a3ad6fe0478

Contents

example_osm_graph	3
is_osm_graph	4
new_osm_graph	4
ox_add_edge_speeds	5
ox_add_edge_travel_times	5
ox_as_dodgr	6
ox_as_sf	7
ox_as_sfnetwork	7
ox_as_tidygraph	8
ox_basic_stats	8
ox_bearings	9
ox_centrality	10
ox_circuitry	10
ox_clear_cache	11
ox Consolidate_intersections	12
ox_distance_matrix	12
ox_distances	13
ox_example	14
ox_features_from_bbox	14
ox_features_from_place	15
ox_geocode	16
ox_geocode_to_sf	16
ox_graph_from_address	17
ox_graph_from_bbox	17
ox_graph_from_place	18
ox_graph_from_point	19

<i>example_osm_graph</i>	3
--------------------------	---

ox_isochrone	19
ox_k_shortest_paths	20
ox_load_graphml	21
ox_nearest_edges	22
ox_nearest_nodes	22
ox_orientation_entropy	23
ox_plot_figure_ground	23
ox_plot_orientation	24
ox_save_graphml	25
ox_settings	25
ox_shortest_path	26
ox_simplify	27
ox_to_geojson	27
ox_to_maplibre	28
plot.osm_graph	29

Index	30
--------------	-----------

example_osm_graph	<i>A small synthetic osm_graph for examples and tests</i>
-------------------	---

Description

Builds a tiny $n \times n$ regular street grid as an [osm_graph](#), with no network access. Edges are bidirectional and weighted by their planar length. Useful for examples, tests and learning the API offline.

Usage

```
example_osm_graph(n = 4, spacing = 100)
```

Arguments

- n Grid size; the network has $n * n$ nodes. Default 4.
- spacing Distance between adjacent nodes, in CRS units. Default 100.

Value

An [osm_graph](#) in an arbitrary projected CRS.

Examples

```
g <- example_osm_graph()
g
ox_basic_stats(g)
```

is_osm_graph	<i>Test whether an object is an osm_graph</i>
--------------	---

Description

Test whether an object is an osm_graph

Usage

```
is_osm_graph(x)
```

Arguments

x	An object.
---	------------

Value

A logical scalar.

new_osm_graph	<i>Construct an osm_graph</i>
---------------	-------------------------------

Description

Low-level constructor wrapping tidy sf nodes and edges into the central osm_graph object used across the package. Most users obtain an osm_graph from [ox_graph_from_place\(\)](#) and friends rather than calling this directly.

Usage

```
new_osm_graph(nodes, edges, meta = list())
```

Arguments

nodes	An sf object of POINT geometries with at least an integer or numeric osmid column.
edges	An sf object of LINESTRING geometries with u and v columns referencing node osmids, plus a numeric length column.
meta	A named list of metadata (e.g. network_type, simplified).

Value

An object of class osm_graph.

ox_add_edge_speeds	<i>Add edge speeds</i>
--------------------	------------------------

Description

Assigns a free-flow speed (km/h) to every edge based on its highway class, adding a speed_kph column. Unknown classes get fallback.

Usage

```
ox_add_edge_speeds(g, speeds = NULL, fallback = 40)
```

Arguments

<code>g</code>	An osm_graph .
<code>speeds</code>	Optional named numeric vector of highway = kph overrides, merged over the built-in defaults.
<code>fallback</code>	Speed (km/h) for edges with no matching class. Default 40.

Value

The [osm_graph](#) with a speed_kph edge column.

Examples

```
g <- example_osm_graph()
g <- ox_add_edge_speeds(g, speeds = c(residential = 25))
head(g$edges$speed_kph)
```

ox_add_edge_travel_times	<i>Add edge travel times</i>
--------------------------	------------------------------

Description

Adds a travel_time edge column (in seconds) from edge length (metres) and speed_kph. Speeds are added with [ox_add_edge_speeds\(\)](#) first if missing. The resulting column can be used as a routing weight for time-based shortest paths and isochrones.

Usage

```
ox_add_edge_travel_times(g)
```

Arguments

<code>g</code>	An osm_graph .
----------------	--------------------------------

Value

The [osm_graph](#) with speed_kph and travel_time edge columns.

