# Package 'rcrisp'

August 21, 2025

**Title** Automate the Delineation of Urban River Spaces

**Version** 0.2.0

**Description** Provides tools to automate the morphological delineation of riverside urban areas based on a method introduced in Forgaci (2018) <doi:10.7480/abe.2018.31>. Delineation entails the identification of corridor boundaries, segmentation of the corridor, and delineation of the river space using two-dimensional spatial information from street network data and digital elevation data in a projected CRS. The resulting delineation can be used to characterise spatial phenomena that can be related to the river as a central element.

**License** Apache License (>= 2)

**URL** https://cityriverspaces.github.io/rcrisp/, https://doi.org/10.5281/zenodo.15793526

**BugReports** https://github.com/CityRiverSpaces/rcrisp/issues

**Depends** R (>= 4.1.0)

**Imports** checkmate, dbscan, dplyr, lwgeom, osmdata, rcoins, rlang, rstac, sf (>= 0.9.0), sfheaders, sfnetworks, stringr, terra, tidygraph, units, visor

**Suggests** knitr, purrr, rmarkdown, testthat (>= 3.0.0), withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Claudiu Forgaci [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3218-5102>), Francesco Nattino [aut] (ORCID: <https://orcid.org/0000-0003-3286-0139>), Fakhereh Alidoost [ctb] (ORCID:

<https://orcid.org/0000-0001-8407-6472>),
Meiert Willem Grootes [ctb] (ORCID:
<https://orcid.org/0000-0002-5733-4795>),
Netherlands eScience Center [fnd]

**Maintainer** Claudiu Forgaci <c.forgaci@tudelft.nl>

**Repository** CRAN

**Date/Publication** 2025-08-21 17:20:02 UTC

# Contents

## as_bbox *Get the bounding box from the x object*

### Description

If the x does not have a CRS, WGS84 is assumed.

### Usage

```
as_bbox(x)
```

### Arguments

| | |
|---|---|
| x | Simple feature object (or compatible) or a bounding box, provided either as a matrix (with x, y as rows and min, max as columns) or as a vector (xmin, ymin, xmax, ymax) |

### Value

A bbox object as returned by `sf::st_bbox()`

### Examples

```
library(sf)
bounding_coords <- c(25.9, 44.3, 26.2, 44.5)
bb <- as_bbox(bounding_coords)
class(bb)
st_crs(bb)
```

## as_crs *Standardise the coordinate reference system (CRS) of an object*

### Description

Standardise the coordinate reference system (CRS) of an object

### Usage

```
as_crs(x, allow_geographic = FALSE)
```

### Arguments

| | |
|---|---|
| x | An object of class `sf`, `sfc`, `bbox`, or a numeric or character vector representing a CRS (e.g., EPSG code). If `numeric`, the value should be an unrestricted positive number representing a valid EPSG code. |
| allow_geographic | Logical, whether to allow geographic CRS (lat/lon). |

## Value

An object of class sf::crs with a valid CRS.

## Examples

```
library(sf)

# Standardise a numeric EPSG code
as_crs(4326, allow_geographic = TRUE)

# Standardise a character EPSG code
as_crs("EPSG:4326", allow_geographic = TRUE)

# Standardise a bbox object
bb <- st_bbox(c(xmin = 25.9, ymin = 44.3, xmax = 26.2, ymax = 44.5),
              crs = 4326)
as_crs(bb, allow_geographic = TRUE)

# Standardise a simple feature object
bb_sfc <- st_as_sfc(bb)
bb_sf <- st_as_sf(bb_sfc)
as_crs(bb_sf, allow_geographic = TRUE)
as_crs(bb_sfc, allow_geographic = TRUE)
```

---

# as_network *Create a network from a collection of line strings*

---

## Description

Create a network from a collection of line strings

## Usage

```
as_network(edges, flatten = TRUE, clean = TRUE)
```

## Arguments

| | |
|---|---|
| edges | An sf::sf or sf::sfc_LINESTRING object with the network edges |
| flatten | Whether all intersections between edges should be converted to nodes |
| clean | Whether general cleaning tasks should be run on the generated network (see clean_network() for the description of tasks) |

