Package 'tmap'

December 11, 2015

License GPL-3

Title Thematic Maps

Type Package

LazyLoad yes

Description Thematic maps are geographical maps in which statistical data are visualized. This package offers a flexible, layer-based, way to create thematic maps, such as choropleths and bubble maps.

Version 1.2

Date 2015-12-11

Depends R (>= 3.0),

Imports methods, sp, raster (>= 2.4-30), grid, gridBase, gridSVG, rgdal, rgeos, classInt, htmlwidgets, KernSmooth, osmar, RColorBrewer, spdep, svgPanZoom, XML,

Suggests dplyr, knitr, OpenStreetMap

VignetteBuilder knitr

RoxygenNote 5.0.1

NeedsCompilation no

Author Martijn Tennekes [aut, cre],
Joel Gombin [ctb],
Kent Russell [ctb],
Richard Zijdeman [ctb]

Maintainer Martijn Tennekes <mtennekes@gmail.com>

Repository CRAN

Date/Publication 2015-12-11 15:45:40

R topics documented:

tmap-package *Thematic Maps*

Description

Thematic maps are geographical maps in which statistical data are visualized. This package offers a flexible, layer-based, way to create thematic maps, such as choropleths and bubble maps. It is based on the grammar of graphics, and resembles the syntax of ggplot2.

Details

This page provides a brief overview of all package functions. See `vignette("tmap-nutshell")` for a short manual with examples.

Quick plotting method

qtm	To plot a thematic map

Main plotting method

Shape specification:

tm_shape	To specify a shape object

Aesthetics layers:

tm_fill	To create a polygon layer (without borders)
tm_borders	To create polygon borders
tm_polygons	To create a polygon layer with borders
tm_bubbles	To create a layer of bubbles
tm_dots	To create a layer of dots
tm_lines	To create a layer of lines
tm_raster	To create a raster layer
tm_text	To create a layer of text labels

Facetting (small multiples)

tm_facets	To define facets

Attributes:

tm_grid	To create grid lines
tm_scale_bar	To create a scale bar
tm_compass	To create a map compass
tm_credits	To create a text for credits

Layout element:

tm_layout	To adjust the layout (main function)
tm_legend	Shortcut to adjust the legend

Handy tool functions

bb	To create, extract or modify a bounding box
get_asp_ratio	To get the aspect ratio of a shape object
get_IDs	To get ID values of a shape object
append_data	To append a data frame to a shape object
approx_areas	To approximate area sizes of polygons
calc_densities	To calculate density values
get_projection	To get the map projection
set_projection	To set the map projection
split	To split a shape object
sbind	To bind shape objects
map_coloring	To color polygons with different colors for adjacent polygons

Generate spatial objects

smooth_map	To create a smooth map (raster, contour lines and dasymetric polygons)
smooth_raster_cover	To create a smooth cover of a raster object
sample_dots	To sample dots from polygons
points_to_raster	To bin spatial points to a raster
poly_to_raster	To convert polygons to a raster

Input functions

read_shape	To read a shape object
read_GPX	To read a GPX file
read_osm	To read Open Street Map data

Output functions

itmap	Interactive tmap widget
animation_tmap	Create an animation
save_tmap	To save thematic maps
write_shape	To write a shape object

Spatial datasets

World	World country data (spatial polygons)
Europe	European country data (spatial polygons)
NLD_prov	Netherlands province data (spatial polygons)
NLD_muni	Netherlands municipal data (spatial polygons)
metro	Metropolitan araes (spatial points)
rivers	Rivers (spatial lines)
land	Global land cover (spatial grid)

Author(s)

Martijn Tennekes <mtennekes@gmail.com>

See Also

```
vignette("tmap-nutshell")
```

+.tmap *Stacking of tmap elements*

Description

The plus operator allows you to stack tmap-elements, and groups of tmap-elements.

Usage

```
## S3 method for class 'tmap'
e1 + e2
```

Arguments

e1	first tmap-element
e2	second tmap-element

See Also

tmap-element and vignette("tmap-nutshell")

animation_tmap *Create animation*

Description

Create a gif or mpeg animation from a tmap plot. The free tool ImageMagick is required.

Usage

```
animation_tmap(tm, filename = "animation.gif", width = 1000,
  height = 1000, delay = 40)
```

Arguments

tm	tmap object. In order to create a series of tmap plots, which will be the frames of the animation, it is important to set nrow and ncol in tm_facets, for otherwise a small multiples plot is generated. Commonly, where one map is shown at a time, both nrow and ncol are set to 1.
filename	filename of the video (should be a .gif or .mpg file)
width	width of the animation file (in pixels)
height	height of the animation file (in pixels)
delay	delay time between images (in 1/100th of a second)

Note

Not only tmap plots are supported, but any series of R plots.

Examples

```
## Not run:
require(dplyr)
data(World, metro, Europe)

(tm_shape(Europe) +
tm_fill("yellow") +
tm_borders() +
tm_facets(by = "name", nrow=1,ncol=1) +
tm_layout(scale=2, outer.margins=0,asp=0)) %>%
animation_tmap(filename="European countries.gif", width=1200, height=800, delay=100)

(tm_shape(World) +
  tm_polygons() +
tm_shape(metro) +
  tm_bubbles(paste0("pop", seq(1970, 2030, by=10)), border.col = "black", border.alpha = .5) +
tm_facets(free.scales.bubble.size = FALSE, nrow=1,ncol=1) +
tm_format_World(scale=2, outer.margins=0,asp=0)) %>%
animation_tmap(filename="World population.gif", width=1200, height=550, delay=100)

## End(Not run)
```

---

append_data *Append data to a shape object*

Description

Data, in the format of a data.frame, is appended to a shape object. This is either done by a right join where keys are specified for both data and shape, or by fixed order.

Usage

```
append_data(shp, data, key.shp = NULL, key.data = NULL,
  ignore.duplicates = FALSE, ignore.na = FALSE,
  fixed.order = is.null(key.data) && is.null(key.shp))
```

Arguments

shp	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame) 4. SpatialGrid(DataFrame) 5. SpatialPixels(DataFrame)

data	data.frame
key.shp	variable name of shp map data to be matched with key.data. If not specified, and fixed.order is FALSE, the ID's of the polygons/lines/points are taken.
key.data	variable name of data to be matched with key.shp. If not specified, and fixed.order is FALSE, the row names of data are taken.
ignore.duplicates	should duplicated keys in data be ignored? (FALSE by default)
ignore.na	should NA values in key.data and key.shp be ignored? (FALSE by default)
fixed.order	should the data be append in the same order as the shapes in shp?

Value

Shape object with appended data.

Examples

```
## Not run:
data(Europe)

f <- tempfile()
download.file("http://kejser.org/wp-content/uploads/2014/06/Country.csv", destfile = f)
domain_codes <- read.table(f, header=TRUE, sep="|")
unlink(f)

domain_codes <- subset(domain_codes, select = c("Alpha3Code", "TopLevelDomain"))
domain_codes$Alpha3Code <- toupper(domain_codes$Alpha3Code)

Europe <- append_data(Europe, domain_codes, key.shp = "iso_a3", key.data = "Alpha3Code",
    ignore.na = TRUE)

qtm(Europe, text="TopLevelDomain")

## End(Not run)
```

---

approx_areas *Approximate area sizes of the shapes*

Description

Approximate the area sizes of the polygons either in 1) absolute numbers based on the polygon coordinates, 2) proportional numbers, 3) normalized numbers and 4) squared units (e.g. kilometers).

Usage

```
approx_areas(shp, unit = "km", unit.size = 1000, total.area = NA)
```

Arguments

shp	shape object, i.e., a SpatialPolygons(DataFrame)
unit	one of "abs": Absolute numbers based on polygon coordinates. "prop": Proportional numbers. In other words, the total 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. **other:** For instance, "km", "m", or "miles". For this method, total.area or unit.size is required. In this case, the area sizes are returned in squared units, e.g., in squared kilometers. The default method is "km".
unit.size	size of the unit in terms of coordinate units. The coordinate system of many projections is approximately in meters while thematic maps typically range many kilometers, so by default unit="km" and unit.size=1000 (meaning 1 kilometer equals 1000 coordinate units).
total.area	total area size of shp in number of squared units (by default kilometers). Useful if the total area of the shp differs from a reference total area value.

Details

To approximate the sizes in squared units, either the total.area or the unit.size is required. Note that this method is an approximation, since it depends on the used projection and the level of detail of the SpatialPolygons object. Projections with equal-area property are highly recommended.

Value

Numeric vector of area sizes.

Examples

```
data(NLD_muni)

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

tm_shape(NLD_muni) +
tm_bubbles(size="area", title.size="Area in km2")
```

---

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,
  current.projection = NULL, projection = NULL)
```

Arguments

x	One of the following: • A shape (from class Spatial or Raster) • A bounding box (either 2 by 2 matrix or an Extent object). • Open Street Map search query. The bounding is automatically generated by querying q from Open Street Map Nominatim. See http://wiki.openstreetmap.org/wiki/Nominatim. If x is not specified, a bounding box can be created from scratch (see details.
ext	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 relative=TRUE. If a negative value is specified, then the shortest side of the bounding box (so width or height) is extended with ext, 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.
cx	center x coordinate
cy	center y coordinate
width	width of the bounding box. These are either absolute or relative (depending on the argument relative).
height	height of the bounding box. These are either absolute or relative (depending on the argument relative).
xlim	limits of the x-axis. These are either absolute or relative (depending on the argument relative).
ylim	limits of the y-axis. See xlim.
relative	boolean that determines whether relative values are used for width, height, xlim and ylim or absolute. If x is unspecified, relative is set to "FALSE".
current.projection	projection string (see set_projection) of the that corresponds to the
projection	projection string (see set_projection) to transform the bounding box to.

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 relavitve determines whether relative or absolute values are used.
- Setting the x and y limits. The argument relavitve 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 coorinates `cx` and `cy`, together with the `width` and `height`.
- Setting the x and y limits `xlim` and `ylim`

Examples

```
## 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)

## convert to longlat
bb(NLD_muni, projection="longlat")

## change existing bounding box
bb(NLD_muni, ext=1.5)
bb(NLD_muni, width=2, relative = TRUE)
bb(NLD_muni, xlim=c(.25, .75), ylim=c(.25, .75), relative = TRUE)
bb("Limburg", projection = get_projection(NLD_muni))

tm_shape(World, bbox=bb("Italy", projection = "eck4")) + tm_polygons()
#identical: tm_shape(World, x="Italy", projection = "eck4") + tm_polygons()

## create new bounding box
tm_shape(NLD_muni, bbox=bb(cx=190000, cy=330000, width=50000, height=50000)) + tm_polygons()
```

---

calc_densities *Calculate densities*

Description

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

Usage

```
calc_densities(shp, var, unit = "km", unit.size = 1000, total.area = NA,
  suffix = "", drop = TRUE)
```

Arguments

shp	a shape object, i.e., a SpatialPolygons(DataFrame)
var	name(s) of a qualtity variable name contained in the shp data
unit	the preferred unit, for instance, "km", "m", or "miles". Density values are calculated in var/unit^2.
unit.size	size of the unit in terms of coordinate units. The coordinate system of many projections is approximately in meters while thematic maps typically range many kilometers, so by default unit="km" and unit.size=1000 (meaning 1 kilometer equals 1000 coordinate units).
total.area	total area size of shp in number of squared units (by default kilometers). Useful if the total area of the shp differs from a reference total area value.
suffix	character that is appended to the variable names. The resulting names are used as column names of the returned data.frame.
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. This can be appended directly to the shape file with append_data with fixed.order=TRUE.

Examples

```
data(NLD_muni)

NLD_muni_pop_per_km2 <- calc_densities(NLD_muni, var = c("pop_men", "pop_women"), suffix = "_km2")
NLD_muni <- append_data(NLD_muni, NLD_muni_pop_per_km2, fixed=TRUE)

qtm(NLD_muni, fill=c("pop_men_km2", "pop_women_km2"), nrow=1, format="NLD")
```

---

get_asp_ratio *Get aspect ratio*

Description

Get the aspect ratio of a shape object, i.e., the map width devided by the map height.

Usage

```
get_asp_ratio(shp)
```

Arguments

shp	shape object, either Spatial or a Raster.

Value

aspect ratio

get_IDs *Get ID's of the shape items*

Description

Get ID's of the shape items. For polygons and lines, the ID attribute is used. For points, the coordinates are used.

Usage

```
get_IDs(shp)
```

Arguments

shp	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame)

Value

vector of ID's

get_proj4 *Get a PROJ.4 character string*

Description

Get full PROJ.4 string from either an existing PROJ.4 string or a shortcut.

Usage

```
get_proj4(x)
```

Arguments

x	a PROJ.4 character string or a shortcut, which is one of: "longlat" Not really a projection, but a plot of the longitude-latitude coordinates (WGS84 datum). "wintri" Winkel Tripel (1921). Popular projection that is useful in world maps. It is the standard of world maps made by the National Geographic Society. Type: compromise "robin" Robinson (1963). Another popular projection for world maps. Type: compromise

"eck4" Eckert IV (1906). Projection useful for world maps. Area sizes are preserved, which makes it particularly useful for truthful choropleths. Type: equal-area

"hd" Hobo-Dyer (2002). Another projection useful for world maps in which area sizes are preserved. Type: equal-area

"gall" Gall (Peters) (1855). Another projection useful for world maps in which area sizes are preserved. Type: equal-area

"merc" Mercator (1569). Projection in which shapes are locally preserved. However, areas close to the poles are inflated. Google Maps uses a close variant of the Mercator. Type: conformal

"utmXX(s)" Universal Transverse Mercator. Set of 60 projections where each projection is a traverse mercator optimized for a 6 degree longitude range. These ranges are called UTM zones. Zone 01 covers -180 to -174 degrees (West) and zone 60 174 to 180 east. Replace XX in the character string with the zone number. For southern hemisphere, add "s". So, for instance, the Netherlands is "utm31" and New Zealand is "utm59s"

"mill" Miller (1942). Projetion based on Mercator, in which poles are displayed. Type: compromise

"eqc0" Equirectangular (120). Projection in which distances along meridians are conserved. The equator is the standard parallel. Also known as Plate Carr\'ee. Type: equidistant

"eqc30" Equirectangular (120). Projection in which distances along meridians are conserved. The latitude of 30 is the standard parallel. Type: equidistant

"eqc45" Equirectangular (120). Projection in which distances along meridians are conserved. The latitude of 45 is the standard parallel. Also known as Gall isographic. Type: equidistant

"rd" Rijksdriehoekstelsel. Triangulation coordinate system used in the Netherlands.

Value

validated PROJ.4 character string

See Also

http://en.wikipedia.org/wiki/List_of_map_projections for a overview of projections. http://trac.osgeo.org/proj/ for the PROJ.4 project home page. An extensive list of PROJ.4 codes can be created with rgdal's make_EPSG.

---

itmap *Interactive thematic map*

Description

Convert the tmap output to an interactive SVG, that can be plot in RStudio. In development.

Usage

