

Package: tmaptools (via r-universe)

October 9, 2026

Type Package

Title Thematic Map Tools

Version 3.3-0.9000

Description Set of tools for reading and processing spatial data. The aim is to supply the workflow to create thematic maps. This package also facilitates 'tmap', the package for visualizing thematic maps.

License GPL-3

Encoding UTF-8

Depends R (>= 3.5), methods

Imports sf (>= 0.9.2), lwgeom (>= 0.1-4), cli, stars (>= 0.4-1), units (>= 0.6-1), stats, XML

Suggests tmap, cols4all, rmapshaper, osmdata, OpenStreetMap, raster

URL <https://github.com/r-tmap/tmaptools>,
<https://r-tmap.github.io/tmaptools/>

BugReports <https://github.com/r-tmap/tmaptools/issues>

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

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

Date/Publication 2026-09-09 09:34:49 UTC

RemoteUrl <https://github.com/r-tmap/tmaptools>

RemoteRef HEAD

RemoteSha 14dd9705a17f13238ebec2d88578d2a750e480a8

Contents

tmaptools-package	2
approx_areas	4

approx_distances	5
bb	7
bb_poly	9
calc_densities	11
crop_shape	12
geocode_OSM	13
get_asp_ratio	15
get_neighbours	16
map_coloring	17
read_GPX	19
read_osm	20
rev_geocode_OSM	21
simplify_shape	23
Index	25

tmertools-package	<i>Thematic Map Tools</i>
-------------------	---------------------------

Description

This package offers a set of handy tool functions for reading and processing spatial data. The aim of these functions is to supply the workflow to create thematic maps, e.g. read shape files, set map projections, append data, calculate areas and distances, and query OpenStreetMap. The visualization of thematic maps can be done with the tmap package.

Details

This page provides a brief overview of all package functions.

Tool functions (shape)

approx_areas	Approximate area sizes of polygons
approx_distances	Approximate distances
bb	Create, extract or modify a bounding box
bb_poly	Convert bounding box to a polygon
get_asp_ratio	Get the aspect ratio of a shape object

Tool functions (colors)

<code>map_coloring</code>	Find different colors for adjacent polygons
---------------------------	---

Spatial transformation functions

<code>crop_shape</code>	Crop shape objects
<code>simplify_shape</code>	Simplify a shape

Input and output functions

<code>geocode_OSM</code>	Get a location from an address description
<code>read_GPX</code>	Read a GPX file
<code>read_osm</code>	Read Open Street Map data
<code>rev_geocode_OSM</code>	Get an address description from a location

Author(s)

Maintainer: Martijn Tennekes <mtennekes@gmail.com>

Authors:

- Martijn Tennekes <mtennekes@gmail.com>

See Also

Useful links:

- <https://github.com/r-tmap/tmertools>
- <https://r-tmap.github.io/tmertools/>
- Report bugs at <https://github.com/r-tmap/tmertools/issues>

approx_areas	<i>Approximate area sizes of the shapes</i>
--------------	---

Description

Approximate the area sizes of the polygons in real-world area units (such as sq km or sq mi), proportional numbers, or normalized numbers. Also, the areas can be calibrated to a prespecified area total. This function is a convenient wrapper around [st_area](#).

Usage

```
approx_areas(shp, target = "metric", total.area = NULL)
```

Arguments

shp	shape object, i.e., an sf or sp object.
target	target unit, one of "prop": Proportional numbers. In other words, the sum of the area sizes equals one. "norm": Normalized numbers. All area sizes are normalized to the largest area, of which the area size equals one. "metric" (default): Output area sizes will be either "km" (kilometer) or "m" (meter) depending on the map scale "imperial": Output area sizes will be either "mi" (miles) or "ft" (feet) depending on the map scale other: Predefined values are "km^2", "m^2", "mi^2", and "ft^2". Other values can be specified as well, in which case to is required). These units are the output units. See <code>orig</code> for the coordinate units used by the shape <code>shp</code>.
total.area	total area size of shp in number of target units (defined by <code>target</code>). Useful if the total area of the shp differs from a reference total area value. For "metric" and "imperial" units, please provide the total area in squared kilometers respectively miles.

Details

Note that the method of determining areas is an approximation, since it depends on the used projection and the level of detail of the shape object. Projections with equal-area property are highly recommended. See https://en.wikipedia.org/wiki/List_of_map_projections for equal area world map projections.

Value

Numeric vector of area sizes (class [units](#)).

