Package 'geometr'

July 14, 2020

Title Generate and Modify Interoperable Geometric Shapes

Version 0.2.5

Description Provides tools that generate and process fully accessible and tidy geometric shapes. The package improves interoperability of spatial and other geometric classes by providing getters and setters that produce identical output from various classes.

URL https://ehrmanns.github.io/geometr/

BugReports https://github.com/EhrmannS/geometr/issues

Depends R (>= 2.10)

Language en-gb

License GPL-3

Encoding UTF-8

LazyData true

Imports checkmate, crayon, deldir, dplyr, grDevices, grid, methods, raster, readr, rgdal, rlang, sf, sp, spatstat, tibble, Rcpp

Suggests testthat, rmarkdown, bookdown, magrittr, covr, knitr

LinkingTo Rcpp

RoxygenNote 7.1.1

SystemRequirements C++11

VignetteBuilder knitr

NeedsCompilation yes

Author Steffen Ehrmann [aut, cre] (<https://orcid.org/0000-0002-2958-0796>), Dan Sunday [cph] (fast point-in-polygon algorithm.)

Maintainer Steffen Ehrmann <steffen.science@funroll-loops.de>

Repository CRAN

Date/Publication 2020-07-14 13:20:05 UTC

R topics documented:

`.getDecimals` *Get the number of decimal places*

Description

Get the number of decimal places

Usage

```
.getDecimals(x)
```

Arguments

`x`	[`numeric(1)`] the number for which to derive decimal places.

`.makeTinyMap` *Make a tiny map*

Description

A tiny map is used via the show method of a geom.

Usage

```
.makeTinyMap(geom = NULL)
```

Arguments

`geom`	[`geom`] the geom from which to create a tiny map.

.rad *Convert degree to radians*

Description

Convert degree to radians

Usage

```
.rad(degree)
```

Arguments

degree	[numeric(1)] a degree value to convert to radians.

.testAnchor *Test anchor for consistency*

Description

Test anchor for consistency

Usage

```
.testAnchor(x, ...)
```

Arguments

x	[data.frame | geom] the object to be tested for consistency.
...	[.] additional arguments.

.testTemplate *Test template for consistency*

Description

Test template for consistency

Usage

```
.testTemplate(x, ...)
```

Arguments

x	[RasterLayer | matrix] the object to be tested for consistency.
...	[.] additional arguments.

.testWindow *Test window for consistency*

Description

Test window for consistency

Usage

```
.testWindow(x, ...)
```

Arguments

x	[data.frame] the object to be tested for consistency.
...	[.] additional arguments.

.updateOrder *Update column order*

Description

Set the order of the table columns to `c("fid","gid",rest)`

Usage

```
.updateOrder(input = NULL)
```

Arguments

`input`	`[data.frame(1)]` a table that contains at least the columns `fid` and `gid`.

Value

A new table where the columns have the correct order.

.updateVertices *Update the vertices*

Description

Set the vertices in a table so that they are valid for a geom.

Usage

```
.updateVertices(input = NULL)
```

Arguments

`input`	`[data.frame(1)]` a table of vertices which should be brought into the correct form.

.updateWindow *Update the window*

Description

Set a window to the minimum/maximum values of input vertices.

Usage

```
.updateWindow(input = NULL, window = NULL)
```

Arguments

input	[data.frame(1)] a table of vertices for which a new window should be derived.
window	[data.frame(1)] the old window.

Value

A new window that has the extent of input.

gc_geom *Transform a spatial object to class* geom

Description

Transform a spatial object to class geom

Usage

```
## S4 method for signature 'Spatial'
gc_geom(input = NULL, ...)

## S4 method for signature 'sf'
gc_geom(input = NULL, group = FALSE, ...)

## S4 method for signature 'ppp'
gc_geom(input = NULL, ...)

## S4 method for signature 'Raster'
gc_geom(input = NULL, ...)
```

Arguments

input	the object to transform to class geom.
...	additional arguments.
group	[logical(1)] should the attributes of multi* features be grouped, i.e. should the unique values per multi* feature be assigned into the groups table (TRUE), or should they be kept as duplicated per-feature attributes (FALSE, default)?

Value

an object of class geom

See Also

Other spatial classes: gc_grob(), gc_ppp(), gc_raster(), gc_sf(), gc_sp()

Examples

```
gc_geom(input = gtPPP)

gc_geom(input = gtSF$polygon)

gc_geom(input = gtRasters$categorical)
```

---

gc_grob *Transform a spatial object to a grob*

Description

Transform a spatial object to a grob

Usage

```
## S4 method for signature 'geom'
gc_grob(input = NULL, theme = gtTheme, ...)
```

Arguments

input	the object to transform to class grob.
theme	[gtTheme(1)] the theme from which to take parameters.
...	instead of providing a modified theme, you can also determine specific graphic parameters (see gpar) separately; see setTheme for details.

Value

Depending on the provided geometry either a pointsGrob,

See Also

Other spatial classes: gc_geom(), gc_ppp(), gc_raster(), gc_sf(), gc_sp()

---

gc_ppp *Transform a spatial object to class* ppp

Description

Transform a spatial object to class ppp

Usage

```
## S4 method for signature 'geom'
gc_ppp(input = NULL)
```

Arguments

input	the object to transform to class ppp.

Value

an object of class ppp

See Also

Other spatial classes: gc_geom(), gc_grob(), gc_raster(), gc_sf(), gc_sp()

Examples

```
gc_ppp(input = gtGeoms$point)
```

---

gc_raster *Transform a spatial object to class* Raster*

Description

Transform a spatial object to class Raster*

Usage

```
## S4 method for signature 'geom'
gc_raster(input = NULL)
```

Arguments

input	the object to transform to class Raster*.

Value

an object of class `Raster*`

See Also

Other spatial classes: `gc_geom()`, `gc_grob()`, `gc_ppp()`, `gc_sf()`, `gc_sp()`

Examples

```
rasGeom <- gc_geom(input = gtRasters$categorical)

gc_raster(input = rasGeom)
```

---

gc_sf *Transform a spatial object to class* `sf`

Description

Transform a spatial object to class `sf`

Usage

```
## S4 method for signature 'geom'
gc_sf(input = NULL)
```

Arguments

`input`	the object to transform to class `sf`.

Value

If `input` is a `geom` and has attributes other than `fid` and `gid`, a "Simple feature collection", otherwise a "Geometry set". Several features of the `geom` are returned as MULTI* feature, when they have `gid` and optionally other attributes in common, otherwise they are returned as a single simple feature.

See Also

Other spatial classes: `gc_geom()`, `gc_grob()`, `gc_ppp()`, `gc_raster()`, `gc_sp()`

Examples

```
gc_sf(input = gtGeoms$point)

gc_sf(input = gtGeoms$line)

gc_sf(input = gtGeoms$polygon)
```

gc_sp *Transform a spatial object to class* Spatial

Description

Transform a spatial object to class Spatial

Usage

```
## S4 method for signature 'geom'
gc_sp(input = NULL)
```

Arguments

input	the object to transform to class Spatial.