Examples

```
g <- example_osm_graph()
g <- ox_add_edge_travel_times(g)
from <- ox_nearest_nodes(g, 0, 0)
to <- ox_nearest_nodes(g, 300, 300)
ox_shortest_path(g, from, to, weight = "travel_time")
```

ox_as_dodgr

Convert to a dodgr graph

Description

Returns a data.frame in the column layout expected by the dodgr routing package (from_id, from_lon, from_lat, to_id, to_lon, to_lat, d), suitable for dodgr::dodgr_dists() and friends.

Usage

```
ox_as_dodgr(g, weight = "length")
```

Arguments

g	An osm_graph .
weight	Edge column used as the distance/weight d. Default "length".

Value

A data.frame dodgr graph.

Examples

```
g <- example_osm_graph()
head(ox_as_dodgr(g))
```

ox_as_sf	<i>Extract sf nodes and edges from an osm_graph</i>
----------	---

Description

Extract sf nodes and edges from an osm_graph

Usage

```
ox_as_sf(g)
```

Arguments

<code>g</code>	An osm_graph.
----------------	---------------

Value

A named list with sf elements nodes and edges.

Examples

```
g <- example_osm_graph()
parts <- ox_as_sf(g)
parts$edges
```

ox_as_sfnetwork	<i>Convert to an sfnetwork</i>
-----------------	--------------------------------

Description

Returns the graph as a `sfnetworks::sfnetwork()` object, ready for the `sfnetworks/tidygraph` spatial-network workflow.

Usage

```
ox_as_sfnetwork(g, directed = TRUE)
```

Arguments

<code>g</code>	An osm_graph .
<code>directed</code>	Build a directed network. Default TRUE.

Value

An sfnetwork.

Examples

```
g <- example_osm_graph()
ox_as_sfnetwork(g)
```

ox_as_tidygraph	<i>Convert to a tidygraph table graph</i>
-----------------	---

Description

Returns the graph as a `tidygraph::tbl_graph()`, dropping geometry (node coordinates are kept as x/y columns).

Usage

```
ox_as_tidygraph(g, directed = TRUE)
```

Arguments

<code>g</code>	An osm_graph .
<code>directed</code>	Build a directed graph. Default TRUE.

Value

A `tbl_graph`.

Examples

```
g <- example_osm_graph()
ox_as_tidygraph(g)
```

ox_basic_stats	<i>Basic street-network statistics</i>
----------------	--

Description

Summary measures for an `osm_graph`: node and edge counts, total and mean edge length, mean out-degree, self-loop count and average circuitry. Computation is performed by the bundled Rust core.

Usage

```
ox_basic_stats(g, weight = "length")
```


Arguments

`g` An [osm_graph](#).
`weight` Edge column used as length/weight. Default "length".

Value

A one-row [tibble](#) of statistics.

Examples

```
g <- example_osm_graph()
ox_basic_stats(g)
```

ox_bearings	<i>Compute edge compass bearings</i>
-------------	--------------------------------------

Description

Initial compass bearing (degrees clockwise from north) of each edge, from its first to its last coordinate. Geographic coordinates are used; projected graphs are transformed to EPSG:4326 first.

Usage

```
ox_bearings(g)
```

Arguments

`g` An [osm_graph](#).

Value

A numeric vector of bearings, one per edge.

Examples

```
g <- example_osm_graph()
head(ox_bearings(g))
```

ox_centrality	<i>Node centrality</i>
---------------	------------------------

Description

Computes betweenness and/or closeness centrality for every node, using the Rust core (Brandes' algorithm for betweenness; one Dijkstra per node for closeness).

Usage

```
ox_centrality(
  g,
  type = c("betweenness", "closeness"),
  weight = "length",
  normalized = TRUE
)
```

Arguments

<code>g</code>	An osm_graph .
<code>type</code>	Centrality measures to compute: any of "betweenness" and "closeness". Default both.
<code>weight</code>	Edge column used as weight. Default "length".
<code>normalized</code>	Scale scores for comparability across graphs. Betweenness is divided by $(n - 1)(n - 2)$; closeness uses the Wasserman–Faust correction for disconnected graphs. Default TRUE.

Value

A [tibble](#) with column `osmid` plus one column per requested measure.