```
itmap(tm, file = NULL, width = NULL, height = NULL)
```

Arguments

tm	tmap object. A tmap object is created with qtm or by stacking tmap-elements.
file	file name. If specified, the SVG image is saved to this file.
width	width
height	height

Examples

```
## Not run:
data(World, metro)
metro$growth <- (metro$pop2020 - metro$pop2010) / (metro$pop2010 * 10) * 100

require(dplyr)

(tm_shape(World) +
tm_polygons("income_grp", palette="-Blues", contrast=.7, id="name", title="Income group") +
tm_shape(metro) +
tm_bubbles("pop2010", col = "growth",
   border.col = "black", border.alpha = .5,
   style="fixed", breaks=c(-Inf, seq(0, 6, by=2), Inf),
   palette="-RdYlBu", contrast=1,
   title.size="Metro population",
   title.col="Growth rate (%)", id="name") +
tm_layout(legend.bg.color = "grey90", legend.bg.alpha=.5, legend.frame=TRUE, asp=0)) %>%
itmap()

## End(Not run)
```

---

land *Spatial data of global land cover*

Description

Spatial data of global land cover, of class SpatialGridDataFrame. The data includes a population times series from 1950 to (forecasted) 2030. All metro areas with over 1 million inhabitants in 2010 are included.

Usage

```
data(land)
```

Details

Important: publication of these maps is only allowed when cited to Tateishi et al. (2014), and when "Geospatial Information Authority of Japan, Chiba University and collaborating organizations." is shown. See http://www.iscgm.org/gm/glcnmo.html#use.

Source

http://www.iscgm.org/gm/glcnmo.html

References

Tateishi, R., Thanh Hoan, N., Kobayashi, T., Alsaaideh, B., Tana, G., Xuan Phong, D. (2014), Journal of Geography and Geology, 6 (3).

---

map_coloring *Map coloring*

Description

Color the polygons of a map such that adjacent polygons have different colors

Usage

```
map_coloring(x, algorithm = "greedy", ncols = NA, minimize = FALSE,
  palette = NULL, contrast = 1)
```

Arguments

x	Either a SpatialPolygons(DataFrame) or an adjacency list.
algorithm	currently, only "greedy" is implemented.
ncols	number of colors. By default it is 8 when palette is undefined. Else, it is set to the length of palette
minimize	logical that determines whether algorithm will search for a minimal number of colors. If FALSE, the ncols colors will be picked by a random procedure.
palette	color palette.
contrast	vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when auto.palette.mapping=TRUE). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

Value

If palette is defined, a vector of colors is returned, otherwise a vector of color indices.

Examples

```
## Not run:
qtm(World, fill="MAP_COLORS", fill.palette="Pastel2")
tm_shape(World) +
  tm_polygons("MAP_COLORS", palette="Pastel2")

## End(Not run)
data(World, metro)

World$color <- map_coloring(World, palette="Pastel2")
qtm(World, fill = "color")

# map_coloring used indirectly:
qtm(World, fill = "MAP_COLORS")
```

`metro` *Spatial data of metropolitan areas*

Description

Spatial data of metropolitan areas, of class `SpatialPointsDataFrame`. The data includes a population times series from 1950 to (forecasted) 2030. All metro areas with over 1 million inhabitants in 2010 are included.

Usage

```
data(metro)
```

Source

http://esa.un.org/unpd/wup/CD-ROM/

References

United Nations, Department of Economic and Social Affairs, Population Division (2014). World Urbanization Prospects: The 2014 Revision, CD-ROM Edition.

`points_to_raster` *Bin spatial points to a raster*

Description

Bin spatial points to a raster. For each raster cell, the number of points are counted. Optionally, a factor variable can be specified by which the points are counts are split.

Usage

```
points_to_raster(shp, nrow = NA, ncol = NA, N = 250000, by = NULL,
  to.Raster = FALSE)
```

Arguments

shp	shape object. Either a SpatialPoints(DataFrame) or a SpatialGrid(DataFrame).
nrow	number of raster rows. If NA, it is automatically determined by N and the aspect ratio of shp.
ncol	number of raster columns. If NA, it is automatically determined by N and the aspect ratio of shp.
N	preferred number of raster cells.
by	name of a data variable which should be a factor. The points are split and counted according to the levels of this factor.
to.Raster	logical; should the output be a Raster object (TRUE), or a SpatialGridDataFrame (FALSE). If TRUE, a RasterBrick is returned when by is specified, and a RasterLayer when by is unspecified.

Details

This function is a wrapper around rasterize.

Value

A SpatialGridDataFrame, or a Raster object when (to.Raster=TRUE)

See Also

poly_to_raster

Examples

```
## Not run:
data(NLD_muni, NLD_prov)

# sample points (each point represents 1000 people)
NLD_muni_points <- sample_dots(NLD_muni, vars = "population",
   w=1000, convert2density = TRUE)

# dot map
tm_shape(NLD_muni_points) + tm_dots()

# convert points to raster
NLD_rst <- points_to_raster(NLD_muni_points, N = 1e4)

# plot raster
tm_shape(NLD_rst) +
tm_raster() +
tm_shape(NLD_prov) +
```

```
tm_borders() +
tm_format_NLD() + tm_style_grey()

## End(Not run)
```

---

poly_to_raster *Convert spatial polygons to a raster*

Description

Convert spatial polygons to a raster. For each raster cell, the data of the corresponding polygon is copied.

Usage

```
poly_to_raster(shp, nrow = NA, ncol = NA, N = 250000, use.cover = FALSE,
  to.Raster = FALSE, ...)
```

Arguments

shp	shape object. Either a SpatialPoints(DataFrame) or a SpatialGrid(DataFrame).
nrow	number of raster rows. If NA, it is automatically determined by N and the aspect ratio of shp.
ncol	number of raster columns. If NA, it is automatically determined by N and the aspect ratio of shp.
N	preferred number of raster cells.
use.cover	logical; should the cover method be used? This method determines per raster cell which polygon has the highest cover fraction. This method is better, but very slow, since N times the number of polygons combinations are processed (using the getCover argument of rasterize). By default, when a raster cell is covered by multiple polygons, the last polygon is taken (see fun argment of rasterize))
to.Raster	logical; should the output be a Raster object (TRUE), or a SpatialGridDataFrame (FALSE). If TRUE, a RasterBrick is returned when by is specified, and a RasterLayer when by is unspecified.
...	arguments passed on to rasterize

Value

A SpatialGridDataFrame, or a Raster object when (to.Raster=TRUE)

See Also

points_to_raster

Examples

```
## Not run:
data(NLD_muni)

# choropleth of 65+ population percentages
qtm(NLD_muni, fill="pop_65plus", format="NLD")

# rasterized version
NLD_rst <- poly_to_raster(NLD_muni)
qtm(NLD_rst, raster="pop_65plus", format="NLD")

## End(Not run)
```

`print.tmap` *Draw thematic map*

Description

Draw thematic map on current graphics device

Usage

```
## S3 method for class 'tmap'
print(x, vp = NULL, ...)
```

Arguments

x	tmap object. A tmap object is created with qtm or by stacking tmap-elements.
vp	viewport to draw the plot in. This is particularly useful for insets.
...	not used

Value

A list of data.frames is silently returned, containing all ID and aesthetic variables per layer group.

`qtm` *Quick thematic map plot*

Description

Draw a thematic map quickly.

Usage

```
qtm(shp, fill = NA, bubble.size = NULL, bubble.col = NULL, text = NULL,
  text.size = 1, text.col = NA, line.lwd = NULL, line.col = NULL,
  raster = NA, borders = NA, scale = NA, title = NA, format = NULL,
  style = NULL, ...)
```

Arguments

shp	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame) 4. SpatialGrid(DataFrame) 5. SpatialPixels(DataFrame) 6. RasterLayer, RasterStack, or RasterBrick
fill	either a color to fill the polygons, or name of the data variable in shp to draw a choropleth. Only applicable when shp is type 1 (see above).
bubble.size	name of the data variable in shp for the bubble map that specifies the sizes of the bubbles. If neither bubble.size nor bubble.col is specified, no bubble map is drawn. Only applicable when shp is type 1, 2, or 3 (see above).
bubble.col	name of the data variable in shp for the bubble map that specifies the colors of the bubbles. If neither bubble.size nor bubble.col is specified, no bubble map is drawn. Only applicable when shp is type 1, 2, or 3 (see above).
text	Name of the data variable that contains the text labels. Only applicable when shp is type 1, 2, or 3 (see above).
text.size	Font size of the text labels. Either a constant value, or the name of a numeric data variable. Only applicable when shp is type 1, 2, or 3 (see above).
text.col	name of the data variable in shp for the that specifies the colors of the text labels. Only applicable when shp is type 1, 2, or 3 (see above).
line.lwd	either a line width or a name of the data variable that specifies the line width. Only applicable when shp is type 3 (see above).
line.col	either a line color or a name of the data variable that specifies the line colors. Only applicable when shp is type 3 (see above).
raster	either a color or a name of the data variable that specifices the raster colors. Only applicable when shp is type 4, 5, or 6 (see above).
borders	color of the polygon borders. Use NA to omit the borders.
scale	numeric value that serves as the global scale parameter. All font sizes, bubble sizes, border widths, and line widths are controled by this value. The parameters bubble.size, text.size, and line.lwd can be scaled seperately with respectively bubble.scale, text.scale, and line.scale.
title	main title. For legend titles, use X.style, where X is layer name (see ...).
format	tm_layout wrapper used for format. Currently available in tmap: "World", "Europe", "NLD", "World_wide", "Europe_wide", "NLD_wide". Own wrappers can be used as well (see details).

style	tm_layout wrapper used for style. Available in tmap: "bw", "classic". Own wrappers can be used as well (see details). #Currently available in tmap: "cobalt", "albatross", "beaver".
...	arguments passed on to the tm_* functions. If an argument name is not unique for a particular tm_ function, then it should be prefixed with the function name without "tm_". For instance, style is an argument of tm_fill, tm_bubbles, and tm_lines. Therefore, in order to define the style for a choropleth, its arugment name should be fill.style.

Details

This function is a convenient wrapper of the main plotting method of stacking tmap-elements. The first argument is a shape object (normally specified by tm_shape). The next arguments, from fill to raster, are the aesthetics from the main layers. The remaining arguments are related to the map layout. Any argument from any main layer can be specified (see ...). It is also possible to stack tmap-elements on a qtm plot. See examples.

For format any character value, say "xxx" can be used if the wrapper function "tm_format_xxx" exists. The same applies for the arguments colors, and style.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(Europe, World, metro)

# just the map
qtm(Europe)

# choropleth
qtm(World, fill = "economy", text="iso_a3", text.size = "AREA", format = "World",
fill.title="Economy", style="col_blind")

qtm(Europe, fill="gdp_cap_est", text="iso_a3", text.size="pop_est",
    fill.title="GDP per capita", fill.textNA="Non-European countries",
format="Europe", style="cobalt")

qtm(World, fill="HPI", fill.n=9, fill.palette="div", fill.auto.palette.mapping=FALSE,
fill.title="Happy Planet Index", format="World", style="gray")

# bubble map
qtm(World, borders = NULL) + qtm(metro, bubble.size = "pop2010",
    bubble.title.size="Metropolitan Areas", bubble.scale=.5, format = "World")
```

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", "tracks", "routes", "track_points",
  "route_points"))
```

Arguments

file	a GPX filename (including directory)
layers	vector of GPX layers. Possible options are "waypoints", "tracks", "routes", "track_points", "route_points". By dedault, all those layers are read.

Value

for each defiend layer, a shape is returned (only if the layer has any features). If only one layer is defined, the corresponding shape is returned. If more than one layer is defined, a list of shape objects, one for each layer, is returned.

Examples

```
## Not run:
### Demo to visualise the route of the Amstel Gold Race, a professional cycling race

# download data
tmpdir <- tempdir()
tmpfile <- tempfile()
download.file("http://www.gpstracks.nl/routes-fiets/f-limburg-amstel-gold-race-2014.zip",
  tmpfile, mode="wb")
unzip(tmpfile, exdir=tmpdir)

# read GPX file
AGR <- read_GPX(file.path(tmpdir, "f-limburg-amstel-gold-race-2014.gpx"))

# read OSM of Zuid-Limburg (take bounding box and extend it by 1.05)
ZLim <- read_osm(bb(AGR$tracks, ext=1.05))

# change route part names
levels(AGR$tracks$name) <- paste(c("First", "Second", "Third", "Final"), "loop")

# plot it
tm_shape(ZLim) +
  tm_raster(saturation=.25) +
```

```
tm_shape(AGR$tracks) +
tm_lines(col = "name", lwd = 4, title.col="Amstel Gold Race", palette="Dark2") +
tm_shape(AGR$waypoints) +
tm_bubbles(size=.1, col="gold", border.col = "black") +
tm_text("name", size = .75, bg.color="white", bg.alpha=.25, auto.placement = .25) +
tm_legend(position=c("right", "top"), frame=TRUE, bg.color = "gold")

## End(Not run)
```

---

read_osm *Read Open Street Map data*

Description

Read Open Street Map data. Either OSM tiles are read and returned as a spatial raster, or vectorized OSM data are queried and returned as spatial polygons, lines, and/or points.

Usage

```
read_osm(x, raster = NA, zoom = NULL, type = NULL, minNumTiles = NULL,
  mergeTiles = NULL, ...)