## Value

An sfnetworks::sfnetwork object

## Examples

```
edges <- sf::st_sfc(
  sf::st_linestring(matrix(c(0, 0, 1, 1), ncol = 2, byrow = TRUE)),
  sf::st_linestring(matrix(c(0, 1, 1, 0), ncol = 2, byrow = TRUE)),
  crs = sf::st_crs("EPSG:32635")
)

# Run with default values
as_network(edges)

# Only build the spatial network
as_network(edges, flatten = FALSE, clean = FALSE)
```

---

# bucharest_dambovita *rcrisp example delineation data for Bucharest*

---

## Description

Delineation generated with rcrisp example data found at https://zenodo.org/records/16325879

## Usage

```
bucharest_dambovita
```

## Format

A list of sf objects representing:

**valley** The valley boundaries of river Dâmbovița.

**corridor** The corridor boundaries of river Dâmbovița.

**segments** Corridor segments of river Dâmbovița.

**riverspace** River space of Dâmbovița.

---

# cache_directory *Return the cache directory used by the package*

---

## Description

By default, the user-specific directory as returned by `tools::R_user_dir()` is used. A different directory can be used by setting the environment variable `CRISP_CACHE_DIRECTORY`. This can also be done by adding the following line to the `.Renviron` file: `CRISP_CACHE_DIRECTORY=/path/to/crisp/cache/dir`.

## Usage

```
cache_directory()
```

## Value

The cache directory used by rcrisp.

## Examples

```
cache_directory()
```

---

`check_cache` *Check cache*

---

## Description

A warning is raised if the cache size is > 100 MB or if it includes files older than 30 days.

## Usage

```
check_cache()
```

## Value

`NULL`

## Examples

```
check_cache()
```

---

`clean_network` *Clean a spatial network*

---

## Description

Subdivide edges by adding missing nodes, (optionally) simplify the network (see `simplify_network()`), remove pseudo-nodes, and discard all but the main connected component.

## Usage

```
clean_network(network, simplify = TRUE)
```

## Arguments

| | |
|---|---|
| `network` | A network object of class `sfnetworks::sfnetwork` |
| `simplify` | Whether the network should be simplified with `simplify_network()` |

## Value

A cleaned network object of class sfnetworks::sfnetwork

## Examples

```
bucharest_osm <- get_osm_example_data()
edges <- dplyr::bind_rows(bucharest_osm$streets,
                          bucharest_osm$railways)
network <- sfnetworks::as_sfnetwork(edges, directed = FALSE)
clean_network(network)
```

---

# clear_cache *Remove cache files*

---

## Description

Remove files from cache directory either before a given date or entirely.

## Usage

```
clear_cache(before_date = NULL)
```

## Arguments

| | |
|---|---|
| before_date | Date before which cache files should be removed provided as object of class Date or as a case dependent character vector accepted by as.Date() |

## Value

List of file paths of removed files

## Examples

```
# Clear all cache
clear_cache()

# Clear cache before given date
before_date <- as.Date("1-1-1999", "%m-%d-%Y")
clear_cache(before_date = before_date)
```

## default_stac_dem *Default STAC collection*

### Description

Endpoint and collection ID of the default STAC collection where to access digital elevation model (DEM) data. This is the global Copernicus DEM 30 dataset hosted on AWS, as listed in the Earth-Search STAC API endpoint. Note that AWS credentials need to be set up in order to access the data (not the catalog). References:

- EarthSearch STAC API
- Copernicus DEM
- AWS Copernicus DEM datasets
- Data license

### Usage

```
default_stac_dem
```

### Format

An object of class `list` of length 2.