Examples

```
g <- example_osm_graph(n = 4)
ox_centrality(g, type = "betweenness")
```

ox_circuity	<i>Average network circuity</i>
-------------	---------------------------------

Description

The ratio of total edge length to total straight-line (great-circle for geographic CRS, Euclidean for projected) distance between edge endpoints. A value of 1 means perfectly straight streets; higher values indicate more winding networks.

Usage

```
ox_circuitry(g)
```

Arguments

g An [osm_graph](#).

Value

A numeric scalar (≥ 1).

Examples

```
g <- example_osm_graph()
ox_circuitry(g)
```

ox_clear_cache	<i>Clear the session cache</i>
----------------	--------------------------------

Description

Empties the in-memory cache of downloaded OpenStreetMap responses.

Usage

```
ox_clear_cache()
```

Value

Invisibly NULL.

Examples

```
ox_clear_cache()
```

 ox_consolidate_intersections

Consolidate nearby intersections

Description

Merges groups of nodes lying within tolerance of one another into single nodes placed at the group centroid, then rewrites edges to the consolidated nodes and drops the resulting self-loops. Useful for collapsing the multiple OSM nodes that represent one complex junction (e.g. dual carriageways).

Usage

```
ox_consolidate_intersections(g, tolerance = 10)
```

Arguments

g	An osm_graph .
tolerance	Distance below which nodes are merged, in CRS units. Default 10.

Details

Clustering uses `sf::st_is_within_distance()`; connected components are found by the Rust core. `tolerance` is in the units of the graph CRS, so project the graph first (e.g. to a metric CRS) for a meaningful distance.

Value

A consolidated [osm_graph](#) (with `meta$consolidated = TRUE`).

Examples

```
g <- example_osm_graph(n = 4, spacing = 100)
# nothing is within 10 units here, so the graph is unchanged
ox_consolidate_intersections(g, tolerance = 10)
```

 ox_distance_matrix

Shortest-path distance matrix

Description

Computes the matrix of minimum-weight distances between every from node and every to node (Rust core; one Dijkstra per source).

Usage

```
ox_distance_matrix(g, from, to = from, weight = "length")
```

Arguments

g	An osm_graph .
from	Node osmids for the matrix rows.
to	Node osmids for the matrix columns. Defaults to from.
weight	Edge column used as weight. Default "length".

Value

A numeric matrix (length(from) x length(to)) with osmid dimnames; Inf marks unreachable pairs.

Examples

```
g <- example_osm_graph(n = 3)
nodes <- g$nodes$osmid
ox_distance_matrix(g, from = nodes[1:2], to = nodes[3:4])
```

ox_distances	<i>Single-source shortest distances</i>
--------------	---

Description

Minimum-weight distance from from to every node in the graph.

Usage

```
ox_distances(g, from, weight = "length")
```

Arguments

g	An osm_graph .
from	A node osmid.
weight	Edge column used as weight. Default "length".

Value

A [tibble](#) with columns osmid and distance (Inf for unreachable nodes).

Examples

```
g <- example_osm_graph()
ox_distances(g, ox_nearest_nodes(g, 0, 0))
```

 ox_example

Load a bundled real-world example network

Description

Loads a small, real street network shipped with the package (downloaded once from OpenStreetMap and simplified) so that examples and vignettes can show real analyses without network access.

Usage

```
ox_example(name = c("olinda", "manhattan", "rome"))
```

Arguments

name	Which network to load (all drivable, simplified): "olinda" (historic centre of Olinda, Pernambuco, Brazil), "manhattan" (a square mile of Midtown Manhattan, New York — a strong grid), or "rome" (the organic centro storico of Rome, Italy).
------	--

Value

An [osm_graph](#).

Examples

```
g <- ox_example("olinda")
g
ox_basic_stats(g)
```

 ox_features_from_bbox

Download features within a bounding box

Description

Queries OpenStreetMap (via Overpass) for elements matching tags — points of interest, amenities, buildings, transit stops, and so on — returning them as an `sf` of points (ways and relations are represented by their centroid).

Usage

```
ox_features_from_bbox(bbox, tags)
```

Arguments

bbox	Numeric <code>c(xmin, ymin, xmax, ymax)</code> in longitude/latitude.
tags	Named list of OSM tag filters. Each element is either TRUE (key present with any value) or a character vector of allowed values, e.g. <code>list(amenity = c("school", "hospital"))</code> .