Value

an object of class Spatial

See Also

Other spatial classes: gc_geom(), gc_grob(), gc_ppp(), gc_raster(), gc_sf()

Examples

```
gc_sp(input = gtGeoms$point)

gc_sp(input = gtGeoms$line)

gc_sp(input = gtGeoms$polygon)
```

geom-class *Geometry class (S4) and methods*

Description

A geom stores a table of points, a table of feature to which the points are associated and a table of groups, to which features are associated. A geom can be spatial, but is not by default. A geom can either have absolute or relative values, where relative values specify the point position relative to the window slot.

Details

A geom is one of three geometry objects:

- "point", when none of the points are connected to other points,
- "line", where points with the same fid are connected following the sequence of their order, without the line closing in itself and
- "polygon", where points with the same fid are connected following the sequence of their order and the line closes in on itself due to first and last point being the same. Moreover, polygon objects can contain holes.

The data model for storing points follows the spaghetti model. Points are stored as a sequence of x and y values, associated to a feature ID. The feature ID relates coordinates to features and thus common attributes. Points and Lines are implemented straightforward in this model, but polygons, which may contain holes, are a bit trickier. In geometr they are implemented as follows:

1. All points with the same fid make up one polygon, irrespective of it containing holes or not.
2. The outer path/ring of a polygon is composed of all points until a duplicated of its first point occurs. This signals that all following points are part of another path/ring, which must be inside the outer path and which consists of all points until a duplicate of it's first point occurs.
3. This repeats until all points of the feature are processed.

Moreover, a geom does not have the slot *extent*, which characterises the minimum and maximum value of the point coordinates and which is thus derived "on the fly" from the points. Instead it has a *reference window*, which is sort of a second extent that may be bigger (or smaller) than extent and which determines the relative position of the points when plotting.

Slots

type [character(1)]
: the type of feature, either "point", "line", "polygon" or "grid".

point [data.frame(1)]
: the fid (feature ID), x and y coordinates per point and optional arbitrary point attributes.

feature [data.frame(1)]
: fid (feature ID), gid (group ID) and optional arbitrary feature attributes.

group [data.frame(1)]
: gid (group ID) and optional arbitrary group attributes.

window [data.frame(1)]
: the minimum and maximum value in x and y dimension of the reference window in which the geom dwells.

scale [character(1)]
: whether the point coordinates are stored as "absolute" values, or "relative" to window.

crs [character(1)]
: the coordinate reference system in proj4 notation.

history [list(.)]
: a list of steps taken to derive the geom in focus.

geometr *geometr: Generate and Modify Interoperable Geometric Shapes*

Description

The geometr package provides tools that generate and process easily accessible and tidy geometric shapes (of class geom). Moreover, it aims to improve interoperability of spatial and other geometric classes. Spatial classes are typically a collection of geometric shapes (or their vertices) that are accompanied by various metadata (such as attributes and a coordinate reference system). Most spatial classes are thus conceptually quite similar, yet a common standard lacks for accessing features, vertices or the metadata. Geometr fills this gap by providing tools

- that produce an identical output for the same metadata of different classes (via so-called getters) and
- that use an identical input to write to various classes that originally require different input (via so-called setters).

Author(s)

Maintainer, Author: Steffen Ehrmann <steffen.ehrmann@idiv.de>

Copyright holder Dan Sunday fast point-in-polygon algorithm

See Also

- Github project: https://github.com/EhrmannS/geometr
- Report bugs: https://github.com/EhrmannS/geometr/issues

getCRS *Get the coordinate reference system of a spatial object.*

Description

Get the coordinate reference system of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getCRS(x)

## S4 method for signature 'geom'
getCRS(x)

## S4 method for signature 'Spatial'
getCRS(x)
```

```
## S4 method for signature 'sf'
getCRS(x)

## S4 method for signature 'ppp'
getCRS(x)

## S4 method for signature 'Raster'
getCRS(x)
```

Arguments

x	the object from which to extract the coordinate reference system.

Value

The coordinate reference system of x given as proj4string.

See Also

Other getters: getExtent(), getFeatures(), getGroups(), getHistory(), getLayer(), getName(), getPoints(), getRes(), getType(), getWindow()

---

getExtent *Get the extent (bounding box) of a spatial object.*

Description

Get the extent (bounding box) of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getExtent(x)

## S4 method for signature 'geom'
getExtent(x)

## S4 method for signature 'Spatial'
getExtent(x)

## S4 method for signature 'sf'
getExtent(x)

## S4 method for signature 'ppp'
getExtent(x)

## S4 method for signature 'Raster'
getExtent(x)
```

```
## S4 method for signature 'matrix'
getExtent(x)
```

Arguments

x	the object from which to derive the extent.

Value

A table of the lower left and upper right corner of the extent of x.

See Also

Other getters: getCRS(), getFeatures(), getGroups(), getHistory(), getLayer(), getName(), getPoints(), getRes(), getType(), getWindow()

Examples

```
getExtent(gtGeoms$polygon)

getExtent(x = gtSP$SpatialPolygons)

getExtent(x = gtSF$multilinestring)

getExtent(x = gtPPP)

getExtent(x = gtRasters$categorical)

getExtent(x = matrix(0, 3, 5))
```

---

getFeatures *Get the table of feature attributes*

Description

Get tabular information of the attributes of features.

Usage

```
## S4 method for signature 'ANY'
getFeatures(x, ...)

## S4 method for signature 'geom'
getFeatures(x, ...)

## S4 method for signature 'Spatial'
getFeatures(x, ...)
```

```
## S4 method for signature 'sf'
getFeatures(x, ...)

## S4 method for signature 'ppp'
getFeatures(x, ...)

## S4 method for signature 'Raster'
getFeatures(x)

## S4 method for signature 'matrix'
getFeatures(x)
```

Arguments

x	the object from which to derive the attribute table.
...	subset based on logical predicates defined in terms of the columns in x or a vector of booleans. Multiple conditions are combined with &. Only rows where the condition evaluates to TRUE are kept.

Value

A table of the feature attributes of x or an object where the features table has been subsetted.

See Also

Other getters: getCRS(), getExtent(), getGroups(), getHistory(), getLayer(), getName(), getPoints(), getRes(), getType(), getWindow()

Examples

```
getFeatures(x = gtGeoms$polygon)

# get a subset of an sf-object
getFeatures(x = gtSF$multilinestring, a == 1)

# get the values of a RasterLayer
getFeatures(x = gtRasters$continuous)
```

---

getGroups *Get the table of group attributes*

Description

Get tabular information of the attributes of groups of features.