osm_poly(query)

osm_line(query)

osm_point(query)
```

Arguments

x	bounding box or osmar object
raster	logical that determines whether a raster or vector shapes are returned. In the latter case, specify the vector selections (see argument . . .). By default, raster=TRUE if no vector selections are made, and raster=FALSE otherwise.
zoom	passed on to openmap. Only applicable when raster=TRUE.
type	passed on to openmap Only applicable when raster=TRUE.
minNumTiles	passed on to openmap Only applicable when raster=TRUE.
mergeTiles	passed on to openmap Only applicable when raster=TRUE.
...	arguments that specify polygons, lines, and/or points queries, created with osm_poly, osm_line, and osm_point respectively.
query	query to select polygons, lines, or points. Currently, two formats are supported: 1) key, 2) key=value. See http://wiki.openstreetmap.org/wiki/Map_Features for Open Street Map keys and values.

Value

The output of read_osm is a SpatialGridDataFrame if raster=TRUE, and otherwise a named list of SpatialPolygonsDataFrame, SpatialLinesDataFrame, and/or SpatialPointsDataFrame objects. The names of this list are the names of arguments defined at ....

Examples

```
## Not run:
#### Choropleth with OSM background

# load Netherlands shape
data(NLD_muni)

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


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

#### A close look at Aalborg Congress Centre (host for the useR2014)

# define bounding box of Aalborg Congress Centre
bb_Aal <- bb(xlim = c(9.9075, 9.9175), ylim=c(57.043, 57.046))

# read OSM raster data
rast_Aal <- read_osm(bb_Aal, type="mapquest")

# raster OSM of Aalburg
qtm(rast_Aal)

# read OSM vector data
vec_Aal <- read_osm(bb_Aal,
buildings=osm_poly("building"),
roads=osm_line("highway"),
trees=osm_point("natural=tree"),
park=osm_poly("leisure=park"),
cemetery=osm_poly("landuse=cemetery"),
railway=osm_line("railway"),
parking=osm_poly("amenity=parking"))

# vector OSM of Aalburg
tm_shape(vec_Aal$park, bbox=bb_Aal) +
tm_polygons(col = "darkolivegreen3") +
tm_shape(vec_Aal$cemetery) +
tm_polygons(col="darkolivegreen3") +
tm_shape(vec_Aal$parking) +
tm_polygons(col="grey85") +
tm_shape(vec_Aal$building) +
```

```
tm_polygons(col = "gold") +
tm_shape(vec_Aal$roads) +
tm_lines("grey40", lwd = 3) +
tm_shape(vec_Aal$trees) +
tm_bubbles(size=.25, col="forestgreen") +
tm_shape(vec_Aal$railway) +
tm_lines(col = "grey40", lwd = 3, lty = "longdash") +
tm_layout(inner.margins=0, bg.color="grey95")

## End(Not run)
```

`read_shape` *Read shape file*

Description

Read an ESRI shape file. Optionally, set the current projection if it is missing.

Usage

```
read_shape(file, current.projection = NULL, ...)
```

Arguments

`file` a shape file name (including directory)

`current.projection`
the current projection of the shape object, if it is missing in the shape file. It should be either a `PROJ.4` character string (see http://trac.osgeo.org/proj/), of one of the following shortcuts:

`"longlat"` Not really a projection, but a plot of the longitude-latitude coordinates (WGS84 datum).

`"wintri"` Winkel Tripel (1921). Popular projection that is useful in world maps. It is the standard of world maps made by the National Geographic Society. Type: compromise

`"robin"` Robinson (1963). Another popular projection for world maps. Type: compromise

`"eck4"` Eckert IV (1906). Projection useful for world maps. Area sizes are preserved, which makes it particularly useful for truthful choropleths. Type: equal-area

`"hd"` Hobo-Dyer (2002). Another projection useful for world maps in which area sizes are preserved. Type: equal-area

`"gall"` Gall (Peters) (1855). Another projection useful for world maps in which area sizes are preserved. Type: equal-area

`"merc"` Mercator (1569). Projection in which shapes are locally preserved. However, areas close to the poles are inflated. Google Maps uses a close variant of the Mercator. Type: conformal

"utmXX(s)" Universal Transverse Mercator. Set of 60 projections where each projection is a traverse mercator optimized for a 6 degree longitude range. These ranges are called UTM zones. Zone 01 covers -180 to -174 degrees (West) and zone 60 174 to 180 east. Replace XX in the character string with the zone number. For southern hemisphere, add "s". So, for instance, the Netherlands is "utm31" and New Zealand is "utm59s"

"mill" Miller (1942). Projetion based on Mercator, in which poles are displayed. Type: compromise

"eqc0" Equirectangular (120). Projection in which distances along meridians are conserved. The equator is the standard parallel. Also known as Plate Carr\'ee. Type: equidistant

"eqc30" Equirectangular (120). Projection in which distances along meridians are conserved. The latitude of 30 is the standard parallel. Type: equidistant

"eqc45" Equirectangular (120). Projection in which distances along meridians are conserved. The latitude of 45 is the standard parallel. Also known as Gall isographic. Type: equidistant

"rd" Rijksdriehoekstelsel. Triangulation coordinate system used in the Netherlands.

See http://en.wikipedia.org/wiki/List_of_map_projections for a overview of projections. Use set_projection to reproject the shape object.

... other parameters, such as stringsAsFactors, are passed on to readOGR

Details

This function is a convenient wrapper of rgdal's readOGR. It is possible to set the current projection, if it is undefined in the shape file. If a reprojection is required, use set_projection.

For the Netherlands: often, the Dutch Rijksdriehoekstelsel (Dutch National Grid) projection is provided in the shape file without proper datum shift parameters to wgs84. This functions automatically adds these parameters. See http://www.qgis.nl/2011/12/05/epsg28992-of-rijksdriehoekstelsel-verschuiving/ (in Dutch) for details.

Value

shape object

---

rivers *Spatial data of rivers*

Description

Spatial data of rivers, of class SpatialLinesDataFrame

Usage

```
data(rivers)
```

Source

http://www.naturalearthdata.com

`sample_dots` *Sample dots from spatial polygons*

Description

Sample dots from spatial polygons according to a spatial distribution of a population. The population may consist of classes. The output, a SpatialPointsDataFrame, can be used to create a dot map (see `tm_dots`), where the dots are colored according to the classes.

Usage

```
sample_dots(shp, vars = NULL, convert2density = FALSE, nrow = NA,
  ncol = NA, N = 250000, npop = NA, n = 10000, w = NA,
  shp.id = NULL, var.name = "class", var.labels = vars, unit = "km",
  unit.size = 1000, randomize = TRUE, ...)
```

Arguments

`shp`	A shape object, more specifically, a `SpatialPolygonsDataFrame`.
`vars`	Names of one or more variables that are contained in `shp`. If `vars` is not provided, the dots are sampled uniformly. If `vars` consists of one variable name, the dots are sampled according to the distribution of the corresponding variable. If `vars` consist of more than one variable names, then the dots are sampled according to the distributions of those variables. A categorical variable is added that contains the distrubtion classes (see `var.name`).
`convert2density`	Should the variables be converted to density values? Density values are used for the sampling algorithm, so use `TRUE` when the values are absolute counts.
`nrow`	Number of grid rows
`ncol`	Number of grid colums
`N`	Number of grid points
`npop`	Population total. If `NA`, it is recontructed from the data. If density values are specified, the population total is approximated using the polygon areas (see also `unit` and `unit.size`).
`n`	Number of sampled dots
`w`	Number of population units per dot. It is the population total divided by `n`. If specified, `n` is calculated accordingly.
`shp.id`	Name of the variable of `shp` that contains the polygon identifying numbers or names.
`var.name`	Name of the variable that will be created to store the classes. The classes are defined by `vars`, and the labels can be configured with `var.labels`.

var.labels	Labels of the classes (see var.name).
unit	Unit, see calc_densities. Needed to relate npop to w, if they are not both specified.
unit.size	Unit size, see calc_densities. Needed to relate npop to w, if they are not both specified.
randomize	should the order of sampled dots be randomized? The dots are sampled class-wise (specified by vars). If this order is not randomized (so if randomize=FALSE), then the dots from the last class will be drawn on top, which may introduce a perception bias. By default randomize=TRUE, so the sampled dots are randomized to prevent this bias.
...	other arguments passed on to calc_densities and approx_areas

Details

The sampling algoritm is the following: TO DO

Examples

```
data(World)
World_dots <- sample_dots(World, vars="pop_est_dens", nrow=200, ncol=400, w=1e6)

tm_shape(World_dots) + tm_dots(size = .02, jitter=.1) +
   tm_layout("One dot represents one million people", title.position = c("right", "bottom"))

## Not run:
# download Dutch neighborhood shape file
dir <- tempdir()
temp <- tempfile()

download.file("http://www.cbs.nl/nl-NL/menu/themas/dossiers/nederland-regionaal/links/
  2014-buurtkaart-shape-versie-1-el.htm", temp, mode="wb")
unzip(temp, exdir = dir)

NLD_nbhd <- read_shape(file.path(dir, "buurt_2014.shp"))

# fix self-intersection but in shape object
NLD_nbhd <- rgeos::gBuffer(NLD_nbhd, byid=TRUE, width=0)

# remove all water neighborhoods
NLD_nbhd <- NLD_nbhd[NLD_nbhd$OPP_LAND>0, ]

# process data
NLD_nbhd@data <- within(NLD_nbhd@data, {
# convert land area ("OPP_LAND") from hectare (ha) to km2
OPP_LAND <- OPP_LAND / 100

# set negative percentages of western and non-western immigrants to 0
P_WEST_AL[P_WEST_AL<0] <- 0
P_N_W_AL[P_N_W_AL<0] <- 0
```

```
# calculate population density values
dens <- (AANT_INW / OPP_LAND)
dens[is.nan(dens)] <- 0

# divide population in three groups:
# western and non-western immigrants, and native Dutch
west <- dens * P_WEST_AL / 100
non_west <- dens * P_N_W_AL / 100
dutch <- dens - non_west - west
})

# Select The Hague (Den Haag) area
DH_bbox <- bb(NLD_nbhd[which(NLD_nbhd$GM_NAAM=="'s-Gravenhage"), ])

# Crop shape to The Hague area (use 1.05 bounding box extension to make sure the
# whole region is covered after reprojection to the Mercator porjection)
DH_nbhd <- raster::crop(NLD_nbhd, bb(DH_bbox, 1.05))

# Read OSM layer
DH_nbhd_osm <- read_osm(bb(DH_bbox, current.projection="rd", projection="longlat"),
type = "mapquest")

# Sample dots (each dot represents 100 persons)
DH_nbhd_dots <- sample_dots(DH_nbhd, c("dutch", "west", "non_west"),
convert2density = FALSE,
N=250000, w=100,
var.labels = c("Dutch (native)", "Western immigrants", "Non-western immigrants"),
shp.id = "ID")

# Show map
tm_shape(DH_nbhd_osm) + tm_raster(saturation=.2) +
tm_shape(DH_nbhd_dots) +
    tm_dots("class", size=.04, alpha=.75, palette="Dark2",
     title = "The Hague population") +
tm_layout(inner.margins=0, legend.frame=TRUE, legend.bg.color="grey90")

## End(Not run)
```

---

save_tmap *Save tmap*

Description

Save tmap to a file, such as png, jpg, or pdf.

Usage

```
save_tmap(tm, filename = shp_name(tm), width = par("din")[1],
  height = par("din")[2], units = c("in", "cm", "mm"), dpi = 300,
  outer.margins = 0, asp = 0, scale = NA, ...)
```

Arguments

tm	tmap object
filename	filename including extension, and optionally the path. The extensions pdf, eps, svg, wmf (Windows only), png, jpg, bmp, or tiff are supported. Use itmap to create an interactive svg.
width	width. Defaults to the width of current plotting window. Units are set with the argument units.
height	height. Defaults to the height of current plotting window. Units are set with the argument units.
units	units for width and height when either one is explicitly specified (in, cm, or mm)
dpi	dots per inch. Only applicable for raster graphics.
outer.margins	overrides the outer.margins argument of tm_layout (unless set to NA)
asp	overrides the asp argument of tm_layout (unless set to NA)
scale	overrides the scale argument of tm_layout (unless set to NA)
...	arguments passed on to device functions

Examples

```
## Not run:
data(NLD_muni, NLD_prov)
(tm_shape(NLD_muni) +
 tm_fill(col="population", convert2density=TRUE,
   style="kmeans", title="Population (per km2)", legend.hist=FALSE) +
 tm_borders("black", alpha=.5) +
tm_shape(NLD_prov) +
 tm_borders("grey25", lwd=2) +
tm_format_NLD(inner.margins = c(.02, .15, .06, .15)) +
tm_scale_bar(position = c("left", "bottom")) +
tm_compass(position=c("right", "bottom")) +
tm_style_classic()) %>% save_tmap()

## End(Not run)
```

---

sbind *Combine shape objects*

Description

Combine shape objects into one shape object. It works analogous to rbind.

Usage

```
sbind(...)
```

Arguments

...	shape objects. Each shape object is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame)

Value

shape object

---

set_projection *Set and get the map projection*

Description

The function set_projection sets the projection of a shape file. It is a convenient wrapper of spTransform and projectRaster with shortcuts for commonly used projections. The projection can also be set directly in the plot call with tm_shape. This function is also used to set the current projection information if this is missing. The function get_projection is used to get the projection information.

Usage

```
set_projection(shp, projection = NULL, current.projection = NULL,
  overwrite.current.projection = FALSE)

get_projection(shp)
```

Arguments

shp	shape object of class Spatial or Raster
projection	character that determines the new projection. Either a PROJ.4 character string or a shortcut. See get_proj4 for a list of shortcut values. This argument is only used to transform the shp. Use current.projection to specify the current projection of shp.
current.projection	the current projection of shp. Only use this if the current projection is missing or wrong.
overwrite.current.projection	logical that determines whether the current projection is overwritten if it already has a projection that is different.

Value

set_projection returns a (transformed) shape object with updated projection information. get_projection returns the PROJ.4 character string of shp.

smooth_map *Create a smooth map in various formats: smooth raster, contour lines, and dasymetric polygons.*

Description

Create contour lines (isolines) from a shape object. To make the iso lines smooth, a 2D kernal density estimator is applied on the shape object. These lines are used to draw an isopleth. Also, the polygons between the countour lines are returned. They can be used to create a dasymetric map.

Usage