See Also[approx_distances](#)**Examples**

```

if (require(tmap)) {
  data(NLD_muni)

  NLD_muni$area <- approx_areas(NLD_muni, total.area = 33893)

  tm_shape(NLD_muni) +
    tm_bubbles(size="area",
               size.legend = tm_legend(title = expression("Area in " * km^2)))

  # function that returns min, max, mean and sum of area values
  summary_areas <- function(x) {
    list(min_area=min(x),
          max_area=max(x),
          mean_area=mean(x),
          sum_area=sum(x))
  }

  # area of the polygons
  summary_areas(approx_areas(NLD_muni))

  # area of the polygons, adjusted corrected for a specified total area size
  summary_areas(approx_areas(NLD_muni, total.area=33893))

  # proportional area of the polygons
  summary_areas(approx_areas(NLD_muni, target = "prop"))

  # area in squared miles
  summary_areas(approx_areas(NLD_muni, target = "mi mi"))

  # area of the polygons when unprojected
  summary_areas(approx_areas(sf::st_transform(NLD_muni, crs = 4326)))
}

```

approx_distances	<i>Approximate distances</i>
------------------	------------------------------

Description

Approximate distances between two points or across the horizontal and vertical centerlines of a bounding box.

Usage

```
approx_distances(x, y = NULL, projection = NULL, target = NULL)
```

Arguments

x	object that can be coerced to a bounding box with bb , or a pair of coordintes (vector of two). In the former case, the distance across the horizontal and vertical centerlines of the bounding box are approximated. In the latter case, y is also required; the distance between points x and y is approximated.
y	a pair of coordintes, vector of two. Only required when x is also a pair of coordintes.
projection	projection code, needed in case x is a bounding box or when x and y are pairs of coordinates.
target	target unit, one of: "m", "km", "mi", and "ft".

Value

If y is specified, a list of two: unit and dist. Else, a list of three: unit, hdist (horizontal distance) and vdist (vertical distance).

See Also

[approx_areas](#)

Examples

```
## Not run:
if (require(tmap)) {
  data(NLD_prov)

  # North-South and East-West distances of the Netherlands
  approx_distances(NLD_prov)

  # Distance between Maastricht and Groningen
  p_maastricht <- geocode_OSM("Maastricht")$coords
  p_groningen <- geocode_OSM("Groningen")$coords
  approx_distances(p_maastricht, p_groningen, projection = 4326, target = "km")

  # Check distances in several projections
  sapply(c(3035, 28992, 4326), function(projection) {
    p_maastricht <- geocode_OSM("Maastricht", projection = projection)$coords
    p_groningen <- geocode_OSM("Groningen", projection = projection)$coords
    approx_distances(p_maastricht, p_groningen, projection = projection)
  })
}

## End(Not run)
```

bb *Bounding box generator*

Description

Swiss army knife for bounding boxes. Modify an existing bounding box or create a new bounding box from scratch. See details.

Usage

```
bb(
  x = NA,
  ext = NULL,
  cx = NULL,
  cy = NULL,
  width = NULL,
  height = NULL,
  xlim = NULL,
  ylim = NULL,
  relative = FALSE,
  asp.target = NULL,
  asp.limit = NULL,
  current.projection = NULL,
  projection = NULL,
  output = c("bbox", "matrix", "extent")
)
```

Arguments

- | | |
|-----|--|
| x | <p>One of the following:</p> <ul style="list-style-type: none"> • A shape from class sf, stars, sp, or raster. • A bounding box (st_bbox, Extent (raster package, which will no longer be supported in the future versions), numeric vector of 4 (default order: xmin, ymin, xmax, ymax), or a 2x2 matrix). • Open Street Map search query. The bounding is automatically generated by querying x from Open Street Map Nominatim. See geocode_OSM |
| ext | <p>Extension factor of the bounding box. If 1, the bounding box is unchanged. Values smaller than 1 reduces the bounding box, and values larger than 1 enlarges the bounding box. This argument is a shortcut for both width and height with <code>relative=TRUE</code>. If a negative value is specified, then the shortest side of the bounding box (so width or height) is extended with <code>ext</code>, and the longest side is extended with the same absolute value. This is especially useful for bounding boxes with very low or high aspect ratios.</p> |
| cx | <p>center x coordinate</p> |