Value

An sf of POINT features with osm_type, osm_id and one column per tag encountered.

Examples

```
bbox <- c(-34.91, -8.07, -34.87, -8.04)
ox_features_from_bbox(bbox, tags = list(amenity = "school"))
```

`ox_features_from_place`*Download features for a named place*

Description

Geocodes query with [ox_geocode\(\)](#) and downloads matching features around it. See [ox_features_from_bbox\(\)](#) for the tags format.

Usage

```
ox_features_from_place(query, tags, dist = 2000)
```

Arguments

query	A place name, e.g. "Recife, Brazil".
tags	Named list of OSM tag filters.
dist	Search half-width in metres around the geocoded point. Default 2000.

Value

An sf of POINT features.

Examples

```
ox_features_from_place("Olinda, Brazil", tags = list(amenity = "hospital"))
```

ox_geocode	<i>Geocode a place or address</i>
------------	-----------------------------------

Description

Resolve a free-form query to coordinates and metadata using the OpenStreetMap Nominatim service.

Usage

```
ox_geocode(query, limit = 1)
```

Arguments

query	A character scalar, e.g. "Recife, Brazil".
limit	Maximum number of results. Default 1.

Value

A [tibble](#) with columns display_name, lat, lon, osm_type, osm_id and class.

Examples

```
ox_geocode("Recife, Brazil")
```

ox_geocode_to_sf	<i>Geocode a place to an sf boundary</i>
------------------	--

Description

Like [ox_geocode\(\)](#) but returns the place geometry (boundary polygon when available, otherwise a point) as an sf object.

Usage

```
ox_geocode_to_sf(query, limit = 1)
```

Arguments

query	A character scalar, e.g. "Recife, Brazil".
limit	Maximum number of results. Default 1.

Value

An sf object (one row per result) in EPSG:4326.

Examples

```
ox_geocode_to_sf("Recife, Brazil")
```

ox_graph_from_address *Download a street network around an address*

Description

Download a street network around an address

Usage

```
ox_graph_from_address(address, dist = 1000, network_type = "drive")
```

Arguments

address	A street address.
dist	Buffer half-width in metres. Default 1000.
network_type	One of "drive", "walk", "bike" or "all".

Value

An [osm_graph](#).

Examples

```
g <- ox_graph_from_address("Marco Zero, Recife", dist = 600)
```

ox_graph_from_bbox *Download a street network within a bounding box*

Description

Download a street network within a bounding box

Usage

```
ox_graph_from_bbox(bbox, network_type = "drive")
```

Arguments

bbox	Numeric vector c(xmin, ymin, xmax, ymax) in longitude/latitude (EPSG:4326).
network_type	One of "drive", "walk", "bike" or "all".

Value

An [osm_graph](#).

Examples

```
bbox <- c(-34.91, -8.07, -34.87, -8.04)
g <- ox_graph_from_bbox(bbox, network_type = "drive")
```

ox_graph_from_place	<i>Download a street network for a named place</i>
---------------------	--

Description

Geocodes query with [ox_geocode\(\)](#) and downloads the street network within the bounding box of the matched place.

Usage

```
ox_graph_from_place(query, network_type = "drive")
```

Arguments

query	A place name, e.g. "Recife, Brazil".
network_type	One of "drive", "walk", "bike" or "all".

Value

An [osm_graph](#).

Examples

```
g <- ox_graph_from_place("Olinda, Brazil", network_type = "drive")
```

ox_graph_from_point	<i>Download a street network around a point</i>
---------------------	---

Description

Download a street network around a point

Usage

```
ox_graph_from_point(point, dist = 1000, network_type = "drive")
```

Arguments

point	Numeric c(lon, lat).
dist	Buffer half-width in metres (a square bounding box of side 2 * dist is used). Default 1000.
network_type	One of "drive", "walk", "bike" or "all".

Value

An [osm_graph](#).

Examples

```
g <- ox_graph_from_point(c(-34.89, -8.05), dist = 800)
```

ox_isochrone	<i>Compute isochrones (service areas)</i>
--------------	---

Description

For one or more origin nodes, finds the set of nodes reachable within each cutoff (by the chosen edge weight — distance or, with [ox_add_edge_travel_times\(\)](#), travel time) and returns a polygon per cutoff: the hull of the reachable nodes. With several origins, reachability is the minimum cost from any origin.