```
smooth_map(shp, var = NULL, nrow = NA, ncol = NA, N = 250000,
  unit = "km", unit.size = 1000, smooth.raster = TRUE, nlevels = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  bandwidth = NA, cover.type = NA, cover = NULL, cover.threshold = 0.6,
  weight = 1, extracting.method = "full", buffer.width = NA,
  to.Raster = FALSE)
```

Arguments

shp	shape object of class Spatial or Raster. Spatial points, polygons, and grids are supported. Spatial lines are not.
var	variable name. Not needed for SpatialPoints. If missing, the first variable name is taken.
nrow	number of rows in the raster that is used to smooth the shape object. Only applicable if shp is not a SpatialGrid(DataFrame) or Raster
ncol	number of rows in the raster that is used to smooth the shape object. Only applicable if shp is not a SpatialGrid(DataFrame) or Raster
N	preferred number of points in the raster that is used to smooth the shape object. Only applicable if shp is not a SpatialGrid(DataFrame) or Raster
unit	unit specification. Needed when calculating density values. When set to NA, the densities values are based on the dimensions of the raster (defined by nrow and ncol). See also unit.size.
unit.size	size of the unit in terms of coordinate units. The coordinate system of many projections is approximately in meters while thematic maps typically range many kilometers, so by default unit="km" and unit.size=1000 (meaning 1 kilometer equals 1000 coordinate units).
smooth.raster	logical that determines whether 2D kernel density smoothing is applied to the raster shape object. Not applicable when shp is a SpatialPoints object.
nlevels	preferred number of levels
style	method to cut the color scale: e.g. "fixed", "equal", "pretty", "quantile", or "kmeans". See the details in classIntervals.
breaks	in case style=="fixed", breaks should be specified

bandwidth — single numeric value or vector of two numeric values that specifiy the bandwidth of the kernal density estimator. By default, it is 1/50th of the shortest side in units (specified with unit.size).

cover.type — character value that specifies the type of raster cover, in other words, how the boundaries are specified. Options: "original" uses the same boundaries as shp (default for polygons), "smooth" calculates a smooth boundary based on the 2D kernal density (determined by smooth_raster_cover), "rect" uses the bounding box of shp as boundaries (default for spatial points and grids).

cover — SpatialPolygons shape that determines the covered area in which the contour lines are placed. If specified, cover.type is ignored.

cover.threshold — numeric value between 0 and 1 that determines which part of the estimated 2D kernal density is returned as cover. Only applicable when cover.type="smooth".

weight — single number that specifies the weight of a single point. Only applicable if shp is a SpatialPoints object.

extracting.method — Method of how coordinates are extracted from the dasymetric polygons. Options are: "full" (default), "grid", and "single". See details. For the slowest method "full", extract is used. For "grid", points on a grid layout are selected that intersect with the polygon. For "simple", a simple point is generated with gPointOnSurface.

buffer.width — Buffer width of the iso lines to cut dasymetric polygons. Should be small enough to let the polygons touch each other without space in between. However, too low values may cause geometric errors.

to.Raster — should the "raster" output (see output) be a RasterLayer? By default, it is returned as a SpatialGridDataFrame

Details

For the estimation of the 2D kernal density, code is borrowed from bkde2D. This implemention is slightly different: bkde2D takes point coordinates and applies linear binning, whereas in this function, the data is already binned, with values 1 if the values of var are not missing and 0 if values of var are missing.

Value

List with the following items:

"raster" A smooth raster, which is either a SpatialGridDataFrame or a RasterLayer (see to.Raster)

"iso" Contour lines, which is a SpatialLinesDataFrame

"dasy" Dasymetric polygons, which is a SpatialPolygonsDataFrame

"bbox" Bounding box of the used raster

"ncol" Number of rows in the raster

"nrow" Number of columns in the raster

smooth_raster_cover *Get a smoothed cover of a raster object*

Description

Get a smoothed cover of a raster object. From all non-missing values of a raster object, a 2D kernal density is applied. The output is a SpatialPolygons object. Used by smooth_map.

Usage

```
smooth_raster_cover(shp, var = NA, bandwidth = NA, threshold = 0.6,
  output = "SpatialPolygons")
```

Arguments

shp	raster object, from either SpatialGrid(DataFrame) or Raster class.
var	name of the variable from which missing values are flagged. If unspecified, the first variable will be taken.
bandwidth	single numeric value or vector of two numeric values that specifiy the bandwidth of the kernal density estimator. See details.
threshold	numeric value between 0 and 1 that determines which part of the estimated 2D kernal density is returned as cover.
output	class of the returned object. One of: SpatialPolygons, SpatialLines, SpatialGridDataFrame, or RasterLayer. A vector of class names results in a list of output objects.

Details

For the estimation of the 2D kernal density, code is borrowed from bkde2D. This implemention is slightly different: bkde2D takes point coordinates and applies linear binning, whereas in this function, the data is already binned, with values 1 if the values of var are not missing and 0 if values of var are missing.

split.SpatialPolygons *Divide into multiple shape objects*

Description

Divide a shape object into multiple objects.

Usage

```
## S3 method for class 'SpatialPolygons'
split(x, f, drop = FALSE, ...)

## S3 method for class 'SpatialPoints'
split(x, f, drop = FALSE, ...)

## S3 method for class 'SpatialLines'
split(x, f, drop = FALSE, ...)
```

Arguments

x	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame)
f	factor to split x
drop	unused factor levels are dropped
...	other arguments (not used)

Value

List of shape objects.

---

style_catalogue *Create a style catalogue*

Description

Create a style catalogue for each predefined tmap style. The result is a set of png images, one for each style.

Usage

```
style_catalogue(path = "./tmap_style_previews", styles = NA,
  include.global.styles = TRUE)

style_catalog(path = "./tmap_style_previews", styles = NA,
  include.global.styles = TRUE)
```

Arguments

path	path where the png images are stored
styles	vector of tm_style_XXX function names. By default, it contains all styles that are included in tmap, and, if include.global.styles=TRUE, also the user defined tm_style_XXX function names.
include.global.styles	See above

---

tmap-element *tmap element*

Description

Building block for drawing thematic maps. All element functions have the prefix tm_.

Details

The fundamental, and hence required element is

- tm_shape that specifies the shape object, and also specifies the projection and bounding box

The elements that serve as aesthetics layers are

- tm_fill to fill the polygons
- tm_borders to draw polygon borders
- tm_polygons to draw polygons (it is a combination of tm_fill and tm_borders)
- tm_bubbles to draw bubbles
- tm_lines to draw lines
- tm_text to add text annotations
- tm_raster to draw a raster

The layers can be stacked by simply adding them with the + symbol. The combination of the elements described above form one group. Multiple groups can be stacked. Each group should start with tm_shape.

The attribute elements are

- tm_grid to specify coordinate grid lines
- tm_credits to add a credits/acknowledgements text label
- tm_scale_bar to add a measurement scale bar
- tm_compass to add a map compass

The element tm_facets specifies facets (small multiples). The element tm_layout is used to change the layout of the map.

See Also

vignette("tmap-nutshell")

The examples in each of the element functions

tm_bubbles *Draw bubbles or dots*

Description

Creates a tmap-element that draws bubbles or small dots. Both colors and sizes of the bubbles can be mapped to data variables.

Usage

```
tm_bubbles(size = 0.2, col = NA, alpha = NA, border.col = NA,
  border.lwd = 1, border.alpha = NA, scale = 1, perceptual = FALSE,
  size.lim = NA, sizes.legend = NULL, sizes.legend.labels = NULL, n = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  palette = NULL, labels = NULL, auto.palette.mapping = TRUE,
  contrast = 1, max.categories = 12, colorNA = NA, textNA = "Missing",
  jitter = 0, xmod = 0, ymod = 0, title.size = NA, title.col = NA,
  legend.size.show = TRUE, legend.col.show = TRUE, legend.format = list(),
  legend.size.is.portrait = FALSE, legend.col.is.portrait = TRUE,
  legend.hist = FALSE, legend.hist.title = NA, legend.size.z = NA,
  legend.col.z = NA, legend.hist.z = NA, id = NA)

tm_dots(col = NA, size = 0.02, title = NA, legend.is.portrait = TRUE,
  legend.z = NA, ...)
```

Arguments

size	a single value or a shp data variable that determines the bubble sizes. The reference value size=1 corresponds to the area of bubbles that have the same height as one line of text. If a data variable is provided, the bubble sizes are scaled proportionally (or perceptually, see perceptual) where the largest bubble will get size=1. If multiple values are specified, small multiples are drawn (see details).
col	color(s) of the bubble. Either a color (vector), or categorical variable name(s). If multiple values are specified, small multiples are drawn (see details).
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
border.col	color of the bubble borders.
border.lwd	line width of the bubble borders. If NA (default), no bubble borders are drawn.
border.alpha	transparency number, regarding the bubble borders, between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
scale	bubble size multiplier number.
perceptual	logical that determines whether bubbles are scales with a perceptually (TRUE) or mathematically (FALSE, default value). The perceived area of larger bubbles is often underestimated. Flannery (1971) experimentally derived a method to compensate this, which is enabled by this argument.

`size.lim` vector of two limit values of the `size` variable. Only bubbles are drawn whose value is greater than or equal to the first value. Bubbles whose values exceed the second value are drawn at the size of the second value. Only applicable when `size` is the name of a numeric variable of `shp`

`sizes.legend` vector of bubble sizes that are shown in the legend. By default, this is determined automatically.

`sizes.legend.labels` vector of labels for that correspond to `sizes.legend`.

`n` preferred number of color scale classes. Only applicable when `col` is a numeric variable name.

`style` method to process the color scale when `col` is a numeric variable. Discrete options are `"cat"`, `"fixed"`, `"sd"`, `"equal"`, `"pretty"`, `"quantile"`, `"kmeans"`, `"hclust"`, `"bclust"`, `"fisher"`, and `"jenks"`. A numeric variable is processed as a categorial variable when using `"cat"`, i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in `classIntervals`. Continuous options are "cont" and "order". The former maps the values of `col` to a smooth gradient, whereas the latter maps the order of values of `col` to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".

`breaks` in case `style=="fixed"`, breaks should be specified

`palette` color palette (see `RColorBrewer::display.brewer.all`) for the bubbles. Only when `col` is set to a variable. The default palette is taken from `tm_layout`'s argument `aes.palette`.

`labels` labels of the classes

`auto.palette.mapping` When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values.

`contrast` vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when `auto.palette.mapping=TRUE`). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

`max.categories` in case `col` is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than `max.categories`, then levels are combined.

`colorNA` colour for missing values

`textNA` text used for missing values. Use `NA` to omit text for missing values in the legend

`jitter` number that determines the amount of jittering, i.e. the random noise added to the position of the bubbles. 0 means no jittering is applied, any positive number means that the random noise has a standard deviation of `jitter` times the height of one line of text line.

`xmod` horizontal position modification of the bubbles, in terms of the height of one line of text. Either a single number for all polygons, or a numeric variable in the shape data specifying a number for each polygon. Together with `ymod`, it determines position modification of the bubbles. See also `jitter` for random position modifications. In most coordinate systems (projections), the origin is located at the bottom left, so negative `xmod` move the bubbles to the left, and negative `ymod` values to the bottom.

`ymod` vertical position modification. See xmod.

`title.size` title of the legend element regarding the bubble sizes

`title.col` title of the legend element regarding the bubble colors

`legend.size.show`
logical that determines whether the legend for the bubble sizes is shown

`legend.col.show`
logical that determines whether the legend for the bubble colors is shown

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. Parameters are:

scientific Should the labels be formatted scientically? If so, square brackets are used, and the `format` of the numbers is `"g"`. Otherwise, `format="f"`, and `text.separator`, `text.less.than`, and `text.or.more` are used. Also, the numbers are automatically rounded to millions or billions if applicable.

format By default, `"f"`, i.e. the standard notation `xxx.xxx`, is used. If `scientific=TRUE` then `"g"`, which means that numbers are formatted scientically, i.e. `n.dddE+nn` if needed to save space.

digits Number of digits after the decimal point if `format="f"`, and the number of significant digits otherwise.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character string to use to translate "Less than" (which is the default).

text.or.more Character string to use to translate "or more" (which is the default).

... Other arguments passed on to `formatC`

`legend.size.is.portrait`
logical that determines whether the legend element regarding the bubble sizes is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.col.is.portrait`
logical that determines whether the legend element regarding the bubble colors is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.hist` logical that determines whether a histogram is shown regarding the bubble colors

`legend.hist.title`
title for the histogram. By default, one title is used for both the histogram and the normal legend for bubble colors.

legend.size.z	index value that determines the position of the legend element regarding the bubble sizes with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.
legend.col.z	index value that determines the position of the legend element regarding the bubble colors. (See legend.size.z)
legend.hist.z	index value that determines the position of the histogram legend element. (See legend.size.z)
id	name of the data variable that specifies the indices of the bubbles. Only used for SVG output (see itmap).
title	shortcut for title.col for tm_dots
legend.is.portrait	shortcut for legend.col.is.portrait for tm_dots
legend.z	shortcut for legend.col.z shortcut for tm_dots
...	arguments passed on to tm_bubbles

Details

Small multiples can be drawn in two ways: either by specifying the by argument in tm_facets, or by defining multiple variables in the aesthetic arguments. The aesthetic arguments of tm_bubbles are size and col. In the latter case, the arguments, except for the ones starting with legend., can be specified for small multiples as follows. If the argument normally only takes a single value, such as n, then a vector of those values can be specified, one for each small multiple. If the argument normally can take a vector, such as palette, then a list of those vectors (or values) can be specified, one for each small multiple.

Value

tmap-element

References

Flannery J (1971). The Relative Effectiveness of Some Common Graduated Point Symbols in the Presentation of Quantitative Data. Canadian Cartographer, 8 (2), 96-109.

See Also

vignette("tmap-nutshell")

Examples

```
data(World, metro)
metro$growth <- (metro$pop2020 - metro$pop2010) / (metro$pop2010 * 10) * 100

tm_shape(World) +
    tm_fill("grey70") +
tm_shape(metro) +
    tm_bubbles("pop2010", col = "growth",
```

```
        border.col = "black", border.alpha = .5,
        style="fixed", breaks=c(-Inf, seq(0, 6, by=2), Inf),
        palette="-RdYlBu", contrast=1,
        title.size="Metro population",
        title.col="Growth rate (%)") +
tm_format_World()

## Not run:
x <- sample_dots(World, vars="gdp_md_est", convert2density = TRUE, w = 100000)
tm_shape(x) +
tm_dots() +
tm_layout("World GDP (one dot is 100 billon dollars)", title.position = c("right", "bottom"))