## delineate *Delineate a corridor around a river*

### Description

Delineate a corridor around a river

### Usage

```
delineate(
  city_name,
  river_name,
  crs = NULL,
  network_buffer = NULL,
  buildings_buffer = NULL,
  corridor_init = "valley",
  dem = NULL,
  dem_buffer = 2500,
  max_iterations = 10,
  capping_method = "shortest-path",
  angle_threshold = 100,
  corridor = TRUE,
```

```
  segments = FALSE,
  riverspace = FALSE,
  force_download = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| city_name | A character vector of length one |
| river_name | A character vector of length one |
| crs | The projected Coordinate Reference System (CRS) to use. If not provided, the suitable Universal Transverse Mercator (UTM) CRS is selected |
| network_buffer | Add a buffer (an integer in meters) around river to retrieve additional data (streets, railways, etc.). Default is 3000 m. |
| buildings_buffer | Add a buffer (an integer in meters) around the river to retrieve additional data (buildings). Default is 100 m. |
| corridor_init | How to estimate the initial guess of the river corridor. It can take the following values:<br>• "valley": use the river valley boundary, as estimated from a Digital Elevation Model (DEM) (for more info see delineate_valley())<br>• numeric or integer: use a buffer region of the given size (in meters) around the river centerline<br>• An sf::sf or sf::sfc object: use the given input geometry |
| dem | Digital elevation model (DEM) of the region (only used if corridor_init is "valley") |
| dem_buffer | Size of the buffer region (in meters) around the river to retrieve the DEM (only used if corridor_init is "valley" and dem is NULL). |
| max_iterations | Maximum number of iterations employed to refine the corridor edges (see corridor_edge()). |
| capping_method | The method employed to connect the corridor edge end points (i.e., to "cap" the corridor), as character vector of length one. See cap_corridor() for the available methods. |
| angle_threshold | Only network edges forming angles above this threshold (in degrees) are considered when forming segment edges. See delineate_segments() and rcoins::stroke(). Only used if segments is TRUE. |
| corridor | Whether to carry out the corridor delineation |
| segments | Whether to carry out the corridor segmentation |
| riverspace | Whether to carry out the riverspace delineation |
| force_download | Download data even if cached data is available |
| ... | Additional (optional) input arguments for retrieving the DEM dataset (see get_dem()). Only relevant if corridor_init is "valley" and dem is NULL |

## Value

A list with the corridor, segments, and riverspace geometries as `sf::sfc_POLYGON` objects.

## Examples

```
# Set parameters
city <- "Bucharest"
river <- "Dâmbovița"

# Delineate with defaults
delineate(city, river)

# Use custom CRS
delineate(city, river, crs = "EPSG:31600")  # National projected CRS

# Use custom network buffer
delineate(city, river, network_buffer = 3500)

# Use custom buildings buffer
delineate(city, river, buildings_buffer = 150, riverspace = TRUE)

# Provide DEM as input
bucharest_dem <- get_dem_example_data()
delineate(city, river, dem = bucharest_dem)
```

---

# delineate_corridor *Delineate a river corridor on a spatial network*

---

## Description

The corridor edges on the two river banks are drawn on the provided spatial network starting from an initial guess of the corridor (based e.g. on the river valley).

## Usage

```
delineate_corridor(
  network,
  river,
  corridor_init = 1000,
  max_width = 3000,
  max_iterations = 10,
  capping_method = "shortest-path"
)
```

## Arguments

| | |
|---|---|
| network | The spatial network of class sfnetworks::sfnetwork to be used for the delineation. Required, no default. |
| river | A (MULTI)LINESTRING simple feature geometry of class sf::sf or sf::sfc representing the river centerline. Required, no default. |
| corridor_init | How to estimate the initial guess of the river corridor. It can take the following values:<br>• numeric or integer: use a buffer region of the given size (in meters, positive, unrestricted) around the river centerline<br>• An sf::sf or sf::sfc object: use the given input geometry |
| max_width | A positive number representing the (approximate) maximum width of the corridor in meters. The upper limit is unrestricted. The spatial network is trimmed by a buffer region of this size around the river. |
| max_iterations | A positive integer greater than 0, with upper limit unrestricted, representing the maximum number of iterations employed to refine the corridor edges (see corridor_edge()). |
| capping_method | Case-insensitive character vector of length 1 with the method employed to connect the corridor edge end points (i.e. to "cap" the corridor). See cap_corridor() for the available methods. |

## Value

A simple feature geometry of class sf::sfc_POLYGON representing the river corridor

## Examples