Usage

```
## S4 method for signature 'ANY'
getGroups(x, ...)

## S4 method for signature 'geom'
getGroups(x, ...)

## S4 method for signature 'Raster'
getGroups(x)

## S4 method for signature 'matrix'
getGroups(x)
```

Arguments

x	the object from which to derive the attribute table.
...	subset based on logical predicates defined in terms of the columns in x or a vector of booleans. Multiple conditions are combined with &. Only rows where the condition evaluates to TRUE are kept.

Value

A table of the group attributes of x or an object where the groups table has been subsetted.

See Also

Other getters: getCRS(), getExtent(), getFeatures(), getHistory(), getLayer(), getName(), getPoints(), getRes(), getType(), getWindow()

Examples

```
getGroups(x = gtGeoms$polygon)

# for raster objects, groups are pixels with the same value and their
# attributes are in the raster attribute table (RAT)
getGroups(x = gtRasters$categorical)
```

---

getHistory *Get the history of a spatial object.*

Description

Get the history of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getHistory(x)

## S4 method for signature 'geom'
getHistory(x)

## S4 method for signature 'Raster'
getHistory(x)
```

Arguments

x	the object from which to derive the history.

Value

A list of the events that lead to x.

See Also

Other getters: getCRS(), getExtent(), getFeatures(), getGroups(), getLayer(), getName(), getPoints(), getRes(), getType(), getWindow()

---

getLayer *Get a specific layer of a spatial object.*

Description

Get a specific layer of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getLayer(x)

## S4 method for signature 'geom'
getLayer(x, layer = NULL)

## S4 method for signature 'Spatial'
getLayer(x, layer = NULL)

## S4 method for signature 'sf'
getLayer(x, layer = NULL)

## S4 method for signature 'Raster'
getLayer(x, layer = NULL)

## S4 method for signature 'matrix'
getLayer(x, layer = NULL)
```

Arguments

x	the object from which to get the layer.
layer	[character(.) | integerish(.)] the layer(s) to get.

Value

A list of the requested layers.

See Also

Other getters: getCRS(), getExtent(), getFeatures(), getGroups(), getHistory(), getName(), getPoints(), getRes(), getType(), getWindow()

---

getName *Get the name(s) of a spatial object.*

Description

Get the name(s) of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getName(x)

## S4 method for signature 'geom'
getName(x)

## S4 method for signature 'sf'
getName(x)

## S4 method for signature 'Raster'
getName(x)
```

Arguments

x	the object from which to get the name.

Value

A vector of the requested names.

See Also

Other getters: getCRS(), getExtent(), getFeatures(), getGroups(), getHistory(), getLayer(), getPoints(), getRes(), getType(), getWindow()

`getPoints` *Get the table of point attributes*

Description

Get tabular information of the attributes of points (incl. coordinates).

Usage

```
## S4 method for signature 'ANY'
getPoints(x, ...)

## S4 method for signature 'geom'
getPoints(x, ...)

## S4 method for signature 'Spatial'
getPoints(x)

## S4 method for signature 'sf'
getPoints(x)

## S4 method for signature 'ppp'
getPoints(x)

## S4 method for signature 'Raster'
getPoints(x)

## S4 method for signature 'matrix'
getPoints(x)
```

Arguments

`x`	the object from which to derive the attribute table.
`...`	subset based on logical predicates defined in terms of the columns in `x` or a vector of booleans. Multiple conditions are combined with `&`. Only rows where the condition evaluates to TRUE are kept.

Value

A table of the point attributes of `x` or an object where the point table has been subsetted.

See Also

Other getters: `getCRS()`, `getExtent()`, `getFeatures()`, `getGroups()`, `getHistory()`, `getLayer()`, `getName()`, `getRes()`, `getType()`, `getWindow()`

Examples

```
getPoints(x = gtGeoms$polygon)

getPoints(x = gtGeoms$point)

# for a raster object, the @point slot is extracted from its' compact storage
gtGeoms$grid$continuous@point
getPoints(x = gtGeoms$grid$continuous)
```

---

getRes *Get the spatial resolution of a spatial object.*

Description

Get the spatial resolution of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getRes(x)

## S4 method for signature 'geom'
getRes(x, precision = getOption("digits"))

## S4 method for signature 'Raster'
getRes(x, precision = getOption("digits"))

## S4 method for signature 'matrix'
getRes(x)
```

Arguments

x	the object from which to derive the resolution.
precision	the number of digits to which to round the values.

Value

The resolution of x in x and y dimension.

See Also

Other getters: getCRS(), getExtent(), getFeatures(), getGroups(), getHistory(), getLayer(), getName(), getPoints(), getType(), getWindow()

getType *Get the type of a spatial object.*

Description

Get the type of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getType(x)

## S4 method for signature 'geom'
getType(x)

## S4 method for signature 'Spatial'
getType(x)

## S4 method for signature 'sf'
getType(x)

## S4 method for signature 'ppp'
getType(x)

## S4 method for signature 'Raster'
getType(x)

## S4 method for signature 'matrix'
getType(x)
```

Arguments

x	the object for which to determine the type.

Value

A vector of two values giving the general type (vector/raster) and the specific type/class of x.

See Also

Other getters: getCRS(), getExtent(), getFeatures(), getGroups(), getHistory(), getLayer(), getName(), getPoints(), getRes(), getWindow()

Examples

```
getType(x = gtGeoms$polygon)

getType(x = gtSP$SpatialPolygons)
```

```
getType(x = gtSF$multiline)

getType(x = gtRasters$categorical)
```

`getWindow` *Get the reference window of a spatial object.*

Description

Get the reference window of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getWindow(x)

## S4 method for signature 'geom'
getWindow(x)

## S4 method for signature 'Spatial'
getWindow(x)

## S4 method for signature 'sf'
getWindow(x)

## S4 method for signature 'ppp'
getWindow(x)

## S4 method for signature 'Raster'
getWindow(x)

## S4 method for signature 'matrix'
getWindow(x)
```

Arguments

`x`	the object from which to derive the reference window.

Value

A table of the corners of the reference window of `x`.

See Also

Other getters: `getCRS()`, `getExtent()`, `getFeatures()`, `getGroups()`, `getHistory()`, `getLayer()`, `getName()`, `getPoints()`, `getRes()`, `getType()`

Examples

```
getWindow(x = gtGeoms$line)

getWindow(x = gtSP$SpatialLines)

getWindow(x = gtSF$multilinestring)

getWindow(x = gtPPP)

getWindow(x = gtRasters$categorical)

getWindow(x = matrix(0, 3, 5))
```

---

gs_line *Create a line* geom

Description

Create a line geometry (of class geom) either by specifying anchor values or by sketching it.