Usage

```
ox_isochrone(g, center, cutoffs, weight = "length", ratio = 0.4)
```

Arguments

<code>g</code>	An osm_graph .
<code>center</code>	One or more origin node osmids (see ox_nearest_nodes()).
<code>cutoffs</code>	Numeric vector of cutoff values, in the units of weight.
<code>weight</code>	Edge column used as cost. Default "length".
<code>ratio</code>	Concavity for <code>sf::st_concave_hull()</code> (0 = most concave, 1 = convex). Default 0.4.

Details

Reachable sets come from the Rust Dijkstra core; the hull is built with `sf::st_concave_hull()` when available (GEOS ≥ 3.11), falling back to a convex hull. For metric cutoffs, project the graph to a metric CRS first.

Value

An `sf` object with one polygon row per cutoff (columns `cutoff`, `n_nodes`, `geometry`), ordered from largest to smallest cutoff so smaller areas draw on top.

Examples

```
g <- example_osm_graph(n = 6, spacing = 100)
center <- ox_nearest_nodes(g, 250, 250)
iso <- ox_isochrone(g, center, cutoffs = c(100, 300))
iso
```

`ox_k_shortest_paths` *k shortest paths between two nodes*

Description

Computes up to `k` loopless shortest paths from `from` to `to` using Yen's algorithm in the Rust core. Useful for route alternatives.

Usage

```
ox_k_shortest_paths(g, from, to, k = 3, weight = "length")
```

Arguments

<code>g</code>	An osm_graph .
<code>from, to</code>	Node osmids.
<code>k</code>	Number of paths to return. Default 3.
<code>weight</code>	Edge column used as weight. Default "length".

Value

A [tibble](#) with one row per path: rank, cost and a list-column path of node osmids, ordered by increasing cost. Fewer than k rows are returned when fewer distinct paths exist.

Examples

```
g <- example_osm_graph()
from <- ox_nearest_nodes(g, 0, 0)
to <- ox_nearest_nodes(g, 200, 200)
ox_k_shortest_paths(g, from, to, k = 3)
```

ox_load_graphml	<i>Load a graph from GraphML</i>
-----------------	----------------------------------

Description

Reads a GraphML file written by [ox_save_graphml\(\)](#) back into an [osm_graph](#), restoring node coordinates, edge attributes and edge geometry (from the stored WKT).

Usage

```
ox_load_graphml(path)
```

Arguments

path	Path to a .graphml file.
------	--------------------------

Value

An [osm_graph](#).

Examples

```
g <- example_osm_graph()
f <- tempfile(fileext = ".graphml")
ox_save_graphml(g, f)
ox_load_graphml(f)
```

ox_nearest_edges	<i>Find the nearest edge to a point</i>
------------------	---

Description

Returns, for each supplied coordinate, the graph edge closest in planar distance.

Usage

```
ox_nearest_edges(g, x, y)
```

Arguments

<code>g</code>	An osm_graph .
<code>x, y</code>	Numeric vectors of coordinates in the graph's CRS.

Value

An sf subset of `g$edges`, one row per input point.

Examples

```
g <- example_osm_graph()
ox_nearest_edges(g, x = 50, y = 0)
```

ox_nearest_nodes	<i>Find the nearest node to a point</i>
------------------	---

Description

Returns the osmid of the graph node closest (in planar distance) to each supplied coordinate.

Usage

```
ox_nearest_nodes(g, x, y)
```

Arguments

<code>g</code>	An osm_graph .
<code>x, y</code>	Numeric vectors of coordinates in the graph's CRS.

Value

An integer/numeric vector of node osmids, one per input point.

Examples

```
g <- example_osm_graph()
ox_nearest_nodes(g, x = 0, y = 0)
```

 ox_orientation_entropy

Street-orientation entropy

Description

Shannon entropy (in nats) of the distribution of edge compass bearings, binned into equal sectors. Higher values indicate a more disordered (organic) network; lower values a more ordered (gridiron) one.

Usage

```
ox_orientation_entropy(x, num_bins = 36)
```

Arguments

x An [osm_graph](#) or a numeric vector of bearings (degrees), e.g. from [ox_bearings\(\)](#).
num_bins Number of equal bearing sectors over $[0, 360)$. Default 36.