## End(Not run)
```

`tm_compass` *Map compass*

Description

Creates a map compass.

Usage

```
tm_compass(north = 0, type = NA, fontsize = 0.8, size = NA,
  show.labels = 1, cardinal.directions = c("N", "E", "S", "W"),
  text.color = NA, color.dark = NA, color.light = NA, position = NA)
```

Arguments

`north`	north direction in degrees: 0 means up, 90 right, etc.
`type`	compass type, one of: `"arrow"`, `"4star"`, `"8star"`, `"radar"`, `"rose"`. The default is controlled by `tm_layout` (which uses `"arrow"` for the default style)
`fontsize`	relative font size
`size`	size of the compass in number of text lines. The default values depend on the `type`: for `"arrow"` it is 2, for `"4star"` and `"8star"` it is 4, and for `"radar"` and `"rose"` it is 6.
`show.labels`	number that specifies which labels are shown: 0 means no labels, 1 (default) means only north, 2 means all four cardinal directions, and 3 means the four cardinal directions and the four intercardinal directions (e.g. north-east).
`cardinal.directions`	labels that are used for the cardinal directions north, east, south, and west.
`text.color`	color of the text. By default equal to the argument `attr.color` of `tm_layout`.
`color.dark`	color of the dark parts of the compass, typically (and by default) black.
`color.light`	color of the light parts of the compass, typically (and by default) white.

position	position of the text. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y value of the left bottom corner of the compass. The uppercase values correspond to the position without margins (so tighter to the frame). The default value is controlled by the argument "attr.position" of tm_layout.

Examples

```
data(NLD_muni)
qtm(NLD_muni, theme = "NLD") + tm_compass()
qtm(NLD_muni, theme = "NLD") + tm_compass(type="4star")
qtm(NLD_muni, theme = "NLD") + tm_compass(type="8star")
qtm(NLD_muni, theme = "NLD") + tm_compass(type="rose", position=c("left", "top"))
qtm(NLD_muni, theme = "NLD") + tm_compass(type="radar", position=c("left", "top"), show.labels = 3)
```

---

tm_credits *Credits text*

Description

Creates a text annotation that could be used for credits or acknowledgements.

Usage

```
tm_credits(text, size = 0.7, col = NA, alpha = NA, align = "left",
  bg.color = NA, bg.alpha = NA, fontface = NA, fontfamily = NA,
  position = NA)
```

Arguments

text	text. Multiple lines can be created with the line break symbol "\n".
size	relative text size
col	color of the text. By default equal to the argument attr.color of tm_layout.
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of col is used (normally 1).
align	horizontal alignment: "left" (default), "center", or "right". Only applicable if text contains multiple lines
bg.color	background color for the text
bg.alpha	Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the bg.color is used (normally 1).
fontface	font face of the text. By default, determined by the fontface argument of tm_layout.
fontfamily	font family of the text. By default, determined by the fontfamily argument of tm_layout.

position — position of the text. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y value of the center of the text. The uppercase values correspond to the position without margins (so tighter to the frame). The default value is controlled by the argument "attr.position" of tm_layout.

Examples

```
data(NLD_muni, NLD_prov)

tm_shape(NLD_muni) +
    tm_fill(col="population", convert2density=TRUE,
        style="kmeans", title="Population (per km2)") +
    tm_borders("grey25", alpha=.5) +
    tm_shape(NLD_prov) +
    tm_borders("grey40", lwd=2) +
tm_format_NLD(bg.color="white", frame = TRUE) +
tm_credits("(c) Statistics Netherlands (CBS) and\nKadaster Nederland", position=c("left", "bottom"))
```

---

tm_facets *Small multiples*

Description

Creates a tmap-element that specifies how small multiples are placed in a facet grid. Either the argument by should be specified, i.e. the name of a variable by which the data is grouped, or multiple variable names sould be provided with tm_fill, tm_lines, or tm_bubbles. In this function, the number of rows and columns can be specified, as well as whether the scales are free (i.e. independent of each other).

Usage

```
tm_facets(by = NULL, ncol = NULL, nrow = NULL, free.coords = FALSE,
  drop.shapes = free.coords, free.scales = is.null(by),
  free.scales.fill = free.scales, free.scales.bubble.size = free.scales,
  free.scales.bubble.col = free.scales, free.scales.text.size = free.scales,
  free.scales.text.col = free.scales, free.scales.line.col = free.scales,
  free.scales.line.lwd = free.scales, free.scales.raster = free.scales,
  inside.original.bbox = FALSE, scale.factor = 2)
```

Arguments

by — data variable name by which the data is split

ncol — number of columns of the small multiples grid

nrow — number of rows of the small multiples grid

`free.coords` logical. If the by argument is specified, should each map has its own coordinate ranges?

`drop.shapes` logical. If the by argument is specified, should all non-selected shapes be dropped?

`free.scales` logical. Should all scales of the plotted data variables be free, i.e. independent of each other? Possible data variables are color from `tm_fill`, color and size from `tm_bubbles` and line color from `tm_lines`.

`free.scales.fill` logical. Should the color scale for the choropleth be free?

`free.scales.bubble.size` logical. Should the bubble size scale for the bubble map be free?

`free.scales.bubble.col` logical. Should the color scale for the bubble map be free?

`free.scales.text.size` logical. Should the text size scale be free?

`free.scales.text.col` logical. Should the text color scale be free?

`free.scales.line.col` Should the line color scale be free?

`free.scales.line.lwd` Should the line width scale be free?

`free.scales.raster` Should the color scale for raster layers be free?

`inside.original.bbox` If `free.coords`, should the bounding box of each small multiple be inside the original bounding box?

`scale.factor` Number that determines how the elements (e.g. font sizes, bubble sizes, line widths) of the small multiples are scaled in relation to the scaling factor of the shapes. The elements are scaled to the `scale.factor`th root of the scaling factor of the shapes. So, for `scale.factor=1`, they are scaled proportional to the scaling of the shapes. Since elements, especially text, are often too small to read, a higher value is recommended. By default, `scale.factor=2`.

Value

`tmap-element`

See Also

`vignette("tmap-nutshell")`

Examples

```
data(World, Europe, NLD_muni, NLD_prov)

# Facets defined by constant values
tm_shape(World) +
    tm_fill(c("forestgreen", "goldenrod")) +
```

```
tm_format_World(title=c("A green world", "A dry world"), bg.color="lightskyblue2",
    title.position=c("left", "bottom"))

# Facets defined by multiple variables
tm_shape(Europe) +
    tm_borders() +
    tm_fill(c("gdp_cap_est", "pop_est_dens"), style="kmeans",
        title=c("GDP per capita", "Population density")) +
tm_format_Europe()

tm_shape(NLD_muni) +
    tm_fill(c("pop_0_14", "pop_15_24", "pop_25_44", "pop_45_64", "pop_65plus"),
        style="kmeans",
        palette=list("Oranges", "Greens", "Blues", "Purples", "Greys"),
        title=c("Population 0 to 14", "Population 15 to 24", "Population 25 to 44",
            "Population 45 to 64", "Population 65 and older")) +
tm_shape(NLD_prov) +
    tm_borders() +
tm_format_NLD(frame = TRUE, asp=0)

# Facets defined by groupings
tm_shape(NLD_prov) +
    tm_borders() +
    tm_fill("gold2") +
    tm_facets(by="name") +
    tm_layout()

tm_shape(NLD_prov) +
    tm_fill("gold2") + tm_borders() +
    tm_facets(by="name", free.coords = TRUE, drop.shapes=TRUE) +
tm_layout()

tm_shape(NLD_muni) +
    tm_borders() +
    tm_facets(by="province") +
    tm_fill("population", style="kmeans", convert2density = TRUE) +
tm_shape(NLD_prov) +
    tm_borders(lwd=4) +
    tm_facets(by="name", free.coords=TRUE, drop.shapes=TRUE) +
tm_layout(legend.show = FALSE)
```

`tm_fill` *Draw polygons*

Description

Creates a tmap-element that draws the polygons. `tm_fill` fills the polygons. Either a fixed color is used, or a color palette is mapped to a data variable. By default, a divering color palette is used for numeric variables and a qualitative palette for categorical variables. `tm_borders` draws the borders of the polygons. `tm_polygons` fills the polygons and draws the polygon borders.

Usage

```
tm_fill(col = NA, alpha = NA, palette = NULL, convert2density = FALSE,
  area = NULL, n = 5, style = ifelse(is.null(breaks), "pretty", "fixed"),
  breaks = NULL, labels = NULL, auto.palette.mapping = TRUE,
  contrast = 1, max.categories = 12, colorNA = NA, textNA = "Missing",
  thres.poly = 1e-05, title = NA, legend.show = TRUE,
  legend.format = list(), legend.is.portrait = TRUE, legend.hist = FALSE,
  legend.hist.title = NA, legend.z = NA, legend.hist.z = NA, id = NA,
  ...)

tm_borders(col = NA, lwd = 1, lty = "solid", alpha = NA)

tm_polygons(col = NA, alpha = NA, border.col = NA, border.alpha = NA,
  ...)
```

Arguments

col	For tm_fill, it is one of • a single color value • the name of a data variable that is contained in shp. Either the data variable contains color values, or values (numeric or categorical) that will be depicted by a color palette (see palette. In the latter case, a choropleth is drawn. #' • "MAP_COLORING". In this case polygons will be colored such that adjacent polygons do not get the same color. See the underlying function map_coloring for details. For tm_borders, it is a single color value that specifies the border line color. If multiple values are specified, small multiples are drawn (see details).
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
palette	a palette name or a vector of colors. See RColorBrewer::display.brewer.all() for the named palette. Use a "-" as prefix to reverse the palette. The default palette is taken from tm_layout's argument aes.palette.
convert2density	boolean that determines whether col is converted to a density variable. Should be TRUE when col consists of absolute numbers. The area size is either approximated from the shape object, or given by the argument area.
area	Name of the data variable that contains the area sizes in squared kilometer.
n	preferred number of classes (in case col is a numeric variable).
style	method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former maps

the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".

breaks in case style=="fixed", breaks should be specified.

labels labels of the classes.

auto.palette.mapping
When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values.

contrast vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when auto.palette.mapping=TRUE). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

max.categories in case col is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than max.categories, then levels are combined.

colorNA color used for missing values

textNA text used for missing values. Use NA to omit text for missing values in the legend

thres.poly number that specifies the threshold at which polygons are taken into account. The number itself corresponds to the proportion of the area sizes of the polygons to the total polygon size.

title title of the legend element

legend.show logical that determines whether the legend is shown

legend.format list of formatting options for the legend numbers. Only applicable if labels is undefined. Parameters are:

scientific Should the labels be formatted scientically? If so, square brackets are used, and the format of the numbers is "g". Otherwise, format="f", and text.separator, text.less.than, and text.or.more are used. Also, the numbers are automatically rounded to millions or billions if applicable.

format By default, "f", i.e. the standard notation xxx.xxx, is used. If scientific=TRUE then "g", which means that numbers are formatted scientically, i.e. n.dddE+nn if needed to save space.

digits Number of digits after the decimal point if format="f", and the number of significant digits otherwise.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character string to use to translate "Less than" (which is the default).

text.or.more Character string to use to translate "or more" (which is the default).

... Other arguments passed on to formatC

legend.is.portrait
: logical that determines whether the legend is in portrait mode (TRUE) or landscape (FALSE)

legend.hist
: logical that determines whether a histogram is shown

legend.hist.title
: title for the histogram. By default, one title is used for both the histogram and the normal legend.

legend.z
: index value that determines the position of the legend element with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

legend.hist.z
: index value that determines the position of the histogram legend element

id
: name of the data variable that specifies the indices of the polygons. Only used for SVG output (see itmap).

...
: for tm_polygons, these arguments passed to either tm_fill or tm_borders. For tm_fill, these arguments are passed on to map_coloring.

lwd
: border line width (see par)

lty
: border line type (see par)

border.col
: border line color

border.alpha
: transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).

Details

Small multiples can be drawn in two ways: either by specifying the by argument in tm_facets, or by defining multiple variables in the aesthetic arguments. The aesthetic argument of tm_fill (and tm_polygons) is col. In the latter case, the arguments, except for thres.poly, and the ones starting with legend., can be specified for small multiples as follows. If the argument normally only takes a single value, such as n, then a vector of those values can be specified, one for each small multiple. If the argument normally can take a vector, such as palette, then a list of those vectors (or values) can be specified, one for each small multiple.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(World, Europe, NLD_muni, NLD_prov)

# Constant fill
tm_shape(World) + tm_fill("darkolivegreen3") + tm_format_World(title="A green World")
```

```
# Borders only
tm_shape(Europe) + tm_borders()

# Data variable containing colours values
Europe$isNLD <- ifelse(Europe$name=="Netherlands", "darkorange", "darkolivegreen3")
tm_shape(Europe) +
    tm_fill("isNLD") +
tm_layout("Find the Netherlands!")

# Numeric data variable
tm_shape(NLD_muni) +
    tm_fill(col="population", convert2density=TRUE,
        style="kmeans", title="Population (per km2)", legend.hist=TRUE) +
    tm_borders("grey25", alpha=.5) +
tm_shape(NLD_prov) +
    tm_borders("grey40", lwd=2) +
tm_format_NLD_wide(bg.color="white", frame = FALSE, legend.hist.bg.color="grey90")

tm_shape(World) +
    tm_polygons("HPI", palette="RdYlGn", style="cont", n=8, auto.palette.mapping=FALSE,
     title="Happy Planet Index") +
    tm_text("iso_a3", size="AREA", scale=1.5) +
tm_format_World() +
tm_style_grey()

# Categorical data variable
tm_shape(World) +
    tm_polygons("economy", title="Economy") +
    tm_text("iso_a3", size="AREA", scale=1.5) +
tm_format_World()

# Map coloring algorithm
tm_shape(NLD_prov) +
    tm_fill("name", legend.show = FALSE) +
tm_shape(NLD_muni) +
    tm_polygons("MAP_COLORS", palette="Greys", alpha = .25) +
tm_shape(NLD_prov) +
    tm_borders(lwd=2) +
    tm_text("name", shadow=TRUE) +
tm_format_NLD(title="Dutch provinces and\nmunicipalities", bg.color="white")
```

`tm_grid` *Coordinate grid lines*

Description

Creates a tmap-element that draws coordinate grid lines. It serves as a layer that can be drawn anywhere between other layers. By default the coordinate system of the (master) shape object is used, which results in horizontal and vertical lines. Alternatively, grid lines can be reprojected, for instance to latitude longitude coordinates, and hence be curved.