Usage

```
gs_line(
  anchor = NULL,
  window = NULL,
  features = 1,
  vertices = NULL,
  sketch = NULL,
  ...
)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the geom from. It must include column names x, y and optionally a custom fid.
window	[data.frame(1)] in case the reference window deviates from the bounding box of anchor (minimum and maximum values), specify this here.
features	[integerish(1)] number of lines to create.
vertices	[integerish(.)] number of vertices per line; will be recycled if it does not have as many elements as specified in features.
sketch	[raster(1)] raster object that serves as template to sketch polygons.

... [various]
graphical parameters to gt_locate, in case points are sketched; see gpar

Details

The arguments anchor and sketch indicate how the line is created:

- if anchor is set, the line is created parametrically from the given objects' points,
- if an object is set in sketch, this is used to create the geom interactively, by clicking into the plot.

Value

An invisible geom.

See Also

Other geometry shapes: gs_point(), gs_polygon(), gs_random()

Examples

```
# 1. create a line programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))

# if no window is set, the bounding box will be set as window
(aGeom <- gs_line(anchor = coords))

# the vertices are plottet relative to the window
library(magrittr)
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
gs_line(anchor = coords, window = window) %>%
  visualise(linecol = "green")

# when a geom is used in 'anchor', its properties are passed on
aGeom <- setWindow(x = aGeom, to = window)
gs_line(anchor = aGeom) %>%
  visualise(linecol = "deeppink")

# 2. sketch a line by clicking into a template
gs_line(sketch = gtRasters$continuous, vertices = 4) %>%
  visualise(linecol = "orange", linewidth = 5, new = FALSE)
```

gs_point *Create a point* geom

Description

Create a point geometry (of class geom) either by specifying anchor values or by sketching it.

Usage

```
gs_point(anchor = NULL, window = NULL, vertices = 1, sketch = NULL, ...)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the geom from. It must include column names x, y and optionally a custom fid.
window	[data.frame(1)] in case the reference window deviates from the bounding box of anchor (minimum and maximum values), specify this here.
vertices	[integer(1)] number of vertices.
sketch	[raster(1)] raster object that serves as template to sketch polygons.
...	[various] graphical parameters to gt_locate, in case points are sketched; see gpar

Details

The arguments anchor and sketch indicate how the line is created:

- if anchor is set, the line is created parametrically from the given objects' points,
- if an object is set in sketch, this is used to create the geom interactively, by clicking into the plot.

Value

An invisible geom.

See Also

Other geometry shapes: gs_line(), gs_polygon(), gs_random()

Examples

```
# 1. create points programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))

# if no window is set, the bounding box will be set as window
(aGeom <- gs_point(anchor = coords))

# the vertices are plottet relative to the window
library(magrittr)
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
gs_point(anchor = coords, window = window) %>%
  visualise(linecol = "green")

# when a geom is used in 'anchor', its properties are passed on
aGeom <- setWindow(x = aGeom, to = window)
gs_point(anchor = aGeom) %>%
  visualise(geom = .)

# 2. sketch two points by clicking into a template
gs_point(sketch = gtRasters$continuous, vertices = 2) %>%
  visualise(geom = ., linecol = "green", pointsymbol = 5, new = FALSE)
```

---

gs_polygon *Create a polygon* geom

Description

Create any (regular) polygon geometry (of class geom) either by specifying anchor values or by sketching it.

Usage

```
gs_polygon(
  anchor = NULL,
  window = NULL,
  features = 1,
  vertices = NULL,
  sketch = NULL,
  regular = FALSE,
  ...
)

gs_triangle(anchor = NULL, window = NULL, sketch = NULL, features = 1, ...)

gs_square(anchor = NULL, window = NULL, sketch = NULL, features = 1, ...)
```

```
gs_rectangle(anchor = NULL, window = NULL, sketch = NULL, features = 1, ...)

gs_hexagon(anchor = NULL, window = NULL, sketch = NULL, features = 1, ...)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the geom from. It must include column names x, y and optionally a custom fid.
window	[data.frame(1)] in case the reference window deviates from the bounding box of anchor (minimum and maximum values), specify this here.
features	[integerish(1)] number of polygons to create.
vertices	[integerish(.)] number of vertices per polygon; will be recycled if it does not have as many elements as specified in features.
sketch	[raster(1)] raster object that serves as template to sketch polygons.
regular	[logical(1)] should the polygon be regular, i.e. point symmetric (TRUE) or should the vertices be selected as provided by anchor (FALSE, default)?
...	[various] graphical parameters to gt_locate, in case points are sketched; see gpar

Details

The arguments anchor and sketch indicate how the line is created:

- if anchor is set, the line is created parametrically from the given objects' points,
- if an object is set in sketch, this is used to create the geom interactively, by clicking into the plot.

The argument regular determines how the vertices provided in anchor or via sketch are transformed into a polygon:

- if regular = FALSE the resulting polygon is created from all vertices in anchor,
- if regular = TRUE, only the first two vertices are considered, as center and indicating the distance to the (outer) radius.

Value

An invisible geom.

Functions

- gs_triangle: wrapper of gs_polygon where vertices = 3 and regular = TRUE.
- gs_square: wrapper of gs_polygon where vertices = 4 and regular = TRUE.
- gs_rectangle: wrapper of gs_polygon where vertices = 2, regular = FALSE and the two complementing corners are derived from the two given opposing corners.
- gs_hexagon: wrapper of gs_polygon where vertices = 6 and regular = TRUE.

See Also

Other geometry shapes: gs_line(), gs_point(), gs_random()

Examples

```
# 1. create a polygon programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))

# if no window is set, the bounding box will be set as window
(aGeom <- gs_polygon(anchor = coords))

# the vertices are plottet relative to the window
library(magrittr)
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
gs_polygon(anchor = coords, vertices = 6, window = window,
           regular = TRUE) %>%
  visualise(linecol = "green")

# when a geom is used in 'anchor', its properties are passed on
aGeom <- setWindow(x = aGeom, to = window)
gs_polygon(anchor = aGeom) %>%
  visualise(geom = ., fillcol = "deeppink")
gs_rectangle(anchor = aGeom) %>%
  visualise(geom = ., new = FALSE)

# 2. sketch a hexagon by clicking into a template
gs_hexagon(sketch = gtRasters$continuous) %>%
  visualise(geom = ., linecol = "deeppink", linetype = 2, new = FALSE)
```

---

gs_random *Create a* geom *randomly*

Description

This function creates a random geometry

Usage

```
gs_random(type = "point", window = NULL, vertices = NULL, ...)
```

Arguments

type	[character(1)] Either one of the three main feature types "point", "line" or "polygon", or more specifically one of their subtypes, e.g. "hexagon".
window	[data.frame(1)] in case the reference window deviates from the bounding box [0, 1] (minimum and maximum values), specify this here.
vertices	[integerish(1)] the number of vertices the geometry should have; only meaningful if type does not indicate the number of vertices already. If left at NULL the minimum number of vertices for the geom type, i.e. 1 for point, 2 for line and 3 for polygon.
...	[various] additional arguments.

See Also

Other geometry shapes: gs_line(), gs_point(), gs_polygon()

Examples