```
bucharest_osm <- get_osm_example_data()
streets <- bucharest_osm$streets
railways <- bucharest_osm$railways
river <- bucharest_osm$river_centerline

# Delineate with default values
network <- rbind(streets, railways) |> as_network()
delineate_corridor(network = network, river = river)

# Delineate with user-specified parameters
bucharest_dem <- get_dem_example_data()
corridor_init <- delineate_valley(dem = bucharest_dem, river = river)
delineate_corridor(network = network, river = river,
                   corridor_init = corridor_init,
                   max_width = 4000, max_iterations = 5,
                   capping_method = "direct")
```

## delineate_riverspace *Delineate the space surrounding a river*

### Description

Delineate the space surrounding a river

### Usage

```
delineate_riverspace(
  river,
  occluders = NULL,
  density = 1/50,
  ray_num = 40,
  ray_length = 100
)
```

### Arguments

| | |
|---|---|
| river | List with river surface and centerline |
| occluders | Geometry of occluders |
| density | Density of viewpoints |
| ray_num | Number of rays as numeric vector of length one |
| ray_length | Length of rays in meters as numeric vector of length one |

### Value

Riverspace as object of class sf::sfc_POLYGON

### Examples

```
bucharest_osm <- get_osm_example_data()
delineate_riverspace(bucharest_osm$river_surface, bucharest_osm$buildings)
```

## delineate_segments *Split a river corridor into segments*

### Description

Segments are defined as corridor subregions separated by river-crossing transversal lines that form continuous strokes in the network.

### Usage

```
delineate_segments(corridor, network, river, angle_threshold = 100)
```

### Arguments

| | |
|---|---|
| corridor | The river corridor as a simple feature geometry of class sfc_POLYGON |
| network | The spatial network of class sfnetwork to be used for the segmentation |
| river | The river centerline as a simple feature geometry of class sf::sf or sf::sfc |
| angle_threshold | Only consider angles above this threshold (in degrees) to form continuous strokes in the network. The value can range between 0 and 180, with the default set to 100. See rcoins::stroke() for more details. |

### Value

Segment polygons as a simple feature geometry of class sf::sfc_POLYGON

### Examples

```
bucharest_osm <- get_osm_example_data()
corridor <- bucharest_dambovita$corridor
network <- rbind(bucharest_osm$streets, bucharest_osm$railways) |>
  as_network()
river <- bucharest_osm$river_centerline
delineate_segments(corridor, network, river)
```

---

## delineate_valley *Extract the river valley from the DEM*

---

### Description

The slope of the digital elevation model (DEM) is used as friction (cost) surface to compute the cost distance from any grid cell of the raster to the river. A characteristic value (default: the mean) of the cost distance distribution in a region surrounding the river (default: a buffer region of 2 km) is then calculated, and used to threshold the cost-distance surface. The resulting area is then "polygonized" to obtain the valley boundary as a simple feature geometry.

### Usage

```
delineate_valley(dem, river)
```

### Arguments

| | |
|---|---|
| dem | SpatRaster object with the digital elevation model of the region |
| river | An object of class sf::sf or sf::sfc representing the river |

### Value

River valley as a simple feature geometry of class sfc_MULTIPOLYGON

## Examples

```
bucharest_osm <- get_osm_example_data()
bucharest_dem <- get_dem_example_data()
delineate_valley(bucharest_dem, bucharest_osm$river_centerline)
```

---

`flatten_network` *Flatten a network by adding points at apparent intersections*

---

## Description

All crossing edges are identified, and the points of intersections are injected within the edge geometries. Note that the injected points are not converted to network nodes (this can be achieved via sfnetworks' `sfnetworks::to_spatial_subdivision()`, which is part of the tasks that are included in `clean_network()`.

## Usage