<code>cy</code>	center y coordinate
<code>width</code>	width of the bounding box. These are either absolute or relative (depending on the argument <code>relative</code>).
<code>height</code>	height of the bounding box. These are either absolute or relative (depending on the argument <code>relative</code>).
<code>xlim</code>	limits of the x-axis. These are either absolute or relative (depending on the argument <code>relative</code>).
<code>ylim</code>	limits of the y-axis. See <code>xlim</code> .
<code>relative</code>	boolean that determines whether relative values are used for width, height, <code>xlim</code> and <code>ylim</code> or absolute. If <code>x</code> is unspecified, <code>relative</code> is set to "FALSE".
<code>asp.target</code>	target aspect ratio, which is width/height, of the returned bounding box.
<code>asp.limit</code>	maximum aspect ratio, which is width/height. Number greater than or equal to 1. For landscape bounding boxes, <code>1/asp.limit</code> will be used. The returned bounding box will have an aspect ratio between <code>1/asp.limit</code> and <code>asp.limit</code> .
<code>current.projection</code>	projection that corresponds to the bounding box specified by <code>x</code> .
<code>projection</code>	projection to transform the bounding box to.
<code>output</code>	output format of the bounding box, one of: • "bbox" a <code>sf::bbox</code> object, which is a numeric vector of 4: xmin, ymin, xmax, ymax. This representation used by the <code>sf</code> package. • "matrix" a 2 by 2 numeric matrix, where the rows correspond to x and y, and the columns to min and max. This representation used by the <code>sp</code> package. • "extent" an <code>raster::extent</code> object, which is a numeric vector of 4: xmin, xmax, ymin, ymax. This representation used by the <code>raster</code> package.

Details

An existing bounding box (defined by `x`) can be modified as follows:

- Using the extension factor `ext`.
- Changing the width and height with `width` and `height`. The argument `relative` determines whether relative or absolute values are used.
- Setting the x and y limits. The argument `relative` determines whether relative or absolute values are used.

A new bounding box can be created from scratch as follows:

- Using the extension factor `ext`.
- Setting the center coordinates `cx` and `cy`, together with the width and height.
- Setting the x and y limits `xlim` and `ylim`

Value

bounding box (see argument `output`)

See Also[geocode_OSM](#)**Examples**

```
if (require(tmap)) {  
  
  ## load shapes  
  data(NLD_muni)  
  data(World)  
  
  ## get bounding box (similar to sp's function bbox)  
  bb(NLD_muni)  
  
  ## extent it by factor 1.10  
  bb(NLD_muni, ext=1.10)  
  
  ## double the width  
  bb(NLD_muni, width=2, relative = TRUE)  
  
  ## crop both dimensions from 0.25 to 0.75  
  bb(NLD_muni, xlim=c(.25, .75), ylim=c(.25, .75), relative = TRUE)  
  
  ## extent it such that aspect ratio is 1  
  bb(NLD_muni, asp.target = 1)  
  
  ## convert to longlat (EPSG 4326)  
  bb(NLD_muni, projection=4326)  
}  
  
## Not run:  
if (require(tmap)) {  
  bb("Limburg", projection = 28992)  
  bb_italy <- bb("Italy", projection = "+proj=eck4")  
  
  tm_shape(World, bbox=bb_italy) + tm_polygons()  
  # shorter alternative: tm_shape(World, bbox="Italy") + tm_polygons()  
}  
## End(Not run)
```

bb_poly*Convert bounding box to a spatial polygon*

Description

Convert bounding box to a spatial ([sfc](#)) object . Useful for plotting (see example). The function `bb_earth` returns a spatial polygon of the 'boundaries' of the earth, which can also be done in other projections (if a feasible solution exists).

Usage

```
bb_poly(x, steps = 100, stepsize = NA, projection = NULL)
```

```
bb_earth(
  projection = NULL,
  stepsize = 1,
  earth.datum = 4326,
  bbx = c(-180, -90, 180, 90),
  buffer = 1e-06
)
```

Arguments

x	object that can be coerced to a bounding box with bb
steps	number of intermediate points along the shortest edge of the bounding box. The number of intermediate points along the longest edge scales with the aspect ratio. These intermediate points are needed if the bounding box is plotted in another projection.
stepsize	stepsize in terms of coordinates (usually meters when the shape is projected and degrees of longlat coordinates are used). If specified, it overrules steps
projection	projection in which the coordinates of x are provided. For <code>bb_earth</code> , projection is the projection in which the bounding box is returned (if possible).
earth.datum	Geodetic datum to determine the earth boundary. By default EPSG 4326.
bbx	boundig box of the earth in a vector of 4 values: min longitude, max longitude, min latitude, max latitude. By default <code>c(-180, 180, -90, 90)</code> . If for some projection, a feasible solution does not exist, it may be wise to choose a smaller bbx, e.g. <code>c(-180, 180, -88, 88)</code> . However, this is also automatically done with the next argument, <code>buffer</code> .
buffer	In order to determine feasible earth bounding boxes in other projections, a buffer is used to decrease the bounding box by a small margin (default <code>1e-06</code>). This value is subtracted from each the bounding box coordinates. If it still does not result in a feasible bounding box, this procedure is repeated 5 times, where each time the buffer is multiplied by 10. Set <code>buffer=0</code> to disable this procedure.