```
input <- matrix(nrow = 100, ncol = 100, data = 0)

# create a random polygon with five vertices
set.seed(1)
someGeom <- gs_random(type = "polygon", vertices = 5)
visualise(geom = someGeom)

# in case template is given, this serves as source for the window extent
library(magrittr)
gs_random(template = input) %>%
  visualise(geom = ., new = FALSE, linecol = "red")
```

---

gs_tiles *Create a regular tiling* geom

Description

Create a regular tiling polygon geometry (of class geom) for the extent of an anchor value.

Usage

```
gs_tiles(
  anchor = NULL,
  width = NULL,
  pattern = "squared",
  centroids = FALSE,
  ...
)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the tiling geom from. It must include column names x, y and optionally a custom fid.
width	[numeric(1)] the width (which does not correspond to the height in case of pattern = "hexagonal") of a tile.
pattern	[character(1)] pattern of the tiling. Possible options are "squared" (default) or "hexagonal".
centroids	[logical(1)] should the centroids of the tiling be returned (TRUE) or should the tiling be returned (FALSE, default)?
...	[various] additional arguments; see Details.

Details

When deriving a regular tiling for a prescribed window, there is only a limited set of legal combinations of cells in x and y dimension. For instance, a window of 100 by 100 can't comprise 10 by 5 squares of side-length 10, because then the y-dimension wouldn't be fully covered. The same is true for hexagonal and triangular tilings. As all tilings are regular, the measurement of one dimension is sufficient to specify the dimensions of tiles, which is width.

Possible additional arguments are:

- verbose = TRUE/FALSE
- graphical parameters to gt_locate, in case points are sketched; see gpar

Value

An invisible geom.

See Also

Other tilings: gs_voronoi()

Examples

```
# create a squared tiling
library(magrittr)
aWindow <- data.frame(x = c(-180, 180),
                      y = c(-60, 80))
gs_tiles(anchor = aWindow, width = 10) %>%
  visualise(`10° world tiles` = .)

# create a hexagonal tiling on top of a geom
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_polygon(anchor = coords, window = window)
visualise(`honeycomb background` = aGeom)
gs_tiles(anchor = aGeom, width = 8, pattern = "hexagonal") %>%
  visualise(., linecol = "deeppink", new = FALSE)
```

`gs_voronoi` *Create a voronoi tiling* `geom`

Description

Create a voronoi tiling `geom`

Usage

```
gs_voronoi(anchor = NULL, window = NULL, features = 3, sketch = NULL, ...)
```

Arguments

`anchor`	[`geom(1)`|`data.frame(1)`] Object to derive the `geom` from. It must include column names `x`, `y` and optionally a custom `fid`.
`window`	[`data.frame(1)`] in case the reference window deviates from the bounding box of `anchor` (minimum and maximum values), specify this here.
`features`	[`integerish(1)`] number of tiles to sketch.
`sketch`	[`RasterLayer(1)`	`matrix(1)`] Gridded object that serves as template to sketch the tiling.
`...`	[various] graphical parameters to `gt_locate`, in case the tiling is sketched; see `gpar`.

Value

An invisible `geom`.

See Also

Other tilings: gs_tiles()

Examples

```
# 1. create voronoi polygons programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_point(anchor = coords, window = window)
visualise(voronoi = aGeom)

tiles <- gs_voronoi(anchor = aGeom)
visualise(tiles, new = FALSE)

# 2. sketch a voronoi polygon by clicking into a template
gs_voronoi(sketch = gtRasters$continuous) %>%
  visualise(tiles = ., new = FALSE)
```

---

gtGeoms *Example* geom *objects*

Description

A set of five geometries.

Usage

```
gtGeoms
```

Format

The list contains five objects of class geom, a point, line and polygon object and two grid geoms, one with categorical data and one with continuous data. They are mostly used in the example and test-sections of this package.

gtPPP *Example* ppp *object*

Description

A single ppp object

Usage

```
gtPPP
```

Format

The object contains an object of class ppp. It consists of 15 points without an attribute table. It is mostly used in the example and test-sections of this package.

gtRasters *Example* RasterStack *object*

Description

A set of two conceptually different types of raster.

Usage

```
gtRasters
```

Format

The object of class RasterStack has no projection and is a RasterStack object of 56 by 60 cells. The first raster represents land-use classes and the second raster contains a continuous scale of vegetation cover.

gtSF *Example* sf *objects*

Description

A set of six sp objects

Usage

```
gtSF
```

Format

The list contains six objects of class sf, a POINT, a MULTIPOINT, a LINESTRING, a MULTILINESTRING, a POLYGON, and a MULTIPOLYGON object. They are mostly used in the example and test-sections of this package.

gtSP *Example* Spatial *objects*

Description

A set of four sp objects

Usage

```
gtSP
```

Format

The list contains four objects of class Spatial, a SpatialPoints, a SpatialMultiPoints, a, SpatialLines and a SpatialPolygons object. They are mostly used in the example and test-sections of this package.

gtTheme *Default visualising theme*

Description

Default visualising theme

Usage

```
gtTheme
```

Format

An object of class `gtTheme` of length 1.

gtTheme-class *Theme class (S4) and methods*

Description

An `gtTheme` stores a theme to `visualise` vector and raster objects. It is recommended to use `setTheme` to modify a `gtTheme`, because it carries out all the checks and makes sure that names of the parameters are properly matched.

Slots

`title` [named `list(3)`]
: properties of the title.

`box` [named `list(4)`]
: properties of the bounding box.

`xAxis` [named `list(5)`]
: properties of the x-axis, its labels and ticks.

`yAxis` [named `list(5)`]
: properties of the y-axis, its labels and ticks.

`grid` [named `list(5)`]
: properties of the major and minor grid.

`legend` [named `list(10)`]
: properties of the legend, its title, labels, ticks and bounding box.

`vector` [named `list(7)`]
: properties of a vector object.

`raster` [named `list(2)`]
: properties of a raster object.

gt_locate *Locate (and identify) clicks*

Description

Click into a plot to get the location or identify values

Usage

```
gt_locate(
  samples = 1,
  panel = NULL,
  identify = FALSE,
  snap = FALSE,
  raw = FALSE,
  show = TRUE,
  ...
)
```

Arguments

samples	[integerish(1)] the number of clicks.
panel	[character(1)] the panel in which to locate (i.e. the title shown over the plot).
identify	[logical(1)] get the raster value or geom ID at the sampled location (TRUE) or merely the location (FALSE, default).
snap	[logical(1)] should the returned value(s) be set to the nearest raster cell's center (TRUE) or should they remain the selected, "real" value (FALSE, default)?
raw	[logical(1)] should the complete statistics about the clicks be returned (TRUE), or should only the basic output be returned (FALSE, default)?
show	[logical(1)] should information be plotted (TRUE), or should they merely be returned to the console (FALSE, default)?
...	[various] graphical parameters of the objects that are created when show = TRUE.