```
flatten_network(network)
```

## Arguments

| | |
|---|---|
| `network` | A network object of class `sfnetworks::sfnetwork` |

## Details

The functionality is similar to sfnetworks' `sfnetworks::st_network_blend()`, but in that case an external point is only injected to the closest edge.

## Value

An `sfnetworks::sfnetwork` object with additional points at intersections

## Examples

```
bucharest_osm <- get_osm_example_data()
edges <- dplyr::bind_rows(bucharest_osm$streets,
                          bucharest_osm$railways)
network <- sfnetworks::as_sfnetwork(edges, directed = FALSE)
flatten_network(network)
```

get_dem *Access digital elevation model (DEM) for a given region*

## Description

Access digital elevation model (DEM) for a given region

## Usage

```
get_dem(
  bb,
  dem_source = "STAC",
  stac_endpoint = NULL,
  stac_collection = NULL,
  crs = NULL,
  force_download = FALSE
)
```

## Arguments

| | |
|---|---|
| bb | A bounding box, provided either as a matrix (rows for "x", "y", columns for "min", "max") or as a vector ("xmin", "ymin", "xmax", "ymax"), in lat/lon coordinates (WGS84 coordinate reference system) of class bbox |
| dem_source | Source of the DEM:<br>• If "STAC" (default), DEM tiles are searched on a SpatioTemporal Asset Catalog (STAC) end point, then accessed and mosaicked to the area of interest |
| stac_endpoint | URL of the STAC API endpoint (only used if dem_source is "STAC"). For more info, see get_stac_asset_urls() |
| stac_collection | Identifier of the STAC collection to be queried (only used if dem_source is "STAC"). For more info, see get_stac_asset_urls() |
| crs | Coordinate reference system (CRS) which to transform the DEM to |
| force_download | Download data even if cached data is available |

## Value

DEM as a terra SpatRaster object

## Examples

```
# Get DEM with default values
bb <- get_osm_bb("Bucharest")
crs <- 31600  # National projected CRS

# Get DEM with default values
```

```
get_dem(bb)

# Get DEM from custom STAC endpoint
get_dem(bb,
        stac_endpoint = "some endpoint",
        stac_collection = "some collection")

# Specify CRS
get_dem(bb, crs = crs)
```

---

`get_dem_example_data` *Get example DEM data*

---

## Description

This function retrieves example Digital Elevation Model (DEM) data from the Zenodo data repository, and it can be used in examples and tests. The code used to generate the example dataset is available at https://github.com/CityRiverSpaces/CRiSpExampleData. Note that the example dataset is cached locally, so that subsequent calls to the function can load the example data from disk without having to re-download the data.

## Usage

```
get_dem_example_data(force_download = FALSE)
```

## Arguments

| | |
|---|---|
| `force_download` | Download data even if cached data is available |

## Value

An object of class `terra::SpatRaster` containing the DEM data.

## Examples

```
get_dem_example_data(force_download = TRUE)
```

get_osmdata *Retrieve OpenStreetMap data for a given location*

## Description

Retrieve OpenStreetMap data for a given location, including the city boundary, the river centreline and surface, the streets, the railways, and the buildings

## Usage

```
get_osmdata(
  city_name,
  river_name,
  network_buffer = NULL,
  buildings_buffer = NULL,
  city_boundary = TRUE,
  crs = NULL,
  force_download = FALSE
)
```

## Arguments

| | |
|---|---|
| city_name | The name of the city as character vector of length 1, case-sensitive. Required, no default. |
| river_name | The name of the river as character vector of length 1, case-sensitive. Required, no default. |
| network_buffer | Buffer distance in meters around the river to get the streets and railways, default is 0 means no network data will be downloaded |
| buildings_buffer | Buffer distance in meters around the river to get the buildings, default is 0 means no buildings data will be downloaded |
| city_boundary | A logical indicating if the city boundary should be retrieved. Default is TRUE. |
| crs | An integer or character vector of length one with the EPSG code for the projection. If no CRS is specified, the default is the UTM zone for the city. |
| force_download | Download data even if cached data is available |

## Value

A list with the retrieved OpenStreetMap data sets for the given location, as objects of class `sf::sfc`

## Examples