Value

[sfc](#) object

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {
  data(NLD_muni)

  current.mode <- tmap_mode("view")
  qtm(bb_poly(NLD_muni))

  # restore mode
}
```

```

    tmap_mode(current.mode)
  }

```

calc_densities

Calculate densities

Description

Transpose quantitative variables to density variables, which are often needed for choropleths. For example, the colors of a population density map should correspond population density counts rather than absolute population numbers.

Usage

```

calc_densities(
  shp,
  var,
  target = "metric",
  total.area = NULL,
  suffix = NA,
  drop = TRUE
)

```

Arguments

shp	a shape object, i.e., an sf object.
var	name(s) of a quality variable name contained in the shp data
target	the target unit, see approx_areas . Density values are calculated in $\text{var} / \text{target}^2$.
total.area	total area size of shp in number of target units (defined by unit), approx_areas .
suffix	character that is appended to the variable names. The resulting names are used as column names of the returned data.frame. By default, <code>_sq_<target></code> , where target corresponds to the target unit, e.g. <code>_sq_km</code>
drop	boolean that determines whether an one-column data-frame should be returned as a vector

Value

Vector or data.frame (depending on whether `length(var)==1` with density values.

Examples

```

if (require(tmap) && packageVersion("tmap") >= "3.99") {
  data(NLD_muni)

  NLD_muni_pop_per_km2 <- calc_densities(NLD_muni,
    target = "km km", var = c("population", "dwelling_total"))
  NLD_muni <- sf::st_sf(data.frame(NLD_muni, NLD_muni_pop_per_km2))
}

```

```

tm_shape(NLD_muni) +
  tm_polygons(
    fill = c("population_km.2", "dwelling_total_km.2"),
    fill.legend =
      list(
        tm_legend(expression("Population per " * km^2)),
        tm_legend(expression("Dwellings per " * km^2))) +
  tm_facets(free.scales = TRUE) +
    tm_layout(panel.show = FALSE)
}

```

crop_shape

Crop shape object

Description

Crop a shape object (from class [sf](#), [stars](#), [sp](#), or [raster](#)). A shape file *x* is cropped, either by the bounding box of another shape *y*, or by *y* itself if it is a [SpatialPolygons](#) object and `polygon = TRUE`.

Usage

```
crop_shape(x, y, polygon = FALSE, ...)
```

Arguments

<code>x</code>	shape object, i.e. an object from class sf , stars , sp , or raster .
<code>y</code>	bounding box, an st_bbox , extent (raster package), or a shape object from which the bounding box is extracted (unless <code>polygon</code> is <code>TRUE</code> and <i>x</i> is an sf object).
<code>polygon</code>	should <i>x</i> be cropped by the polygon defined by <i>y</i> ? If <code>FALSE</code> (default), <i>x</i> is cropped by the bounding box of <i>x</i> . Polygon cropping only works when <i>x</i> is a spatial object and <i>y</i> is a SpatialPolygons object.
<code>...</code>	not used anymore

Details

This function is similar to `crop` from the [raster](#) package. The main difference is that `crop_shape` also allows to crop using a polygon instead of a rectangle.

Value

cropped shape, in the same class as *x*

See Also

[bb](#)

Examples

```

if (require(tmap) && packageVersion("tmap") >= "3.99") {
  data(World, NLD_muni, land, metro)

  #land_NLD <- crop_shape(land, NLD_muni)

  #qtm(land_NLD, raster="trees", style="natural")

  metro_Europe <- crop_shape(metro, World[World$continent == "Europe", ], polygon = TRUE)

  qtm(World) +
    tm_shape(metro_Europe) +
    tm_bubbles("pop2010",
              col="red",
              size.legend = tm_legend("European cities")) +
    tm_legend(frame=TRUE)
}

```

geocode_OSM

*Geocodes a location using OpenStreetMap Nominatim***Description**

Geocodes a location (based on a search query) to coordinates and a bounding box. Similar to geocode from the ggmap package. It uses OpenStreetMap Nominatim. For processing large amount of queries, please read the usage policy (<https://operations.osmfoundation.org/policies/nominatim/>).

Usage