Usage

```
tm_grid(n.x = NA, n.y = NA, projection = NA, col = NA, lwd = 1,
  alpha = NA, labels.size = 0.6, labels.col = NA,
  labels.inside.frame = TRUE)
```

Arguments

n.x	prefered number of grid lines for the x axis.
n.y	prefered number of grid lines for the y axis.
projection	projection character. If specified, the grid lines are projected accordingly. See set_projection for projection details. Many world maps are projected, but still have latitude longitude ("longlat") grid lines.
col	color of the grid lines.
lwd	line width of the grid lines
alpha	alpha transparency of the grid lines. Number between 0 and 1. By default, the alpha transparency of col is taken.
labels.size	font size of the tick labels
labels.col	font color of the tick labels
labels.inside.frame	Show labels inside the frame?

---

tm_iso *Draw iso (contour) lines with labels*

Description

This function is a wrapper of tm_lines and tm_text aimed to draw isopleths, which can be created with smooth_map.

Usage

```
tm_iso(col = NA, text = "level", size = 0.5, remove.overlap = TRUE,
  along.lines = TRUE, overwrite.lines = TRUE, ...)
```

Arguments

col	line color. See tm_lines.
text	text to display. By default, it is the variable named "level" of the shape that is created with smooth_map
size	text size (see tm_text)
remove.overlap	see tm_text
along.lines	see tm_text
overwrite.lines	see tm_text
...	arguments passed on to tm_lines or tm_text

See Also

smooth_map

---

tm_layout *Layout of cartographic maps*

Description

This element specifies the map layout. The main function tm_layout controlls title, margins, aspect ratio, colors, frame, legend, among many other things. The function tm_legend is a shortcut to access all legend. arguments without this prefix. The other functions are wrappers for two purposes: the tm_format_ functions specify position related layout settings such as margins, and the tm_style_ functions specify general styling related layout settings such as colors and font. Typically, the former functions are shape dependent, and the latter functions are shape independent. See details for predefined styles and formats. With the global option tmap.style, a default style can be specified. Multiple tm_layout elements (or wrapper functions) can be stacked: called arguments will be overwritten.

Usage

```
tm_layout(title = NA, scale = 1, title.size = 1.3, bg.color = "white",
  aes.color = c(fill = "grey85", borders = "grey40", bubbles = "blueviolet",
  dots = "black", lines = "red", text = "black", na = "grey70"),
  aes.palette = list(seq = "YlOrBr", div = "RdYlGn", cat = "Set3"),
  attr.color = "black", sepia.intensity = 0, saturation = 1,
  frame = TRUE, frame.lwd = 1, frame.double.line = FALSE, asp = NA,
  outer.margins = rep(0.02, 4), inner.margins = NA, outer.bg.color = NULL,
  fontface = "plain", fontfamily = "sans", compass.type = "arrow",
  earth.boundary = FALSE, earth.boundary.color = attr.color,
  earth.boundary.lwd = 1, earth.datum = "WGS84", space.color = NULL,
  legend.show = TRUE, legend.only = FALSE, legend.position = NULL,
  legend.width = 0.3, legend.height = 0.9, legend.hist.height = 0.3,
  legend.hist.width = legend.width, legend.title.size = 1.1,
  legend.text.size = 0.7, legend.hist.size = 0.7,
  legend.format = list(scientific = FALSE, digits = NA, text.separator = "to",
  text.less.than = "Less than", text.or.more = "or more"),
  legend.frame = FALSE, legend.text.color = attr.color,
  legend.bg.color = NA, legend.bg.alpha = 1, legend.hist.bg.color = NA,
  legend.hist.bg.alpha = 1, title.snap.to.legend = FALSE,
  title.position = c("left", "top"), title.color = attr.color,
  title.bg.color = NA, title.bg.alpha = 1, attr.position = c("right",
  "bottom"), design.mode = FALSE)

tm_legend(...)

tm_format_World(title = NA, inner.margins = c(0, 0.05, 0.025, 0.01),
```

```
  legend.position = c("left", "bottom"), attr.position = c("right",
  "bottom"), scale = 0.8, ...)

tm_format_World_wide(title = NA, inner.margins = c(0, 0.2, 0.025, 0.01),
  legend.position = c("left", "bottom"), attr.position = c("right",
  "bottom"), legend.width = 0.4, scale = 0.8, ...)

tm_format_Europe(title = NA, title.position = c("left", "top"),
  legend.position = c("left", "top"), attr.position = c("left", "bottom"),
  inner.margins = c(0, 0.1, 0, 0), ...)

tm_format_Europe_wide(title = NA, title.position = c("left", "top"),
  legend.position = c("left", "top"), attr.position = c("left", "bottom"),
  inner.margins = c(0, 0.25, 0, 0), legend.width = 0.4,
  legend.hist.width = 0.4, ...)

tm_format_NLD(title = NA, frame = FALSE, inner.margins = c(0.02, 0.2,
  0.06, 0.02), legend.position = c("left", "top"), legend.width = 0.4,
  attr.position = c("left", "bottom"), ...)

tm_format_NLD_wide(title = NA, frame = FALSE, inner.margins = c(0.02, 0.3,
  0.06, 0.02), legend.position = c("left", "top"), legend.width = 0.5,
  legend.hist.width = 0.35, attr.position = c("left", "bottom"), ...)

tm_style_white(...)

tm_style_gray(bg.color = "grey85", aes.color = c(fill = "grey70", borders =
  "grey20", bubbles = "blueviolet", dots = "black", lines = "red", text =
  "black", na = "grey60"), ...)

tm_style_natural(bg.color = "lightskyblue1", aes.color = c(fill =
  "darkolivegreen3", borders = "black", bubbles = "tomato2", dots = "firebrick",
  lines = "steelblue", text = "black", na = "white"), aes.palette = list(seq =
  "YlGn", div = "RdYlGn", cat = "Set3"), attr.color = "black",
  space.color = "white", earth.boundary = TRUE, ...)

tm_style_grey(bg.color = "grey85", aes.color = c(fill = "grey70", borders =
  "grey20", bubbles = "blueviolet", dots = "black", lines = "red", text =
  "black", na = "grey60"), ...)

tm_style_cobalt(bg.color = "#002240", aes.color = c(fill = "#0088FF",
  borders = "#002240", bubbles = "#FF9D00", dots = "#FF9D00", lines = "#FFEE80",
  text = "white", na = "grey60"), aes.palette = list(seq = "YlGn", div =
  "RdYlGn", cat = "Set3"), attr.color = "white", ...)

tm_style_col_blind(bg.color = "white", aes.color = c(fill = "grey85",
  borders = "black", bubbles = "#D55E00", dots = "#0072B2", lines = "#009E73",
  text = "black", na = "white"), aes.palette = list(seq = "Blues", div =
```

```
  "RdBu", cat = c("#D55E00", "#56B4E9", "#E69F00", "#009E73", "#F0E442",
  "#0072B2", "#CC79A7")), attr.color = "black", ...)

tm_style_albatross(bg.color = "#00007F", aes.color = c(fill = "#4C4C88",
  borders = "#00004C", bubbles = "#BFBFFF", dots = "#BFBFFF", lines = "#BFBFFF",
  text = "#FFE700", na = "grey60"), aes.palette = list(seq = "YlOrRd", div =
  "RdYlGn", cat = "Set3"), attr.color = "#BFBFFF", ...)

tm_style_beaver(bg.color = "#FFFFFF", aes.color = c(fill = "#FFE200",
  borders = "#000000", bubbles = "#A30000", dots = "#A30000", lines = "#A30000",
  text = "#000000", na = "#E0E0E0"), aes.palette = list(seq = "YlOrBr", div =
  "RdYlGn", cat = "Dark2"), attr.color = "black", ...)

tm_style_bw(saturation = 0, ...)

tm_style_classic(sepia.intensity = 0.7, fontfamily = "serif",
  frame.double.line = TRUE, compass.type = "rose", ...)
```

Arguments

title	Title(s). By default, the name of the statistical variable of which the legend is drawn at the top (see legend.config) is used as a title.
scale	numeric value that serves as the global scale parameter. All font sizes, bubble sizes, border widths, and line widths are controled by this value. Each of these elements can be scaled independantly with the scale, lwd, or size arguments provided by the tmap-elements.
title.size	Relative size of the title
bg.color	Background color. By default it is "white". A recommended alternative for choropleths is light grey (e.g., "grey85").
aes.color	Default color values for the aesthetics layers. Should be a named vector with the names chosen from: fill, borders, bubbles, dots, lines, text, na.
aes.palette	Default color palettes for the aesthetics. It takes a list of three items: seq for sequential palettes, div for diverging palettes, and cat for categorical palettes. By default, Color Brewer palettes (see (see RColorBrewer::display.brewer.all)) are used. It is also possible provide a vector of colors for any of these items.
attr.color	Default color value for map attributes
sepia.intensity	Number between 0 and 1 that defines the amount of sepia effect, which gives the map a brown/yellowish flavour. By default this effect is disabled (sepia.intensity=0). All colored used in the map are adjusted with this effect.
saturation	Number that determines how much saturation (also known as chroma) is used: saturation=0 is greyscale and saturation=1 is normal. A number larger than 1 results in very saturated maps. All colored used in the map are adjusted with this effect. Hacking tip: use a negative number.
frame	Either a boolean that determines whether a frame is drawn, or a color value that specifies the color of the frame.

`frame.lwd` width of the frame

`frame.double.line`
draw a double frame line border?

`asp` Aspect ratio. The aspect ratio of the map (width/height). If `NA`, it is determined by the bounding box (see argument `bbox` of `tm_shape`), the `outer.margins`, and the `inner.margins`. If `0`, then the aspect ratio is adjusted to the aspect ratio of the device.

`outer.margins` Relative margins between device and frame. Vector of four values specifying the bottom, left, top, and right margin. Values are between 0 and 1.

`inner.margins` Relative margins inside the frame. Vector of four values specifying the bottom, left, top, and right margin. Values are between 0 and 1. By default, 0 for each side if master shape is a raster, otherwise 0.02.

`outer.bg.color` Background color outside the frame.

`fontface` font face of all text in the map.

`fontfamily` font family of the text labels.

`compass.type` type of compass, one of: `"arrow"`, `"4star"`, `"8star"`, `"radar"`, `"rose"`. Of course, only applicable if a compass is shown. The compass type can also be set within `tm_compass`.

`earth.boundary` Logical that determines whether the boundaries of the earth are shown or an object that specifies the boundaries. This object can be a vector of size four, a 2 by 2 matrix (bounding box), or an `extent` object. By default, the boundaries are `c(-180, 180, -90, 90)`. Useful for rojected world maps. Often, it is useful to crop both poles (e.g., with `c(-180, 180, -88, 88)`).

`earth.boundary.color`
Color of the earth boundary.

`earth.boundary.lwd`
Line width of the earth boundary.

`earth.datum` Geodetic datum to determine the earth boundary. By default `"WGS84"`, other frequently used datums are `"NAD83"` and `"NAD27"`. Any other `PROJ.4` character string can be used.

`space.color` Color of the space, i.e. the region inside the frame, and outsise the earth boundary.

`legend.show` Logical that determines whether the legend is shown.

`legend.only` logical. Only draw the legend (without map)? Particularly useful for small multiples with a common legend.

`legend.position`
Position of the legend. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y coordinates of the left bottom corner of the legend. The uppercase values correspond to the position without margins (so tighter to the frame). By default, it is automatically placed in the corner with most space based on the (first) shape object.

`legend.width` maximum width of the legend

`legend.height` maximum height of the legend.

`legend.hist.height`
height of the histogram. This hight is initial. If the total legend is downscaled to `legend.height`, the histogram is downscaled as well.

`legend.hist.width`
width of the histogram. By default, it is equal to the `legend.width`.

`legend.title.size`
Relative font size for the legend title

`legend.text.size`
Relative font size for the legend text elements

`legend.hist.size`
Relative font size for the choropleth histogram

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. Parameters are:

scientific Should the labels be formatted scientically? If so, square brackets are used, and the `format` of the numbers is `"g"`. Otherwise, `format="f"`, and `text.separator`, `text.less.than`, and `text.or.more` are used. Also, the numbers are automatically rounded to millions or billions if applicable.

format By default, `"f"`, i.e. the standard notation `xxx.xxx`, is used. If `scientific=TRUE` then `"g"`, which means that numbers are formatted scientically, i.e. `n.dddE+nn` if needed to save space.

digits Number of digits after the decimal point if `format="f"`, and the number of significant digits otherwise.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character string to use to translate "Less than" (which is the default).

text.or.more Character string to use to translate "or more" (which is the default).

... Other arguments passed on to `formatC`

`legend.frame` either a logical that determines whether the legend is placed inside a frame, or a color that directly specifies the frame border color. The width of the frame is automatically determined, but is upper-bounded by `legend.width`.

`legend.text.color`
color of the legend text

`legend.bg.color`
Background color of the legend. Use `TRUE` to match with the overall background color `bg.color`.

`legend.bg.alpha`
Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the `legend.bg.color` is used (normally 1).

`legend.hist.bg.color`
Background color of the histogram

`legend.hist.bg.alpha`
: Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the `legend.hist.bg.color` is used (normally 1).

`title.snap.to.legend`
: Logical that determines whether the title is part of the legend.

`title.position`
: Position of the title. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y coordinates of the tile. The uppercase values correspond to the position without margins (so tighter to the frame). By default the title is placed on top of the legend (determined by `legend.position`).

`title.color`
: color of the title

`title.bg.color`
: background color of the title. Use `TRUE` to match with the overall background color `bg.color`.

`title.bg.alpha`
: Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the `title.bg.color` is used (normally 1).

`attr.position`
: Position of the map attributes, which are `tm_credits`, `tm_scale_bar` and `tm_compass`. Vector of two values, specifing the x and y coordinates. The first value is "left", "LEFT", "center", "right", or "RIGHT", and the second value "top", "TOP", "center", "bottom", or "BOTTOM". The uppercase values correspond to the position without margins (so tighter to the frame). Positions can also be set separately in the map attribute fuctions.

`design.mode`
: Logical that enables the design mode. If `TRUE`, inner and outer margins, legend position, aspect ratio are explicitely shown. Also, feedback text in the console is given.