```
# Set parameters
city <- "Bucharest"
river <- "Dâmbovița"
crs <- "EPSG:31600"  # National projected CRS
```

```
# Get OSM data with defaults
get_osmdata(city_name = city, river_name = river)

# Get OSM data without city boundary
get_osmdata(city_name = city, river_name = river, city_boundary = FALSE)

# Use custom network buffer to get streets and railways
get_osmdata(city_name = city, river_name = river, network_buffer = 3500)

# Use custom buffer to get buildings
get_osmdata(city_name = city, river_name = river, buildings_buffer = 150)

# Use custom CRS
get_osmdata(city_name = city, river_name = river, crs = crs)

# Avoid getting OSM data from cache
get_osmdata(city_name = city, river_name = river, force_download = TRUE)
```

---

`get_osm_bb` *Get the bounding box of a city*

---

## Description

Get the bounding box of a city

## Usage

```
get_osm_bb(city_name)
```

## Arguments

| | |
|---|---|
| `city_name` | A character vector of length one |

## Value

A bbox object with the bounding box of the city

## Examples

```
get_osm_bb(city_name = "Bucharest")
```

## get_osm_buildings *Get OpenStreetMap buildings*

### Description

Get buildings from OpenStreetMap within a given buffer around a river.

### Usage

```
get_osm_buildings(aoi, crs = NULL, force_download = FALSE)
```

### Arguments

| | |
|---|---|
| aoi | Area of interest as sf object or bbox |
| crs | Coordinate reference system as EPSG code |
| force_download | Download data even if cached data is available |

### Value

An object of class sf::sfc_POLYGON

### Examples

```
bb <- get_osm_bb("Bucharest")
crs <- get_utm_zone(bb)
get_osm_buildings(aoi = bb, crs = crs)
```

## get_osm_city_boundary *Get the city boundary from OpenStreetMap*

### Description

This function retrieves the city boundary from OpenStreetMap based on a bounding box with the OSM tags "place:city" and "boundary:administrative". The result is filtered by the city name.

### Usage

```
get_osm_city_boundary(
  bb,
  city_name,
  crs = NULL,
  multiple = FALSE,
  force_download = FALSE
)
```

### Arguments

| | |
|---|---|
| bb | Bounding box of class bbox |
| city_name | A case-sensitive character vector of length 1 with the name of the city |
| crs | Coordinate reference system as EPSG code |
| multiple | A logical indicating if multiple city boundaries should be returned. By default, only the first one is returned. |
| force_download | Download data even if cached data is available |

### Value

An object of class sf::sfc_POLYGON or sf::sfc_MULTIPOLYGON with the city boundary

### Examples

```
bb <- get_osm_bb("Bucharest")
crs <- get_utm_zone(bb)
get_osm_city_boundary(bb = bb, city_name = "Bucharest", crs = crs)
```

---

## get_osm_example_data *Get example OSM data*

---

### Description

This function retrieves example OpenStreetMap (OSM) data from the Zenodo data repository, and it can be used in examples and tests. The code used to generate the example dataset is available at https://github.com/CityRiverSpaces/CRiSpExampleData. Note that the example dataset is cached locally, so that subsequent calls to the function can load the example data from disk without having to re-download the data.

### Usage

```
get_osm_example_data(force_download = FALSE)
```

### Arguments

| | |
|---|---|
| force_download | Download data even if cached data is available |

### Value

A list of sf objects containing the OSM data as sf::sfc objects.

### Examples

```
get_osm_example_data(force_download = TRUE)
```

## get_osm_railways *Get OpenStreetMap railways*

### Description

Get OpenStreetMap railways

### Usage