```

geocode_OSM(
  q,
  projection = NULL,
  return.first.only = TRUE,
  keep.unfound = FALSE,
  details = FALSE,
  as.data.frame = NA,
  as.sf = FALSE,
  geometry = c("point", "bbox"),
  server = "https://nominatim.openstreetmap.org"
)

```

Arguments

q	a character (vector) that specifies a search query. For instance "India" or "CBS Weg 11, Heerlen, Netherlands".
projection	projection in which the coordinates and bounding box are returned. See st_crs for details. By default latitude longitude coordinates (EPSG 4326).

<code>return.first.only</code>	Only return the first result
<code>keep.unfound</code>	Keep list items / data.frame rows with NAs for unfound search terms. By default FALSE
<code>details</code>	provide output details, other than the point coordinates and bounding box
<code>as.data.frame</code>	Return the output as a data.frame. If FALSE, a list is returned with at least two items: "coords", a vector containing the coordinates, and "bbox", the corresponding bounding box. By default false, unless q contains multiple queries. If <code>as.sf = TRUE</code> (see below), <code>as.data.frame</code> will set to TRUE.
<code>as.sf</code>	Return the output as <code>sf</code> object. If TRUE, <code>return.first.only</code> will be set to TRUE. Two geometry columns are added: <code>bbox</code> and <code>point</code> . The argument <code>geometry</code> determines which of them is set to the default geometry.
<code>geometry</code>	When <code>as.sf</code> , this argument determines which column (<code>bbox</code> or <code>point</code>) is set as geometry column. Note that the geometry can be changed afterwards with <code>st_set_geometry</code> .
<code>server</code>	OpenStreetMap Nominatim server name. Could also be a local OSM Nominatim server.

Value

If `as.sf` then a `sf` object is returned. Else, if `as.data.frame`, then a data.frame is returned, else a list.

See Also

[rev_geocode_OSM](#), [bb](#)

Examples

```
## Not run:
if (require(tmap)) {
  geocode_OSM("India")
  geocode_OSM("CBS Weg 1, Heerlen")
  geocode_OSM("CBS Weg 1, Heerlen", projection = 28992)

  data(metro)

  # sample 5 cities from the metro dataset
  five_cities <- metro[sample(length(metro), 5), ]

  # obtain geocode locations from their long names
  five_cities_geocode <- geocode_OSM(five_cities$name_long, as.sf = TRUE)

  # change to interactive mode
  current.mode <- tmap_mode("view")

  # plot metro coordinates in red and geocode coordinates in blue
  # zoom in to see the differences
  tm_shape(five_cities) +
```

```

    tm_dots(col = "blue") +
    tm_shape(five_cities_geocode) +
    tm_dots(col = "red")

    # restore current mode
    tmap_mode(current.mode)
  }

## End(Not run)

```

get_asp_ratio	<i>Get aspect ratio</i>
---------------	-------------------------

Description

Get the aspect ratio of a shape object, a [tmap](#) object, or a bounding box

Usage

```
get_asp_ratio(x, is.projected = NA, width = 700, height = 700, res = 100)
```

Arguments

<code>x</code>	A shape from class sf , stars , <code>sp</code> , or <code>Raster</code> , a bounding box (that can be coerced by bb), or a tmap object.
<code>is.projected</code>	Logical that determined wether the coordinates of <code>x</code> are projected (<code>TRUE</code>) or longitude latitude coordinates (<code>FALSE</code>). By default, it is determined by the coordinates of <code>x</code> .
<code>width</code>	See details; only applicable if <code>x</code> is a tmap object.
<code>height</code>	See details; only applicable if <code>x</code> is a tmap object.
<code>res</code>	See details; only applicable if <code>x</code> is a tmap object.

Details

The arguments `width`, `height`, and `res` are passed on to [png](#). If `x` is a [tmap](#) object, a temporarily png image is created to calculate the aspect ratio of a [tmap](#) object. The default size of this image is 700 by 700 pixels at 100 dpi.

Value

aspect ratio

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {  
  data(World)  
  
  get_asp_ratio(World)  
  
  get_asp_ratio(bb(World))  
  
  tm <- qtm(World)  
  get_asp_ratio(tm)  
}  
  
## Not run:  
  get_asp_ratio("Germany") #note: bb("Germany") uses geocode_OSM("Germany")  
  
## End(Not run)
```

get_neighbours

Get neighbours list from spatial objects

Description

Get neighbours list from spatial objects. The output is similar to the function `poly2nb` of the `spdep` package with `queen = FALSE`, but uses `sf` instead of `sp`. Two polygons are considered neighbours only when their boundaries share a line segment (an edge), not merely a point.