`...`
: other arguments from `tm_layout`

Details

Predefined styles:

`tm_style_white`	White background, commonly used colors (default)
`tm_style_grey/_grey`	Grey background, useful to highlight sequential palettes (e.g. in choropleths)
`tm_style_natural`	Emulation of natural view: blue waters and green land
`tm_style_bw`	Greyscale, obviously useful for greyscale printing
`tm_style_classic`	Classic styled maps (recommended)
`tm_style_cobalt`	Inspired by latex beamer style cobalt
`tm_style_albatross`	Inspired by latex beamer style cobalt
`tm_style_beaver`	Inspired by latex beamer style beaver

Predefined formats

`tm_format_World`	Format specified for world maps
`tm_format_World_wide`	Format specified for world maps with more space for the legend

tm_format_Europe	Format specified for maps of Europe
tm_format_Europe_wide	Format specified for maps of Europe with more space for the legend
tm_format_NLD	Format specified for maps of the Netherlands
tm_format_NLD_wide	Format specified for maps of the Netherlands with more space for the legend

See Also

vignette("tmap-nutshell")

Examples

```
data(World, land)

tm_shape(World) +
    tm_fill("pop_est_dens", style="kmeans", title="Population density") +
tm_format_World(title="The World") + tm_style_albatross(frame.lwd=10)

land_eck4 <- set_projection(land, "eck4") # convert to Eckert IV projection

tm_shape(land_eck4) +
tm_raster("elevation", breaks=c(-Inf, 250, 500, 1000, 1500, 2000, 2500, 3000, 4000, Inf),
palette = terrain.colors(9), title="Elevation", auto.palette.mapping = FALSE) +
tm_shape(World) +
tm_borders("grey20") +
tm_grid(projection="longlat", labels.size = .5) +
tm_text("name", size="AREA") +
tm_compass(position = c(.65, .15), color.light = "grey90") +
tm_credits("Eckert IV projection", position = c("right", "BOTTOM")) +
tm_layout(inner.margins=c(.04,.03, .02, .01),
earth.boundary = TRUE,
space.color="grey90") +
tm_style_classic(bg.color="lightblue") +
tm_legend(position = c("left", "bottom"),
frame = TRUE,
bg.color="lightblue")

WorldOne <- rgeos::gUnaryUnion(World)
tm_shape(World, projection="wintri") +
tm_fill("HPI", palette="div", auto.palette.mapping = FALSE, n=7,
title = "Happy Planet Index") +
tm_shape(WorldOne) +
tm_borders() +
tm_grid(projection = "longlat") +
tm_credits("Winkel Tripel projection", position = c("right", "BOTTOM")) +
tm_style_natural(earth.boundary = c(-180,180,-87,87), inner.margins = .05) +
tm_legend(position=c("left", "bottom"), bg.color="grey95", frame=TRUE)

## Not run:
# the last three elements (tm_layout, tm_style_classic, and tm_legend)
# can also be merged into one element function:
```

```
tm_style_classic(inner.margins=c(.04,.03, .02, .01),
earth.boundary = TRUE,
space.color="grey90",
bg.color="lightblue",
legend.position = c("left", "bottom"),
legend.frame = TRUE,
legend.bg.color="lightblue")

## End(Not run)

## Not run:
# global option tmap.style:
qtm(World, fill="economy", format="World")
options(tmap.style = "col_blind")
qtm(World, fill="economy", format="World")
options(tmap.style = "cobalt")
qtm(World, fill="economy", format="World")
options(tmap.style = "white") # the default

## End(Not run)
```

`tm_lines` *Draw spatial lines*

Description

Creates a `tmap-element` that draw spatial lines.

Usage

```
tm_lines(col = NA, lwd = 1, lty = "solid", alpha = NA, scale = 1,
  lwd.legend = NULL, lwd.legend.labels = NULL, n = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  palette = NULL, labels = NULL, auto.palette.mapping = TRUE,
  contrast = 1, max.categories = 12, colorNA = NA, textNA = "Missing",
  title.col = NA, title.lwd = NA, legend.col.show = TRUE,
  legend.lwd.show = TRUE, legend.format = list(),
  legend.col.is.portrait = TRUE, legend.lwd.is.portrait = FALSE,
  legend.hist = FALSE, legend.hist.title = NA, legend.col.z = NA,
  legend.lwd.z = NA, legend.hist.z = NA, id = NA)
```

Arguments

`col`	color of the lines. Either a color value or a data variable name. If multiple values are specified, small multiples are drawn (see details).
`lwd`	line width. If multiple values are specified, small multiples are drawn (see details).
`lty`	line type.

alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
scale	line width multiplier number.
lwd.legend	vector of line widths that are shown in the legend. By default, this is determined automatically.
lwd.legend.labels	vector of labels for that correspond to lwd.legend.
n	preferred number of color scale classes. Only applicable when lwd is the name of a numeric variable.
style	method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former maps the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".
breaks	in case style=="fixed", breaks should be specified
palette	color palette (see RColorBrewer::display.brewer.all) for the lines. Only when col is set to a variable. The default palette is taken from tm_layout's argument aes.palette.
labels	labels of the classes
auto.palette.mapping	When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values. In this case of line widths, obviously only the positive side is used.
contrast	vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when auto.palette.mapping=TRUE). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).
max.categories	in case col is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than max.categories, then levels are combined.
colorNA	color used for missing values
textNA	text used for missing values. Use NA to omit text for missing values in the legend
title.col	title of the legend element regarding the line colors
title.lwd	title of the legend element regarding the line widths
legend.col.show	logical that determines whether the legend for the line colors is shown

`legend.lwd.show`
: logical that determines whether the legend for the line widths is shown

`legend.format`
: list of formatting options for the legend numbers. Only applicable if `labels` is undefined. Parameters are:

  scientific Should the labels be formatted scientically? If so, square brackets are used, and the `format` of the numbers is `"g"`. Otherwise, `format="f"`, and `text.separator`, `text.less.than`, and `text.or.more` are used. Also, the numbers are automatically rounded to millions or billions if applicable.

  format By default, `"f"`, i.e. the standard notation `xxx.xxx`, is used. If `scientific=TRUE` then `"g"`, which means that numbers are formatted scientically, i.e. `n.dddE+nn` if needed to save space.

  digits Number of digits after the decimal point if `format="f"`, and the number of significant digits otherwise.

  text.separator Character string to use to separate numbers in the legend (default: "to").

  text.less.than Character string to use to translate "Less than" (which is the default).

  text.or.more Character string to use to translate "or more" (which is the default).

  ... Other arguments passed on to `formatC`

`legend.col.is.portrait`
: logical that determines whether the legend element regarding the line colors is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.lwd.is.portrait`
: logical that determines whether the legend element regarding the line widths is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.hist`
: logical that determines whether a histogram is shown regarding the line colors

`legend.hist.title`
: title for the histogram. By default, one title is used for both the histogram and the normal legend for line colors.

`legend.col.z`
: index value that determines the position of the legend element regarding the line colors with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

`legend.lwd.z`
: index value that determines the position of the legend element regarding the line widths. (See `legend.col.z`)

`legend.hist.z`
: index value that determines the position of the legend element regarding the histogram. (See `legend.col.z`)

`id`
: name of the data variable that specifies the indices of the lines. Only used for SVG output (see `itmap`).

Details

Small multiples can be drawn in two ways: either by specifying the `by` argument in `tm_facets`, or by defining multiple variables in the aesthetic arguments. The aesthetic arguments of `tm_lines` are

col and lwd. In the latter case, the arguments, except for the ones starting with legend., can be specified for small multiples as follows. If the argument normally only takes a single value, such as n, then a vector of those values can be specified, one for each small multiple. If the argument normally can take a vector, such as palette, then a list of those vectors (or values) can be specified, one for each small multiple.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(World, Europe, rivers)

qtm(rivers, line.col = "navy")

tm_shape(Europe) +
  tm_fill("MAP_COLORS", palette = "Pastel2") +
tm_shape(rivers) +
  tm_lines(col="black", lwd="scalerank", scale=2, legend.lwd.show = FALSE) +
tm_layout("Rivers of Europe") +
tm_style_cobalt()
```

---

tm_raster *Draw a raster*

Description

Creates a tmap-element that draws a raster. Either a fixed color is used, or a color palette is mapped to a data variable. By default, a divering color palette is used for numeric variables and a qualitative palette for categorical variables.

Usage

```
tm_raster(col = NA, alpha = NA, palette = NULL, n = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  labels = NULL, auto.palette.mapping = TRUE, contrast = 1,
  max.categories = 12, colorNA = NA, saturation = 1, textNA = "Missing",
  title = NA, legend.show = TRUE, legend.format = list(),
  legend.is.portrait = TRUE, legend.hist = FALSE, legend.hist.title = NA,
  legend.z = NA, legend.hist.z = NA)
```

Arguments

col	either a single color value or the name of a data variable that is contained in shp. In the latter case, either the data variable contains color values, or values (numeric or categorical) that will be depicted by a color palette (see palette. If multiple values are specified, small multiples are drawn (see details). By default, it is the name of the first data variable.
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
palette	palette name. See RColorBrewer::display.brewer.all() for options. Use a "-" as prefix to reverse the palette. The default palette is taken from tm_layout's argument aes.palette.
n	preferred number of classes (in case col is a numeric variable)
style	method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former maps the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".
breaks	in case style=="fixed", breaks should be specified
labels	labels of the classes
auto.palette.mapping	When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values.
contrast	vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when auto.palette.mapping=TRUE). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).
max.categories	in case col is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than max.categories, then levels are combined.
colorNA	color used for missing values
saturation	Number that determines how much saturation (also known as chroma) is used: saturation=0 is greyscale and saturation=1 is normal. This saturation value is multiplied by the overall saturation of the map (see tm_layout).
textNA	text used for missing values. Use NA to omit text for missing values in the legend
title	title of the legend element

`legend.show` logical that determines whether the legend is shown

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. Parameters are:

scientific Should the labels be formatted scientically? If so, square brackets are used, and the `format` of the numbers is `"g"`. Otherwise, `format="f"`, and `text.separator`, `text.less.than`, and `text.or.more` are used. Also, the numbers are automatically rounded to millions or billions if applicable.

format By default, `"f"`, i.e. the standard notation `xxx.xxx`, is used. If `scientific=TRUE` then `"g"`, which means that numbers are formatted scientically, i.e. `n.dddE+nn` if needed to save space.

digits Number of digits after the decimal point if `format="f"`, and the number of significant digits otherwise.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character string to use to translate "Less than" (which is the default).

text.or.more Character string to use to translate "or more" (which is the default).

... Other arguments passed on to `formatC`

`legend.is.portrait` logical that determines whether the legend is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.hist` logical that determines whether a histogram is shown

`legend.hist.title` title for the histogram. By default, one title is used for both the histogram and the normal legend.

`legend.z` index value that determines the position of the legend element with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

`legend.hist.z` index value that determines the position of the histogram legend element

Details

Small multiples can be drawn in two ways: either by specifying the `by` argument in `tm_facets`, or by defining multiple variables in the aesthetic arguments. The aesthetic argument of `tm_raster` is `col`. In the latter case, the arguments, except for `thres.poly`, and the ones starting with `legend.`, can be specified for small multiples as follows. If the argument normally only takes a single value, such as `n`, then a vector of those values can be specified, one for each small multiple. If the argument normally can take a vector, such as `palette`, then a list of those vectors (or values) can be specified, one for each small multiple.

Value

`tmap-element`

See Also

vignette("tmap-nutshell")

Examples

```
data(World, land)

pal20 <- c("#003200", "#3C9600", "#006E00", "#556E19", "#00C800", "#8CBE8C",
     "#467864", "#B4E664", "#9BC832", "#EBFF64", "#F06432", "#9132E6",
     "#E664E6", "#9B82E6", "#B4FEF0", "#646464", "#C8C8C8", "#FF0000",
     "#FFFFFF", "#5ADCDC")
tm_shape(land) +
    tm_raster("cover", max.categories = 20, palette=pal20, title="Global Land Cover") +
tm_layout(scale=.8, legend.position = c("left","bottom"))

pal8 <- c("#33A02C", "#B2DF8A", "#FDBF6F", "#1F78B4", "#999999", "#E31A1C", "#E6E6E6", "#A6CEE3")
tm_shape(land, ylim = c(-88,88)) +
    tm_raster("cover_cls", palette = pal8, title="Global Land Cover") +
tm_shape(World) +
    tm_borders(col="black") +
tm_layout(scale=.8,
legend.position = c("left","bottom"),
    legend.bg.color = "white", legend.bg.alpha=.2,
    legend.frame="gray50")

tm_shape(land, ylim = c(-88,88)) +
    tm_raster("trees", palette = "Greens", title="Percent Tree Cover") +
tm_shape(World) +
    tm_borders() +
tm_layout(legend.position = c("left","bottom"), bg.color="lightblue")

## Not run:
tm_shape(land) +
tm_raster("black") +
tm_facets(by="cover_cls") +
tm_layout(title.position = c("left", "bottom"), title.bg.color="gray80")

## End(Not run)
```

---

tm_scale_bar *Scale bar*

Description

Creates a scale bar. By default, the coordinate units are assumed to be meters, and the map units in kilometers. This can be changed in tm_shape.

Usage

```
tm_scale_bar(breaks = NULL, width = 0.25, size = 0.5, text.color = NA,
  color.dark = "black", color.light = "white", position = NA)
```

Arguments

breaks	breaks of the scale bar. If not specified, breaks will be automatically be chosen given the prefered width of the scale bar.
width	(prefered) width of the scale bar. Only applicable when breaks=N ULL
size	relative text size (which is upperbound by the available label width)
text.color	color of the text. By default equal to the argument attr.color of tm_layout.
color.dark	color of the dark parts of the scale bar, typically (and by default) black.
color.light	color of the light parts of the scale bar, typically (and by default) white.
position	position of the text. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y value of the left bottom corner of the scale bar. The uppercase values correspond to the position without margins (so tighter to the frame). The default value is controlled by the argument "attr.position" of tm_layout.

Examples

```
data(NLD_muni)
qtm(NLD_muni, theme = "NLD") + tm_scale_bar(position=c("left", "bottom"))

data(Europe)
tm_shape(Europe, unit = "miles", unit.size=1609) + tm_polygons() + tm_scale_bar()
```

---

tm_shape *Specify the shape object*

Description

Creates a tmap-element that specifies the shape object. Also the projection and covered area (bounding box) can be set. It is possible to use multiple shape objects within one plot (see tmap-element).

Usage