Value

A numeric scalar (entropy in nats).

Examples

```
g <- example_osm_graph()
ox_orientation_entropy(g)
```

 ox_plot_figure_ground *Figure-ground diagram of a street network*

Description

Draws a figure-ground diagram: the streets in a single colour on a solid background, with no axes or margins. Cropping different places to the same extent makes their network form directly comparable, as in Boeing (2025).

Usage

```
ox_plot_figure_ground(g, bg = "black", col = "white", lwd = 1.2, title = NULL)
```

Arguments

<code>g</code>	An osm_graph .
<code>bg, col</code>	Background and street colours. Default white-on-black.
<code>lwd</code>	Street line width. Default 1.2.
<code>title</code>	Optional panel title.

Value

Invisibly, the `osm_graph`.

Examples

```
g <- example_osm_graph()
ox_plot_figure_ground(g)
```

`ox_plot_orientation` *Polar plot of street orientations*

Description

Draws a polar histogram (rose plot) of edge compass bearings, the standard visual summary of a street network's orientation order. Requires `ggplot2`.

Usage

```
ox_plot_orientation(
  x,
  num_bins = 36,
  fill = "#0d3b66",
  title = "Street orientation"
)
```

Arguments

<code>x</code>	An osm_graph or a numeric vector of bearings (degrees), e.g. from ox_bearings() .
<code>num_bins</code>	Number of equal bearing sectors. Default 36.
<code>fill</code>	Bar fill colour. Default the package blue.
<code>title</code>	Optional plot title.

Value

A `ggplot` object.

Examples

```
g <- example_osm_graph()
ox_plot_orientation(g)
```

ox_save_graphml	<i>Save a graph to GraphML</i>
-----------------	--------------------------------

Description

Writes the graph to a GraphML file compatible with OSMnx / NetworkX / Gephi. Edge geometry is preserved losslessly as a WKT attribute, so the graph round-trips through `ox_load_graphml()`.

Usage

```
ox_save_graphml(g, path)
```

Arguments

g	An osm_graph .
path	Output .graphml path.

Value

path, invisibly.

Examples

```
g <- example_osm_graph()
f <- tempfile(fileext = ".graphml")
ox_save_graphml(g, f)
```

ox_settings	<i>Get or set package settings</i>
-------------	------------------------------------

Description

Configure the Overpass and Nominatim endpoints, HTTP behaviour and caching used by all `ox_*` download functions. Called with no arguments it returns the current settings as a list; called with named arguments it updates them and returns the previous values invisibly.

Usage

```
ox_settings(...)
```

Arguments

...	Named settings to update. Recognised names: <code>overpass_url</code> , <code>nominatim_url</code> , <code>user_agent</code> , <code>timeout</code> , <code>max_tries</code> , <code>cache</code> .
-----	---

Value

A named list of settings (current values, or the previous values invisibly when updating).

Examples

```
ox_settings()
old <- ox_settings(timeout = 300)
ox_settings(timeout = old$timeout) # restore
```

ox_shortest_path	<i>Shortest path between two nodes</i>
------------------	--

Description

Computes the minimum-weight path from `from` to `to` using Dijkstra's algorithm in the Rust core.

Usage

```
ox_shortest_path(g, from, to, weight = "length")
```

Arguments

<code>g</code>	An osm_graph .
<code>from, to</code>	Node osmids (as returned by ox_nearest_nodes()).
<code>weight</code>	Edge column used as weight. Default "length".

Value

A vector of node osmids describing the path (length 0 if the target is unreachable).

Examples

```
g <- example_osm_graph()
from <- ox_nearest_nodes(g, 0, 0)
to <- ox_nearest_nodes(g, 300, 300)
ox_shortest_path(g, from, to)
```

`ox_simplify`*Simplify street-network topology*

Description

Removes interstitial (degree-2) nodes that merely shape the geometry of a street, merging each maximal chain of such nodes into a single edge whose geometry follows the original points and whose length is the sum of the merged segments. Only true endpoints and intersections are kept as nodes.

Usage

```
ox_simplify(g)
```

Arguments

`g` An unsimplified [osm_graph](#).

Details

The topology walk is performed by the Rust core; geometry is rebuilt with `sf`. Downloaded graphs are unsimplified by default; this is the standard cleanup step before analysis.