Usage

```
get_neighbours(x)
```

Arguments

x	a shape object, i.e., a <code>sf</code> object or a <code>SpatialPolygons(DataFrame)</code> (<code>sp</code> package).
---	---

Value

A list where the items correspond to the features. Each item is a vector of neighbours.

map_coloring	<i>Map coloring</i>
--------------	---------------------

Description

Color the polygons of a map such that adjacent polygons have different colors. This function returns the color indices

Usage

```
map_coloring(x, algorithm = "dsatur", ncols = 8, permutation = 0, ...)
```

Arguments

x	Either a shape (i.e. a sf or <code>SpatialPolygons(DataFrame)</code> (sp package) object), or an adjacency list.
algorithm	either "dsatur" (default), the DSatur heuristic backed by a tabu search, or "greedy", the heuristic on its own, which is faster but may use more colors than needed. See details.
ncols	target number of colors, all of which are used about equally often. By default 8. Raised to 4 if set lower, and raised further, with a message, if more colors are needed for a valid coloring.
permutation	number that selects one of the many colorings a map has. By default 0. Any other whole number gives another one, so trying 1, 2, 3, ... is a way to look for a map that pleases. It changes both which polygons share a color and which color each group is given, and every permutation is equally valid and equally evenly spread.
...	to catch deprecated arguments <code>palette</code> , <code>contrast</code> and <code>minimize</code> . See details.

Details

As of tmaptools 3.3, the deprecated color functions `get_brewer_pal` and `palette_explorer`, have been removed. These have been replaced `c4a` and `c4a_gui` respectively from the package `cols4all`. Therefore, `map_coloring` will return color indices and will ignore the input arguments `palette` and `contrast`. See example.

Adjacency is determined by shared boundary lines (as in "rook" contiguity), not by shared points alone. This matters for e.g. Voronoi tessellations, where three or more cells can meet at a single vertex without actually sharing an edge; treating such point-touches as adjacency can force the use of unnecessarily many colors.

`ncols` is the target number of colors, with a minimum of 4 (the four-color theorem guarantees that 4 colors suffice for any planar map). All `ncols` of them are used, about equally often, so that no color dominates the map. Only when `ncols` colors are too few for a valid coloring is the number increased, in which case the user is informed via [cli_inform](#). Use `ncols = 4` for a map that uses as few colors as possible.

Evening out the colors is possible because a polygon may take any color that none of its neighbours has. So as long as one color is used at least two polygons more often than another, some polygon can be moved from the first to the second. A color that the coloring itself did not need is simply one that is used zero times, and is filled up the same way; this is what spreads a coloring that needs only 4 colors over all `ncols` of them.

A map has many colorings, and which one comes out is decided by choices between equally good options: which color a polygon is given, which polygon is moved to even out the colors, and which colors are handed to which group in the end. The argument `permutation` settles those choices, so that each number gives a different, equally valid map. Raise it until the map looks good.

As of `tmaptools` 3.4, the argument `minimize` has been removed: `ncols` now determines the number of colors on its own. The result no longer depends on the random seed, and the random number generator of the caller is left untouched, so that repeated plots of the same map keep the same colors. To go looking for a nicer looking map, vary `permutation` instead of the seed.

The `"dsatur"` algorithm first runs the DSatur heuristic (Brelaz, 1979), which colors the most constrained polygon at each step: the one with the largest number of distinct colors among its already colored neighbours, ties broken by the number of neighbours. Since this is a heuristic, it may fail even when a valid coloring exists; on maps of contiguous regions it typically needs one color more than necessary. It is therefore followed by a tabu search (Hertz and de Werra, 1987), which starts from a coloring that uses `ncols` colors but may have neighbours sharing one, and then repeatedly recolors a polygon involved in such a clash until none are left. Two reductions keep this cheap: polygons with fewer than `ncols` neighbours are set aside and colored last (a free color is then always available), and the rest are split into groups that are not connected to each other and searched separately. Use `algorithm = "greedy"` to skip the tabu search.

Value

A vector of color indices.

References

- Brelaz, D. (1979) New methods to color the vertices of a graph. *Communications of the ACM* 22(4), 251-256.
- Hertz, A. and de Werra, D. (1987) Using tabu search techniques for graph coloring. *Computing* 39(4), 345-351.