Value

A tibble of the selected locations and, if identify = TRUE, the respective values. If show = TRUE the values are also shown in the plot.

Examples

```
# locate coordinates with geoms
visualise(geom = gtGeoms$polygon)
gt_locate(samples = 2)

# locate or identify values with rasters
visualise(raster = gtRasters$continuous)
gt_locate(identify = TRUE, snap = TRUE)

# with several panels, specify a target
visualise(gtRasters)
gt_locate(samples = 4, panel = "categorical", snap = TRUE, identify = TRUE)
```

---

gt_reflect *Reflect* geom*s*

Description

Reflect geoms across a reflection axis.

Usage

```
gt_reflect(geom = NULL, angle = NULL, fid = NULL, update = TRUE)
```

Arguments

geom	[geom(.)] the object to reflect.
angle	[numeric(1)] the counter-clockwise angle by which the reflection axis shall be rotated (can be negative to rotate clockwise).
fid	[integerish(.)] if only a subset of features shall be rotated, specify that here.
update	[logical(1)] whether or not to update the window slot after rotation.

Details

The reflection axis is a straight line that goes through the plot origin with the given angle, where positive angles open towards the positive y-axis and negative angles open up towards the negative y-axis.

Value

Reflected geom.

See Also

Other geometry tools: gt_rotate(), gt_scale(), gt_sketch(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
# the original object
coords <- data.frame(x = c(30, 60, 60, 40, 10, 40, 20),
                     y = c(40, 40, 60, 70, 10, 20, 40),
                     fid = c(1, 1, 1, 1, 2, 2, 2))
window <- data.frame(x = c(-80, 80),
                     y = c(-80, 80))
aGeom <- gs_polygon(anchor = coords, window = window)

# reflect several geoms
visualise(geom = gt_reflect(geom = aGeom, angle = 30))

# reflect a single geom
visualise(geom = gt_reflect(geom = aGeom, angle = -45, fid = 1))
```

---

gt_rotate *Rotate* geoms

Description

Rotate geoms by a certain angle about a center

Usage

```
gt_rotate(
  geom = NULL,
  angle = NULL,
  about = c(0, 0),
  fid = NULL,
  update = TRUE
)
```

Arguments

geom	[geom(.)] the object to rotate.
angle	[numeric(1)] the counter-clockwise angle by which geom shall be rotated (can be negative to rotate clockwise).
about	[numeric(2)] the point about which geom shall be rotated.

fid	[integerish(.)] if only a subset of features shall be rotated, specify that here.
update	[logical(1)] whether or not to update the window slot after rotation.

Value

Rotated geom.

See Also

Other geometry tools: gt_reflect(), gt_scale(), gt_sketch(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
# the original object
coords <- data.frame(x = c(30, 60, 60, 40, 10, 40, 20),
                     y = c(40, 40, 60, 70, 10, 20, 40),
                     fid = c(1, 1, 1, 1, 2, 2, 2))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_polygon(anchor = coords, window = window)
visualise(geom = aGeom)

# rotate all geoms
rotatedGeom <- gt_rotate(geom = aGeom, angle = 90, about = c(40, 40))
visualise(geom = rotatedGeom)

# rotate a single geom
rotTri <- gt_rotate(geom = aGeom, angle = -180, about = c(30, 40), fid = 2)
visualise(geom = rotTri)

# rotate different geoms about different centers by different angles
rotateMore <- gt_rotate(geom = aGeom,
                        angle = list(90, -180),
                        about = list(c(40, 40), c(30, 40)))
visualise(geom = rotateMore)
```

---

gt_scale *Scale* geoms

Description

Scale the vertex values of geoms to a values range or so that they are either relative to the @window slot, or absolute values.

Usage

```
gt_scale(geom, range = NULL, to = "relative")
```

Arguments

geom	[geom(.)] the object to be scaled.
range	[list(2)] vector of length two for both of the x and y dimension to which the values should be scaled.
to	[character(1)] the scale to which the coordinates should be transformed; possible are "relative" and "absolute"; ignored in case range != NULL.

Value

Scaled geom.

See Also

Other geometry tools: gt_reflect(), gt_rotate(), gt_sketch(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
coords <- data.frame(x = c(40, 70, 70, 50, 40),
                     y = c(40, 40, 60, 70, 40),
                     fid = 1)
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_polygon(anchor = coords, window = window)

# change to relative scale and back to absolute
(relCoords <- gt_scale(geom = aGeom, to = "relative"))
gt_scale(geom = relCoords, to = "absolute")

# scale to another range
gt_scale(geom = aGeom, range = list(x = c(0, 100), y = c(10, 90)))
```

---

gt_sketch *Sketch* geoms

Description

Sketch geoms by clicking into a plot.

Usage

```
gt_sketch(
  template = NULL,
  shape = NULL,
  features = 1,
  vertices = NULL,
  regular = FALSE,
  fixed = FALSE,
  show = FALSE,
  ...
)
```

Arguments

template	[RasterLayer(1) | matrix(1)] Gridded object that serves as template to sketch the geometry.
shape	[character(1)] a geometry shape that should be sketched, possible are the geom types "point", "line" and "polygon" and special cases thereof (recently implemented are "triangle", "square", "hexagon") and "random".
features	[integerish(1)] number of geometries to create.
vertices	[integerish(.)] number of vertices per geometry; will be recycled if it does not have as many elements as specified in features.
regular	[logical(1)] if a polygon is sketched, should it be regular, i.e. point symmetric (TRUE) with the number of corners defined by vertices or should the vertices be selected according to the click locations, resulting in a non-regular polygon (FALSE, default)?
fixed	[logical(1)] if a regular polygon is sketched, should it be aligned vertically (TRUE, default), or should it be aligned according to the second click (FALSE); only relevant if regular = TRUE.
show	[logical(1)] should plot information be shown in the plot (TRUE), or should the geom merely be returned in the console (FALSE, default)
...	[various] additional arguments to gt_locate.

Value

An invisible geom.

See Also

Other geometry tools: gt_reflect(), gt_rotate(), gt_scale(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
# sketch a point geometry
gt_sketch(template = gtRasters$categorical, shape = "point") %>%
  visualise(points = ., linecol = "green", pointsymbol = 5, new = FALSE)

# sketch a line geometry
gt_sketch(template = gtRasters$categorical, vertices = 4, shape = "line") %>%
  visualise(points = ., linecol = "orange", linewidth = 5, new = FALSE)

# sketch a polygon geometry
gt_sketch(template = gtRasters$continuous, shape = "hexagon") %>%
  visualise(geom = ., linecol = "deeppink", linetype = 2, new = FALSE)
```

---

gt_skew *Skew* geoms

Description

Skew geoms by a shear factor in x and y-dimension.