```
get_osm_railways(
  aoi,
  crs = NULL,
  railway_values = "rail",
  force_download = FALSE
)
```

### Arguments

| | |
|---|---|
| aoi | Area of interest as sf object or bbox |
| crs | A numeric vector of length one with the EPSG code of the CRS |
| railway_values | A case-insensitive character vector with the railway values to retrieve. |
| force_download | Download data even if cached data is available |

### Value

An object of class sf::sfc_LINESTRING

### Examples

```
bb <- get_osm_bb("Bucharest")
crs <- get_utm_zone(bb)
get_osm_railways(aoi = bb, crs = crs)
```

## get_osm_river *Get the river centreline and surface from OpenStreetMap*

### Description

Get the river centreline and surface from OpenStreetMap

### Usage

```
get_osm_river(bb, river_name, crs = NULL, force_download = FALSE)
```

### Arguments

| | |
|---|---|
| bb | Bounding box of class bbox |
| river_name | The name of the river as character vector of length 1, case-sensitive. |
| crs | Coordinate reference system as EPSG code |
| force_download | Download data even if cached data is available |

### Value

A list with the river centreline as object of class sf::sfc_LINESTRING or sf::sfc_MULTILINESTRING and river surface of class sf::sfc_POLYGON or sf::sfc_MULTIPOLYGON.

### Examples

```
bb <- get_osm_bb("Bucharest")
crs <- get_utm_zone(bb)
get_osm_river(bb = bb, river_name = "Dâmbovița", crs = crs,
              force_download = FALSE)
```

---

## get_osm_streets *Get OpenStreetMap streets*

---

### Description

Get OpenStreetMap streets

### Usage

```
get_osm_streets(aoi, crs = NULL, highway_values = NULL, force_download = FALSE)
```

### Arguments

| | |
|---|---|
| aoi | Area of interest as sf object or bbox. Required, no default. |
| crs | A numeric vector of length one with the EPSG code of the CRS |
| highway_values | A character vector with the highway values to retrieve. If left NULL, the function retrieves the following values: "motorway", "trunk", "primary", "secondary", "tertiary" |
| force_download | Download data even if cached data is available |

### Value

An object of class sf::sfc_LINESTRING

## Examples

```
# Set parameters
bb <- get_osm_bb("Bucharest")
crs <- 31600  # National projected CRS

# Get streets with default values
get_osm_streets(aoi = bb, crs = crs)

# Specify street categories to be retrieved
get_osm_streets(aoi = bb, crs = crs, highway_values = "primary")

# Ensure that data is not retrieved from cache
get_osm_streets(aoi = bb, crs = crs, force_download = FALSE)
```

---

# get_river_aoi *Get an area of interest (AoI) around a river, cropping to the bounding box of a city*

---

## Description

Get an area of interest (AoI) around a river, cropping to the bounding box of a city

## Usage

```
get_river_aoi(river, city_bbox, buffer_distance)
```

## Arguments

| | |
|---|---|
| river | A list with the river centreline and surface geometries |
| city_bbox | Bounding box of class bbox around the city |
| buffer_distance | A positive number representing the buffer size around the river in meters. The upper limit is unrestricted. |

## Value

An sf::sfc_POLYGON object in lat/lon coordinates

## Examples

```
bb <- get_osm_bb("Bucharest")
river <- get_osm_river(bb, "Dâmbovița")
get_river_aoi(river = river, city_bbox = bb, buffer_distance = 100)
```

## get_stac_asset_urls *Retrieve the URLs of all the assets intersecting a bbox from a STAC API*

### Description

Retrieve the URLs of all the assets intersecting a bbox from a STAC API

### Usage