Examples

```
if (require(tmap) && require(cols4all)) {
  data(World)

  ## using cols4all directly
  indices <- map_coloring(World)
  pal <- c4a("brewer.set2", n = max(indices))
  World$color = pal[indices]
  tm_shape(World) +
    tm_polygons("color", fill.scale = tm_scale_asis()) +
    tm_crs("auto")

  # using map_coloring via "MAP_COLORS" in tmap
```

```

tm_shape(World) +
  tm_polygons("MAP_COLORS", tm_scale(values = "brewer.set2")) +
  tm_crs("auto")

# other example
data(NLD_prov, NLD_muni)
tm_shape(NLD_prov) +
  tm_fill("name",
    fill.legend = tm_legend_hide()) +
tm_shape(NLD_muni) +
  tm_polygons("MAP_COLORS",
    fill_alpha = .25,
    fill.scale = tm_scale(values = "brewer.greys")) +
tm_shape(NLD_prov) +
  tm_borders(lwd=2) +
  tm_text("name", options = opt_tm_text(shadow = TRUE)) +
  tm_title("Dutch provinces and\nmunicipalities", bg.color="white")
}

```

read_GPX

*Read GPX file***Description**

Read a GPX file. By default, it reads all possible GPX layers, and only returns shapes for layers that have any features.

Usage

```

read_GPX(
  file,
  layers = c("waypoints", "routes", "tracks", "route_points", "track_points"),
  remove.empty.layers = TRUE,
  as.sf = TRUE
)

```

Arguments

<code>file</code>	a GPX filename (including directory)
<code>layers</code>	vector of GPX layers. Possible options are "waypoints", "tracks", "routes", "track_points", "route_points". By default, all those layers are read.
<code>remove.empty.layers</code>	should empty layers (i.e. with 0 features) be removed from the list?
<code>as.sf</code>	not used anymore

Details

Note that this function returns [sf](#) objects, but still uses methods from `sp` and `rgdal` internally.

Value

a list of sf objects, one for each layer

read_osm	<i>Read Open Street Map data</i>
----------	----------------------------------

Description

Read Open Street Map data. OSM tiles are read and returned as a spatial raster. Vectorized OSM data is not supported anymore (see details).

Usage

```
read_osm(
  x,
  zoom = NULL,
  type = "osm",
  minNumTiles = NULL,
  mergeTiles = NULL,
  use.colortable = FALSE,
  ...
)
```

Arguments

x	object that can be coerced to a bounding box with bb (e.g. an existing bounding box or a shape). In the first case, other arguments can be passed on to bb (see ...). If an existing bounding box is specified in projected coordinates, please specify <code>current.projection</code> .
zoom	passed on to openmap . Only applicable when <code>raster=TRUE</code> .
type	tile provider, by default "osm", which corresponds to OpenStreetMap Mapnik. See openmap for options. Only applicable when <code>raster=TRUE</code> .
minNumTiles	passed on to openmap Only applicable when <code>raster=TRUE</code> .
mergeTiles	passed on to openmap Only applicable when <code>raster=TRUE</code> .
use.colortable	should the colors of the returned raster object be stored in a colortable ? If FALSE, a RasterStack is returned with three layers that correspond to the red, green and blue values between 0 and 255.
...	arguments passed on to bb .

Details

As of version 2.0, `read_osm` cannot be used to read vectorized OSM data anymore. The reason is that the package that was used under the hood, `osmar`, has some limitations and is not actively maintained anymore. Therefore, we recommend the package `osmdata`. Since this package is very user-friendly, there was no reason to use `read_osm` as a wrapper for reading vectorized OSM data.

Value

The output of `read_osm` is a `raster` object.

Examples

```
## Not run:
if (require(tmap)) {
  ##### Choropleth with OSM background

  # load Netherlands shape
  data(NLD_muni)

  # read OSM raster data
  osm_NLD <- read_osm(NLD_muni, ext=1.1)

  # plot with regular tmap functions
  tm_shape(osm_NLD) +
    tm_rgb() +
    tm_shape(NLD_muni) +
    tm_polygons("population", convert2density=TRUE, style="kmeans", alpha=.7, palette="Purples")

  ##### A close look at the building of Statistics Netherlands in Heerlen

  # create a bounding box around the CBS (Statistics Netherlands) building
  CBS_bb <- bb("CBS Weg 11, Heerlen", width=.003, height=.002)

  # read Microsoft Bing satellite and OpenCycleMap OSM layers
  CBS_osm1 <- read_osm(CBS_bb, type="bing")
  CBS_osm2 <- read_osm(CBS_bb, type="opencyclemap")

  # plot OSM raster data
  qtm(CBS_osm1)
  qtm(CBS_osm2)

}

## End(Not run)
```

rev_geocode_OSM

Reverse geocodes a location using OpenStreetMap Nominatim

Description

Reverse geocodes a location (based on spatial coordinates) to an address. It uses OpenStreetMap Nominatim. For processing large amount of queries, please read the usage policy (<https://operations.osmfoundation.org/policies/nominatim/>).