Usage

```
gt_skew(geom, x = NULL, y = NULL, fid = NULL, update = TRUE)
```

Arguments

geom	[geom(.)] the object to skew.
x	[numeric(1)] the shear factor in x-dimension.
y	[numeric(1)] the shear factor in y-dimension.
fid	[integerish(.)] if only a subset of features shall be skewed, specify that here.
update	[logical(1)] whether or not to update the window slot after skewing.

Value

Skewed geom.

See Also

Other geometry tools: gt_reflect(), gt_rotate(), gt_scale(), gt_sketch(), gt_stretch(), gt_translate()

Examples

```
# the original object
coords <- data.frame(x = c(30, 60, 60, 40, 10, 40, 20),
                     y = c(40, 40, 60, 70, 10, 20, 40),
                     fid = c(1, 1, 1, 1, 2, 2, 2))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_polygon(anchor = coords, window = window)

# skew several geoms
visualise(geom = gt_skew(geom = aGeom, x = list(0.5), y = list(0, 0.2)))

# skew single geom
visualise(geom = gt_skew(geom = aGeom, x = 0.5, fid = 1))
```

---

gt_stretch *Stretch* geoms

Description

Stretch geoms by a scale factor in x and y-dimension.

Usage

```
gt_stretch(geom, x = NULL, y = NULL, fid = NULL, update = TRUE)
```

Arguments

geom	[geom(.)] the object to stretch.
x	[numeric(1)] the scale factor in x-dimension.
y	[numeric(1)] the scale factor in y-dimension.
fid	[integerish(.)] if only a subset of features shall be stretched, specify that here.
update	[logical(1)] whether or not to update the window slot after stretching.

Value

Stretched geom.

See Also

Other geometry tools: gt_reflect(), gt_rotate(), gt_scale(), gt_sketch(), gt_skew(), gt_translate()

Examples

```
# the original object
coords <- data.frame(x = c(30, 60, 60, 40, 10, 40, 20),
                     y = c(40, 40, 60, 70, 10, 20, 40),
                     fid = c(1, 1, 1, 1, 2, 2, 2))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_polygon(anchor = coords, window = window)

# stretch several geoms
visualise(geom = gt_stretch(geom = aGeom, x = list(0.5), y = list(1, 0.2)))

# stretch single geom
visualise(geom = gt_stretch(geom = aGeom, x = 0.5, fid = 1))
```

---

gt_translate *Translate* geoms

Description

Translate geoms by adding a constant in x and y-dimension.

Usage

```
gt_translate(geom = NULL, x = NULL, y = NULL, fid = NULL, update = TRUE)
```

Arguments

geom	[geom(.)] the object to translate.
x	[numeric(1)] the translation constant (offset) in x-dimension.
y	[numeric(1)] the translation constant (offset) in y-dimension.
fid	[integerish(.)] if only a subset of features shall be rotated, specify that here.
update	[logical(1)] whether or not to update the window slot after rotation.

Value

Mathematically translated geom.

See Also

Other geometry tools: gt_reflect(), gt_rotate(), gt_scale(), gt_sketch(), gt_skew(), gt_stretch()

Examples

```
# the original object
coords <- data.frame(x = c(30, 60, 60, 40, 10, 40, 20),
                     y = c(40, 40, 60, 70, 10, 20, 40),
                     fid = c(1, 1, 1, 1, 2, 2, 2))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_polygon(anchor = coords, window = window)

# translate several geoms
visualise(geom = gt_translate(geom = aGeom, x = 5, y = list(-10, 5)))

# translate a single geom
visualise(geom = gt_translate(geom = aGeom, x = 5, fid = 1))
```

---

makeLayout *Make the layout of a plot*

Description

Make the layout of a plot

Usage

```
makeLayout(x = NULL, theme = gtTheme)
```

Arguments

x	[list(.)] the object, output from makeObject, from which to make the plot.
theme	[gtTheme(1)] the theme from which to take graphical parameters.

makeObject *Make the object to a plot*

Description

Make the object to a plot

Usage

```
makeObject(x, window = NULL, image = FALSE, theme = gtTheme, ...)
```

Arguments

x	[list(1)] named list of the object from which to make the plot.
window	[data.frame(1)] two opposing corners of a rectangle to which the plot is limited.
image	[logical(1)] whether or not x is an image
theme	[gtTheme(1)] the theme from which to take graphical parameters.
...	instead of providing a gtTheme, you can also determine specific graphic parameters (see gpar) separately; see setTheme for details.

setCRS *Set (or transform) the coordinate reference system of a spatial object.*

Description

Set (or transform) the coordinate reference system of a spatial object.

Usage

```
## S4 method for signature 'ANY'
setCRS(x)

## S4 method for signature 'geom'
setCRS(x, crs = NULL)

## S4 method for signature 'Spatial'
setCRS(x, crs = NULL)

## S4 method for signature 'sf'
setCRS(x, crs = NULL)

## S4 method for signature 'Raster'
setCRS(x, crs = NULL)
```

Arguments

x	the object for which to set the coordinate reference system.
crs	[character(1)] the coordinate reference system to set for this object.

Details

In case an object does not yet have a coordinate reference system assigned, this function simply assigns it. In case the object has already a valid crs, a transformation to the new crs will be carried out. The transformation is computed for all classes with the standard defined in the rgdal package.

Value

The object x with an assigned or transformed coordinate reference system.

See Also

Other setters: setFeatures(), setGroups(), setHistory(), setWindow()

---

setFeatures *Set a table of feature attributes*

Description

Set a table of feature attributes

Usage

```
## S4 method for signature 'ANY'
setFeatures(x)

## S4 method for signature 'geom'
setFeatures(x, table = NULL)

## S4 method for signature 'Spatial'
setFeatures(x, table = NULL)

## S4 method for signature 'sf'
setFeatures(x, table = NULL)

## S4 method for signature 'sfc'
setFeatures(x, table = NULL)

## S4 method for signature 'ppp'
setFeatures(x, table = NULL)
```

Arguments

x	the object to which to assign a new attribute table.
table	[data.frame(.)] the attribute table.

Value

The object x with an updated feature attribute table.

See Also

Other setters: setCRS(), setGroups(), setHistory(), setWindow()

---

setGroups *Set a table of group attributes*

Description

Set a table of group attributes

Usage

```
## S4 method for signature 'ANY'
setGroups(x)

## S4 method for signature 'geom'
setGroups(x, table = NULL)

## S4 method for signature 'RasterLayer'
setGroups(x, table = NULL)
```

Arguments

x	the object to which to assign a new attribute table.
table	[data.frame(.)] the new attribute table.