Value

A simplified [osm_graph](#) (with `meta$simplified = TRUE`).

Examples

```
g <- example_osm_graph()
ox_simplify(g) # already simplified: returned unchanged
```

`ox_to_geojson`*Export to GeoJSON*

Description

Writes the graph's edges (or nodes) to a GeoJSON file via `sf::st_write()`. Geometry is transformed to EPSG:4326, the GeoJSON standard CRS.

Usage

```
ox_to_geojson(g, path, layer = c("edges", "nodes"))
```

Arguments

<code>g</code>	An osm_graph .
<code>path</code>	Output file path.
<code>layer</code>	Which layer to write: "edges" (default) or "nodes".

Value

path, invisibly.

Examples

```
g <- example_osm_graph()
ox_to_geojson(g, tempfile(fileext = ".geojson"))
```

ox_to_maplibre	<i>Build a MapLibre GL style fragment</i>
----------------	---

Description

Writes the edges to a GeoJSON file and returns a MapLibre GL JS style fragment (a list with sources and layers) that references it, ready to merge into a map style. Serialize with, e.g., `jsonlite::toJSON(..., auto_unbox = TRUE)`.

Usage

```
ox_to_maplibre(
  g,
  path,
  source_id = "osmnr",
  layer_id = "streets",
  url = basename(path)
)
```

Arguments

<code>g</code>	An osm_graph .
<code>path</code>	GeoJSON output path for the edge data.
<code>source_id, layer_id</code>	Identifiers for the MapLibre source and layer.
<code>url</code>	URL the style should use to fetch the data. Defaults to <code>basename(path)</code> .

Value

A named list with sources and layers, invisibly written data to path.

Examples

```
g <- example_osm_graph()
style <- ox_to_maplibre(g, tempfile(fileext = ".geojson"))
names(style)
```

plot.osm_graph	<i>Plot an osm_graph</i>
----------------	--------------------------

Description

Draws the street-network edges (and optionally nodes) using base sf plotting.

Usage

```
## S3 method for class 'osm_graph'
plot(x, nodes = FALSE, col = "#0d3b66", lwd = 0.7, ...)
```

Arguments

x	An osm_graph.
nodes	Logical; overlay node points. Default FALSE.
col, lwd	Passed to the edge geometry plot.
...	Further arguments passed to sf::plot.sf() .

Value

Invisibly, the osm_graph.

Index

example_osm_graph, 3

is_osm_graph, 4

new_osm_graph, 4

osm_graph, 3, 5–14, 17–28

osm_graph (new_osm_graph), 4

ox_add_edge_speeds, 5

ox_add_edge_speeds(), 5

ox_add_edge_travel_times, 5

ox_add_edge_travel_times(), 19

ox_as_dodgr, 6

ox_as_sf, 7

ox_as_sfnetwork, 7

ox_as_tidygraph, 8

ox_basic_stats, 8

ox_bearings, 9

ox_bearings(), 23, 24

ox_centrality, 10

ox_circuitry, 10

ox_clear_cache, 11

ox_consolidate_intersections, 12

ox_distance_matrix, 12

ox_distances, 13

ox_example, 14

ox_features_from_bbox, 14

ox_features_from_bbox(), 15

ox_features_from_place, 15

ox_geocode, 16

ox_geocode(), 15, 16, 18

ox_geocode_to_sf, 16

ox_graph_from_address, 17

ox_graph_from_bbox, 17

ox_graph_from_place, 18

ox_graph_from_place(), 4

ox_graph_from_point, 19

ox_isochrone, 19

ox_k_shortest_paths, 20

ox_load_graphml, 21

ox_load_graphml(), 25

ox_nearest_edges, 22

ox_nearest_nodes, 22

ox_nearest_nodes(), 20, 26

ox_orientation_entropy, 23

ox_plot_figure_ground, 23

ox_plot_orientation, 24

ox_save_graphml, 25

ox_save_graphml(), 21

ox_settings, 25

ox_shortest_path, 26

ox_simplify, 27

ox_to_geojson, 27

ox_to_maplibre, 28

plot.osm_graph, 29

sf::plot.sf(), 29

sf::st_write(), 27

tibble, 9, 10, 13, 16, 21