Usage

```
rev_geocode_OSM(
  x,
  y = NULL,
  zoom = NULL,
  projection = 4326,
  as.data.frame = NA,
  server = "https://nominatim.openstreetmap.org",
  params = NULL
)
```

Arguments

x	x coordinate(s), or a spatial points object (sf or SpatialPoints)
y	y coordinate(s)
zoom	zoom level
projection	projection in which the coordinates x and y are provided.
as.data.frame	return as data.frame (TRUE) or list (FALSE). By default a list, unless multiple coordinates are provided.
server	OpenStreetMap Nominatim server name. Could also be a local OSM Nominatim server.
params	Additional parameters to pass to server. (must start with &), ex: "&accept-language=en" to return english rather than local language results.

Value

A data frame or a list with all attributes that are contained in the search result

See Also

[geocode_OSM](#)

Examples

```
## Not run:
if (require(tmap)) {
  data(metro)

  # sample five cities from metro dataset
  set.seed(1234)
  five_cities <- metro[sample(length(metro), 5), ]

  # obtain reverse geocode address information
  addresses <- rev_geocode_OSM(five_cities, zoom = 6)
  five_cities <- sf::st_sf(data.frame(five_cities, addresses))

  # change to interactive mode
  current.mode <- tmap_mode("view")
}
```

```

    tm_shape(five_cities) +
      tm_markers(text="name")

    # restore current mode
    tmap_mode(current.mode)
  }

## End(Not run)

```

simplify_shape

*Simplify shape***Description**

Simplify a shape consisting of polygons or lines. This can be useful for shapes that are too detailed for visualization, especially along natural borders such as coastlines and rivers. The number of coordinates is reduced.

Usage

```
simplify_shape(shp, fact = 0.1, keep.units = FALSE, keep.subunits = FALSE, ...)
```

Arguments

shp	an sf or sfc object.
fact	simplification factor, number between 0 and 1 (default is 0.1)
keep.units	prevent small polygon features from disappearing at high simplification (default FALSE)
keep.subunits	should multipart polygons be converted to singlepart polygons? This prevents small shapes from disappearing during simplification if keep.units = TRUE. Default FALSE
...	other arguments passed on to the underlying function ms_simplify (except for the arguments input, keep, keep_shapes and explode)

Details

This function is a wrapper of [ms_simplify](#). In addition, the data is preserved. Also [sf](#) objects are supported.

Value

[sf](#) object

Examples

```
## Not run:
if (require(tmap)) {
  data(World)

  # show different simplification factors
  tm1 <- qtm(simplify_shape(World, fact = 0.05), title="Simplify 0.05")
  tm2 <- qtm(simplify_shape(World, fact = 0.1), title="Simplify 0.1")
  tm3 <- qtm(simplify_shape(World, fact = 0.2), title="Simplify 0.2")
  tm4 <- qtm(simplify_shape(World, fact = 0.5), title="Simplify 0.5")
  tmap_arrange(tm1, tm2, tm3, tm4)

  # show different options for keeping smaller (sub)units
  tm5 <- qtm(simplify_shape(World, keep.units = TRUE, keep.subunits = TRUE),
    title="Keep units and subunits")
  tm6 <- qtm(simplify_shape(World, keep.units = TRUE, keep.subunits = FALSE),
    title="Keep units, ignore small subunits")
  tm7 <- qtm(simplify_shape(World, keep.units = FALSE),
    title="Ignore small units and subunits")
  tmap_arrange(tm5, tm6, tm7)
}

## End(Not run)
```

Index

- * **GIS**
 - tmertools-package, 2
- * **densities**
 - calc_densities, 11
- * **spatial data**
 - tmertools-package, 2
- * **thematic maps**
 - tmertools-package, 2

approx_areas, 2, 4, 6, 11
approx_distances, 2, 5, 5

bb, 2, 6, 7, 10, 12, 14, 15, 20
bb_earth (bb_poly), 9
bb_poly, 2, 9

calc_densities, 11
cli_inform, 17
colortable, 20
crop_shape, 3, 12

geocode_OSM, 3, 7, 9, 13, 22
get_asp_ratio, 2, 15
get_neighbours, 16

map_coloring, 3, 17
ms_simplify, 23

openmap, 20

png, 15

raster, 21
read_GPX, 3, 19
read_osm, 3, 20
rev_geocode_OSM, 3, 14, 21

sf, 4, 7, 8, 11, 12, 14–17, 19, 22, 23
sfc, 9, 10, 23
simplify_shape, 3, 23
SpatialPoints, 22

st_area, 4
st_bbox, 7, 12
st_crs, 13
st_set_geometry, 14
stars, 7, 12, 15

tmap, 15
tmertools (tmertools-package), 2
tmertools-package, 2

units, 4