```
tm_shape(shp, is.master = NA, projection = NULL, bbox = NULL,
  unit = "km", unit.size = 1000, line.center.type = c("segment",
  "midpoint"), ...)
```

Arguments

shp	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame) 4. SpatialGrid(DataFrame)

5. SpatialPixels(DataFrame)
6. RasterLayer, RasterStack, or RasterBrick

For drawing layers tm_fill and tm_borders, 1 is required. For drawing layer tm_lines, 3 is required. Layers tm_bubbles and tm_text accept 1 to 3. For layer tm_raster, 4, 5, or 6 is required.

`is.master` logical that determines whether this tm_shape is the master shape element. The bounding box, projection settings, and the unit specifications of the resulting thematic map are taken from the tm_shape element of the master shape object. By default, the first master shape element with a raster shape is the master, and if there are no raster shapes used, then the first tm_shape is the master shape element.

`projection` character that determines the projection. Either a PROJ.4 character string or a shortcut. See get_proj4 for a list of shortcut values. By default, the projection is used that is defined in the shp object itself, which can be obtained with get_projection.

`bbox` bounding box, which is a 2x2 matrix. See also ...

`unit` unit specification. Needed when calculating density values in choropleth maps (argument convert2density in tm_fill) drawing a scale bar with tm_scale_bar. See also unit.size.

`unit.size` size of the unit in terms of coordinate units. The coordinate system of many projections is approximately in meters while thematic maps typically range many kilometers, so by default unit="km" and unit.size=1000 (meaning 1 kilometer equals 1000 coordinate units).

`line.center.type`
vector of two values specifying how the center of spatial lines is determined Only applicable if shp is a SpatialLines(DataFrame), and bubbles, dots, and/or text labels are used for this shape. The two values are:

"feature", "single" If "feature" is specified, a pair of coordinates (used for bubbles, dots, and text labels) is chosen for each feature (i.e., a row in the SpatialLines(DataFrame)). If "segment" is specified, a pair of coordinates is chosed for each line segment.

"midpoint" **or** "centroid" The midpoint is the middle point on the line, so the coordinates (used for bubbles, dots, and text labels) correspond to the midpoints of the line segments. In case the first value is "feature", then per feature, the midpoint of the line segment that is closest to the centroid is taken.

`...` Arguments passed on to bb, which creates a bounding box. The first argument that will be passed on bb is either shp, or bbox if it is specified.

Value

tmap-element

See Also

read_shape to read ESRI shape files, set_projection, vignette("tmap-nutshell")

Examples

```
data(World, metro, rivers)

tm_shape(World, projection="longlat") +
    tm_polygons() +
tm_layout("Long lat coordinates (WGS84)", inner.margins=c(0,0,.1,0), title.size=.8)

World$highlighted <- ifelse(World$iso_a3 %in% c("GRL", "AUS"), "gold", "gray75")
tm_shape(World, projection="merc", ylim=c(.1, 1), relative = TRUE) +
    tm_polygons("highlighted") +
tm_layout("Mercator projection. Although used in Google Maps, it is discouraged for
statistical purposes. In reality, Australia is 3 times larger than Greenland!",
    inner.margins=c(0,0,.1,0), title.size=.6)

tm_shape(World, projection="wintri") +
    tm_polygons() +
tm_layout(
"Winkel-Tripel projection, adapted as default by the National Geographic Society for world maps.",
    inner.margins=c(0,0,.1,0), title.size=.8)

tm_shape(World) +
    tm_polygons() +
tm_layout("Eckhart IV projection. Recommended in statistical maps for its equal-area property.",
    inner.margins=c(0,0,.1,0), title.size=.8)

# three groups of layers, each starting with tm_shape
tm_shape(World) +
    tm_fill("darkolivegreen3") +
tm_shape(metro) +
    tm_bubbles("pop2010", col = "grey30", scale=.5) +
tm_shape(rivers) +
    tm_lines("lightcyan1") +
tm_layout(bg.color="lightcyan1", inner.margins=c(0,0,.02,0), legend.show = FALSE)
```

---

tm_text *Add text labels*

Description

Creates a tmap-element that adds text labels.

Usage

```
tm_text(text, size = 1, col = NA, root = 3, size.lim = NA,
  sizes.legend = NULL, sizes.legend.labels = NULL,
  sizes.legend.text = "Abc", n = 5, style = ifelse(is.null(breaks),
  "pretty", "fixed"), breaks = NULL, palette = NULL, labels = NULL,
  labels.text = NA, auto.palette.mapping = TRUE, contrast = 1,
  max.categories = 12, colorNA = NA, textNA = "Missing", fontface = NA,
```

```
  fontfamily = NA, alpha = NA, case = NA, shadow = FALSE,
  bg.color = NA, bg.alpha = NA, size.lowerbound = 0.4,
  print.tiny = FALSE, scale = 1, auto.placement = FALSE,
  remove.overlap = FALSE, along.lines = FALSE, overwrite.lines = FALSE,
  xmod = 0, ymod = 0, title.size = NA, title.col = NA,
  legend.size.show = TRUE, legend.col.show = TRUE, legend.format = list(),
  legend.size.is.portrait = FALSE, legend.col.is.portrait = TRUE,
  legend.hist = FALSE, legend.hist.title = NA, legend.size.z = NA,
  legend.col.z = NA, legend.hist.z = NA, id = NA)
```

Arguments

text	name of the variable in the shape object that contains the text labels
size	relative size of the text labels (see note). Eiter one number, a name of a numeric variable in the shape data that is used to scale the sizes proportionally, or the value "AREA", where the text size is proportional to the area size of the polygons.
col	color of the text labels. Either a color value or a data variable name. If multiple values are specified, small multiples are drawn (see details).
root	root number to which the font sizes are scaled. Only applicable if size is a variable name or "AREA". If root=2, the square root is taken, if root=3, the cube root etc.
size.lim	vector of two limit values of the size variable. Only text labels are drawn whose value is greater than or equal to the first value. Text labels whose values exceed the second value are drawn at the size of the second value. Only applicable when size is the name of a numeric variable of shp. See also size.lowerbound which is a threshold of the relative font size.
sizes.legend	vector of text sizes that are shown in the legend. By default, this is determined automatically.
sizes.legend.labels	vector of labels for that correspond to sizes.legend.
sizes.legend.text	vector of example text to show in the legend next to sizes.legend.labels. By default "Abc". When NA, examples from the data variable whose sizes are close to the sizes.legend are taken and "NA" for classes where no match is found.
n	preferred number of color scale classes. Only applicable when col is a numeric variable name.
style	method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former maps the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".
breaks	in case style=="fixed", breaks should be specified

palette — color palette (see RColorBrewer::display.brewer.all) for the text. Only when col is set to a variable. The default palette is taken from tm_layout's argument aes.palette.

labels — labels of the color classes, applicable if col is a data variable name

labels.text — Example text to show in the legend next to the labels. When NA (default), examples from the data variable are taken and "NA" for classes where they don't exist.

auto.palette.mapping — When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values.

contrast — vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when auto.palette.mapping=TRUE). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

max.categories — in case col is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than max.categories, then levels are combined.

colorNA — colour for missing values

textNA — text used for missing values. Use NA to omit text for missing values in the legend

fontface — font face of the text labels. By default, determined by the fontface argument of tm_layout.

fontfamily — font family of the text labels. By default, determined by the fontfamily argument of tm_layout.

alpha — transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the fontcolor is used (normally 1).

case — case of the font. Use "upper" to generate upper-case text, "lower" to generate lower-case text, and NA to leave the text as is.

shadow — logical that determines whether a shadow is depicted behind the text. The color of the shadow is either white or yellow, depending of the fontcolor.

bg.color — background color of the text labels. By default, bg.color=NA, so no background is drawn.

bg.alpha — number between 0 and 1 that specifies the transparancy of the text background (0 is totally transparent, 1 is solid background).

size.lowerbound — lowerbound for size. Only applicable when size is not a constant. If print.tiny is TRUE, then all text labels which relative text is smaller than size.lowerbound are depicted at relative size size.lowerbound. If print.tiny is FALSE, then text labels are only depicted if their relative sizes are at least size.lowerbound (in other words, tiny labels are omitted).

`print.tiny` boolean, see `size.lowerbound`

`scale` text size multiplier, useful in case `size` is variable or `"AREA"`.

`auto.placement` logical (or numeric) that determines whether the labels are placed automatically. If `TRUE`, the labels are placed next to the coordinate points with as little overlap as possible using the simulated annealing algorithm. Therefore, it is recommended for labeling spatial dots or bubbles. If a numeric value is provided, this value acts as a parameter that specifies the distance between the coordinate points and the text labels in terms of text line heights.

`remove.overlap` logical that determines whether the overlapping labels are removed

`along.lines` logical that determines whether labels are rotated along the spatial lines. Only applicabel if a spatial lines shape is used.

`overwrite.lines`
logical that determines whether the part of the lines below the text labels is removed. Only applicabel if a spatial lines shape is used.

`xmod` horizontal position modification of the text (relatively): 0 means no modification, and 1 corresponds to the height of one line of text. Either a single number for all polygons, or a numeric variable in the shape data specifying a number for each polygon. Together with `ymod`, it determines position modification of the text labels. In most coordinate systems (projections), the origin is located at the bottom left, so negative `xmod` move the text to the left, and negative `ymod` values to the bottom.

`ymod` vertical position modification. See xmod.

`title.size` title of the legend element regarding the text sizes

`title.col` title of the legend element regarding the text colors

`legend.size.show`
logical that determines whether the legend for the text sizes is shown

`legend.col.show`
logical that determines whether the legend for the text colors is shown

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. Parameters are:

scientific Should the labels be formatted scientically? If so, square brackets are used, and the `format` of the numbers is `"g"`. Otherwise, `format="f"`, and `text.separator`, `text.less.than`, and `text.or.more` are used. Also, the numbers are automatically rounded to millions or billions if applicable.

format By default, `"f"`, i.e. the standard notation `xxx.xxx`, is used. If `scientific=TRUE` then `"g"`, which means that numbers are formatted scientically, i.e. `n.dddE+nn` if needed to save space.

digits Number of digits after the decimal point if `format="f"`, and the number of significant digits otherwise.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character string to use to translate "Less than" (which is the default).

text.or.more Character string to use to translate "or more" (which is the default).

... Other arguments passed on to formatC

legend.size.is.portrait
: logical that determines whether the legend element regarding the text sizes is in portrait mode (TRUE) or landscape (FALSE)

legend.col.is.portrait
: logical that determines whether the legend element regarding the text colors is in portrait mode (TRUE) or landscape (FALSE)

legend.hist
: logical that determines whether a histogram is shown regarding the text colors

legend.hist.title
: title for the histogram. By default, one title is used for both the histogram and the normal legend for text colors.

legend.size.z
: index value that determines the position of the legend element regarding the text sizes with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

legend.col.z
: index value that determines the position of the legend element regarding the text colors. (See legend.size.z)

legend.hist.z
: index value that determines the position of the histogram legend element. (See legend.size.z)

id
: name of the data variable that specifies the indices of the text labels. Only used for SVG output (see itmap).

Value

tmap-element

Note

The absolute fontsize (in points) is determined by the (ROOT) viewport, which may depend on the graphics device.

See Also

vignette("tmap-nutshell")

Examples

```
data(World, Europe, metro)

tm_shape(World) +
    tm_text("name", size="AREA")

tm_shape(Europe) +
    tm_polygons() +
    tm_text("iso_a3", size="AREA", col = "grey20", root=4, shadow = TRUE, scale=2,
        size.lowerbound = .1) +
tm_shape(Europe) +
    tm_text("name", size="AREA", root=4, scale=1,
```

```
        ymod=-1 * approx_areas(Europe, unit = "norm")^(1/4))

tm_shape(Europe) +
tm_polygons() +
tm_shape(metro) +
tm_bubbles("pop2010", size.lim = c(0, 15e6), legend.size.is.portrait = TRUE,
   title.size = "European metropolitan areas") +
tm_shape(metro[metro$pop2010>=2e6, ]) +
tm_text("name", auto.placement = TRUE)

tm_shape(World) +
tm_text("name", size="pop_est", col="continent", palette="Dark2",
title.size = "Population", title.col="Continent")
```

`World` *World, Europe, and Netherlands map*

Description

Maps of the world, Europe, and the Netherlands (province and municipality level), class `SpatialPolygonsDataFrame`

Usage

```
data(World)

data(Europe)

data(NLD_prov)

data(NLD_muni)
```

Details

The default projections for these maps are Eckhart IV (World), Lambert azimuthal (Europe), and Rijksdriehoekstelsel (Netherlands). See below. To change the projection, use `set_projection`. Alternatively, the projection can be changed temporarily for plotting purposes by using the projection arugment of `tm_shape` (or `qtm`).

`World` World map. The default projection for this world map is Eckhart IV since area sizes are preserved, which is a very important property for choropleths.

`Europe` Europe map. Lambert azimuthal equal-area projection is used by default for this map. Several countries are transcontinental and are partly located in Asia. From these countries, only Russia and Turkey have been included in this map as part of Europe since they are widely considered as European countries. Other transcontinental countries Azerbaijan, Georgia, and Kazakhstan, are also included in the map, but only as background (so without data). From the other surrounding countries, only Greenland is removed from the map, since it interferes with the prefered map legend position at the lop left.

`NLD_prov` and `NLD_muni`, maps of the Netherlands at province and municipality level of 2013. The used projection is the Rijksdriehoekstelsel projection. **Important:** publication of these maps is only allowed when cited to Statistics Netherlands (CBS) and Kadaster Nederland as source.

Source

http://www.naturalearthdata.com for World and Europe

http://www.happyplanetindex.org for World and Europe

http://www.cbs.nl for NLD_prov and NLD_muni.

References

Statistics Netherlands (2014), The Hague/Heerlen, Netherlands, http://www.cbs.nl.

Kadaster, the Netherlands' Cadastre, Land Registry, and Mapping Agency (2014), Apeldoorn, Netherlands, http://www.kadaster.nl.

Statistics Netherlands (2014), The Hague/Heerlen, Netherlands, http://www.cbs.nl.

---

write_shape *Write shape file*

Description

Write a shape object to an ESRI shape file.

Usage

```
write_shape(shp, file)
```

Arguments

shp	a shape object
file	file name (including directory)

Details

This function is a convenient wrapper of rgdal's writeOGR.

Index