Value

The object x with an updated group attribute table.

See Also

Other setters: setCRS(), setFeatures(), setHistory(), setWindow()

setHistory *Set additional entries to the history of an object*

Description

Set additional entries to the history of an object

Usage

```
## S4 method for signature 'ANY'
setHistory(x)

## S4 method for signature 'geom'
setHistory(x, history = NULL)

## S4 method for signature 'RasterLayer'
setHistory(x, history = NULL)
```

Arguments

x	the object for which to set the coordinate reference system.
history	[list(1)] the history to set for this object.

Details

Both, objects of class geom and Raster* have the slot @history, which contains the provenance of that object. With setHistory, that provenance can be updated, based on the modification the object has been exposed to. This happens automatically for all geometry operations that come with geometr.

Value

The object x where the history slot has been updated.

See Also

Other setters: setCRS(), setFeatures(), setGroups(), setWindow()

setTheme *Create a new theme*

Description

Assign parameters in a gtTheme to create a new theme.

Usage

```
setTheme(
  from = NULL,
  title = NULL,
  box = NULL,
  xAxis = NULL,
  yAxis = NULL,
  grid = NULL,
  legend = NULL,
  scale = NULL,
  vector = NULL,
  raster = NULL
)
```

Arguments

from	[gtTheme] an gtTheme object.
title	[named list(.)] plot = TRUE/FALSE, fontsize and colour of the title.
box	[named list(.)] plot = TRUE/FALSE, linewidth, linetype and linecol of the bounding box (not supported recently).
xAxis	[named list(.)] plot = TRUE/FALSE, number of bins and margin of the x-axis, label [named list(.)] plot = TRUE/FALSE, title, fontsize, colour and rotation of the x-axis label, ticks [named list(.)] plot = TRUE/FALSE, fontsize, colour and number of digits to which to round the x-axis ticks.
yAxis	[named list(.)] plot = TRUE/FALSE, number of bins and margin of the y-axis, label [named list(.)]

plot = TRUE/FALSE, title, fontsize, colour and rotation of the y-axis label,

ticks [named list(.)]
plot = TRUE/FALSE, fontsize, colour and number of digits to which to round the y-axis ticks.

grid [named list(.)]
plot = TRUE/FALSE, colour, linetype and linewidth of the major and minor grid and whether or not to plot the minor = TRUE/FALSE grid.

legend [named list(.)]
plot = TRUE/FALSE, number of bins, ascending = TRUE/FALSE order of values and the sizeRatio of plot and legend,

label [named list(.)]
plot = TRUE/FALSE, fontsize and colour of the legend labels,

box [named list(.)]
plot = TRUE/FALSE, linetype, linewidth and colour of the legend box.

scale [named list(.)]
param = 'someParameter' and to = 'someAttribute' to which to scale 'someParameter' to. Whether or not to use the values' identity, the value range that shall be represented by the scale and the number of bins.

vector [named list(.)]
linecol, fillcol, missingcol, linetype, linewidth, pointsize and pointsymbol of a vector object.

raster [named list(.)]
fillcol of a raster object.

Examples

```
input <- gtRasters$continuous
(myTheme <- setTheme(title = list(plot = FALSE)))

visualise(input, theme = myTheme)
```

---

setWindow *Set the reference window of a spatial object.*

Description

Set the reference window of a spatial object.

Usage

```
## S4 method for signature 'ANY'
setWindow(x)

## S4 method for signature 'geom'
setWindow(x, to = NULL)

## S4 method for signature 'ppp'
setWindow(x, to = NULL)
```

Arguments

x	the object for which to set a new reference window.
to	any suitable data-structure that contains the minimum and maximum values in x and y-dimension to which the reference window shall be set, see Details.

Details

Possible data-structures are

- an object of class `Extent`,
- an object of class `bbox`,
- a table with two columns (named x and y) containing the minimum and maximum values for each dimension.

Value

The object `x` with an update reference window.

See Also

Other setters: `setCRS()`, `setFeatures()`, `setGroups()`, `setHistory()`

Examples

```
# create a polygon programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))
(aGeom <- gs_polygon(anchor = coords))
visualise(aGeom)

window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
(aGeom <- setWindow(x = aGeom, to = window))

visualise(aGeom)

window <- data.frame(x = c(0, 2),
                     y = c(0, 2))
setWindow(x = gtPPP, to = window)
```

show,geom-method *Print geom in the console*

Description

Print geom in the console

Usage

```
## S4 method for signature 'geom'
show(object)
```

Arguments

object	[geom] object to show.

show,gtTheme-method *Print gtTheme in the console*

Description

Print gtTheme in the console

Usage

```
## S4 method for signature 'gtTheme'
show(object)
```

Arguments

object	[gtTheme] object to show.

visualise *Visualise raster and geom objects*

Description

Visualise raster and geom objects

Usage

```
visualise(
  ...,
  layer = NULL,
  window = NULL,
  theme = gtTheme,
  trace = FALSE,
  image = FALSE,
  new = TRUE,
  clip = TRUE
)
```

Arguments

...	objects to plot and optional graphical parameters.
layer	[integerish(.) | character(.)] in case the objects to plot have several layers, this is the name or index of the layer(s) that shall be plotted.
window	[data.frame(1)] two opposing corners of a rectangle to which the plot is limited.
theme	[list(7)] visualising options; see setTheme for details.
trace	[logical(1)] Print the raster object's history (i.e. the process according to which it has been created) (TRUE), or simply plot the object (FALSE, default).
image	[logical(1)] set this to TRUE if raster is actually an image; see Details.
new	[logical(1)] force a new plot (TRUE, default).
clip	[logical(1)] clip the plot by the plot box (TRUE, default), or plot all of the objects.

Details

In case you want to plot an image (similar to plotRGB), you either have to:

1. provide a RasterStack with the three layers red, green and blue or

2. provide a matrix with hexadecimal colour values (e.g. '#000000')

and set `image = TRUE`.

Value

Returns invisibly an object of class `recordedplot`, see `recordPlot` for details (and warnings).

Examples

```
coords <- data.frame(x = c(30, 60, 60, 40),
                     y = c(40, 40, 60, 70),
                     fid = 1)
(aGeom <- gs_polygon(anchor = coords))
visualise(aGeom)

window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
withWindow <- setWindow(x = aGeom, to = window)
visualise(expanded = withWindow)

(aRaster <-  gtRasters$categorical)

# plot several objects together
visualise(aRaster, aGeom)

# give names
visualise(`a raster` = aRaster, `a geom` = aGeom)

# use graphical parameters ...
visualise(aGeom, linecol = "green")

# ... or a theme
visualise(aRaster, theme = setTheme(title = list(plot = FALSE)))
```

Index