```
get_stac_asset_urls(bb, endpoint = NULL, collection = NULL)
```

### Arguments

| | |
|---|---|
| bb | A bounding box, provided either as a matrix (rows for "x", "y", columns for "min", "max") or as a vector ("xmin", "ymin", "xmax", "ymax"), in lat/lon coordinates (WGS84 coordinate referece system) of class bbox |
| endpoint | URL of the STAC API endpoint. To be provided together with stac_collection, or leave blank to use defaults (see default_stac_dem) |
| collection | Identifier of the STAC collection to be queried. To be provided together with stac_endpoint, or leave blank to use defaults (see default_stac_dem) |

### Value

A list of URLs for the assets in the collection overlapping with the specified bounding box

### Examples

```
bb <- get_osm_bb("Bucharest")
get_stac_asset_urls(bb)

# Use non-default STAC API
get_stac_asset_urls(bb,
                    endpoint = "some endpoint",
                    collection = "some collection")
```

## get_utm_zone *Get the UTM zone of a spatial object*

### Description

Get the UTM zone of a spatial object

### Usage

```
get_utm_zone(x)
```

### Arguments

| | |
|---|---|
| x | Object in any class accepted by as_bbox() |

### Value

The EPSG code of the UTM zone

### Examples

```
# Get EPSG code for UTM zone of Bucharest
bb <- get_osm_bb("Bucharest")
get_utm_zone(bb)
```

---

## load_dem *Retrieve DEM data from a list of STAC assets*

---

### Description

Load DEM data from a list of tiles, crop and merge using a given bounding box to create a raster DEM for the specified region. Results are cached, so that new queries with the same input parameters will be loaded from disk.

### Usage

```
load_dem(bb, tile_urls, force_download = FALSE)
```

### Arguments

| | |
|---|---|
| bb | A bounding box, provided either as a matrix (rows for "x", "y", columns for "min", "max") or as a vector ("xmin", "ymin", "xmax", "ymax") of class bbox. |
| tile_urls | A list of tiles where to read the DEM data from |
| force_download | Download data even if cached data is available |

### Value

A DEM of class terra::SpatRaster, retrieved and retiled to the given bounding box

### Examples

```
bb <- get_osm_bb("Bucharest")
tile_urls <- get_stac_asset_urls(bb)
load_dem(bb = bb, tile_urls = tile_urls, force_download = TRUE)
```

## osmdata_as_sf *Retrieve OpenStreetMap data as sf object*

### Description

Query the Overpass API for a key:value pair within a given bounding box (provided as lat/lon coordiates). Results are cached, so that new queries with the same input parameters will be loaded from disk.

### Usage

```
osmdata_as_sf(key, value, aoi, force_download = FALSE)
```

### Arguments

| | |
|---|---|
| key | A case-insensitive character vector of length 1 with the key to filter the data |
| value | A case-insensitive character vector with the value(s) to filter the data |
| aoi | An area of interest, provided either as as sf object or "bbox" or as a vector ("xmin", "ymin", "xmax", "ymax") |
| force_download | Download data even if cached data is available |

### Value

An osmdata::osmdata object with the retrieved OpenStreetMap data

### Examples

```
bb <- get_osm_bb("Bucharest")
osmdata_as_sf(key = "highway",
              value = "motorway",
              aoi = bb,
              force_download = FALSE)
```

## reproject *Reproject a raster or vector dataset to the specified coordinate reference system (CRS)*

### Description

Reproject a raster or vector dataset to the specified coordinate reference system (CRS)

### Usage

```
reproject(x, crs, ...)
```

## Arguments

| | |
|---|---|
| x | Raster (SpatRaster) or vector (sf) object |
| crs | CRS to be projected to, provided as numeric, integer or logical vector of length one or sf::crs. If numeric, the value should be a positive number with unrestricted upper bound representing a valid EPSG code. |
| ... | Optional arguments for raster or vector reproject functions |

## Value

sf::sf, sf::sfc, or terra::SpatRaster object reprojected to specified CRS

## Examples

```
# Reproject a raster to EPSG:4326
r <- terra::rast(matrix(1:12, nrow = 3, ncol = 4), crs = "EPSG:32633")
reproject(r, 4326)
```

# Index