Package ‘geometr’

March 30, 2020

Title  Generate and Modify Interoperable Geometric Shapes

Version  0.2.4

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://github.com/EhrmannS/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, sf, sp, spatstat, raster, crayon, deldir, dplyr, grDevices, grid, methods, rgdal, rlang, tibble, readr

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

LinkingTo  Rcpp

RoxygenNote  7.1.0

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@funroll-loops.de>

Repository  CRAN

Date/Publication  2020-03-30 10:20:02 UTC
## R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>.getDecimals</td>
<td>3</td>
</tr>
<tr>
<td>.makeTinyMap</td>
<td>3</td>
</tr>
<tr>
<td>.rad</td>
<td>4</td>
</tr>
<tr>
<td>.testAnchor</td>
<td>4</td>
</tr>
<tr>
<td>.testTemplate</td>
<td>5</td>
</tr>
<tr>
<td>.testWindow</td>
<td>5</td>
</tr>
<tr>
<td>.updateOrder</td>
<td>6</td>
</tr>
<tr>
<td>.updateVertices</td>
<td>6</td>
</tr>
<tr>
<td>.updateWindow</td>
<td>7</td>
</tr>
<tr>
<td>gc_geom</td>
<td>7</td>
</tr>
<tr>
<td>gc_grob</td>
<td>8</td>
</tr>
<tr>
<td>gc_ppp</td>
<td>9</td>
</tr>
<tr>
<td>gc_raster</td>
<td>9</td>
</tr>
<tr>
<td>gc_sf</td>
<td>10</td>
</tr>
<tr>
<td>gc_sp</td>
<td>11</td>
</tr>
<tr>
<td>geom-class</td>
<td>11</td>
</tr>
<tr>
<td>geometr</td>
<td>13</td>
</tr>
<tr>
<td>getCRS</td>
<td>13</td>
</tr>
<tr>
<td>getExtent</td>
<td>14</td>
</tr>
<tr>
<td>getFeatures</td>
<td>15</td>
</tr>
<tr>
<td>getGroups</td>
<td>16</td>
</tr>
<tr>
<td>getHistory</td>
<td>17</td>
</tr>
<tr>
<td>getLayer</td>
<td>18</td>
</tr>
<tr>
<td>getName</td>
<td>19</td>
</tr>
<tr>
<td>getPoints</td>
<td>20</td>
</tr>
<tr>
<td>getRes</td>
<td>21</td>
</tr>
<tr>
<td>getType</td>
<td>22</td>
</tr>
<tr>
<td>getWindow</td>
<td>23</td>
</tr>
<tr>
<td>gs_line</td>
<td>24</td>
</tr>
<tr>
<td>gs_point</td>
<td>26</td>
</tr>
<tr>
<td>gs_polygon</td>
<td>27</td>
</tr>
<tr>
<td>gs_random</td>
<td>29</td>
</tr>
<tr>
<td>gs_tiles</td>
<td>30</td>
</tr>
<tr>
<td>gs_voronoi</td>
<td>32</td>
</tr>
<tr>
<td>gtGeoms</td>
<td>33</td>
</tr>
<tr>
<td>gtPPP</td>
<td>34</td>
</tr>
<tr>
<td>gtRasters</td>
<td>34</td>
</tr>
<tr>
<td>gtSF</td>
<td>35</td>
</tr>
<tr>
<td>gtSP</td>
<td>35</td>
</tr>
<tr>
<td>gtTheme</td>
<td>36</td>
</tr>
<tr>
<td>gtTheme-class</td>
<td>36</td>
</tr>
<tr>
<td>gt_locate</td>
<td>37</td>
</tr>
<tr>
<td>gt_reflect</td>
<td>38</td>
</tr>
<tr>
<td>gt_rotate</td>
<td>39</td>
</tr>
<tr>
<td>gt_scale</td>
<td>40</td>
</tr>
<tr>
<td>gt_sketch</td>
<td>41</td>
</tr>
</tbody>
</table>
.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**

```r
.rad(degree)
```

**Arguments**

- `degree`
  
  numeric(1)

  A degree value to convert to radians.

---

### .testAnchor

*Test anchor for consistency*

**Description**

Test anchor for consistency

**Usage**

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

**Arguments**

- `x`
  
  data.frame | geom

  The object to be tested for consistency.

- `...`

  Additional arguments.
Description

Test template for consistency

Usage

testTemplate(x, ...)

Arguments

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

... [.]
additional arguments.

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

```r
## 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, ...)
```
Transform a spatial object to a grob

Usage

```r
## 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_grob()`, `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
```r
## 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_raster()`, `gc_ppp()`, `gc_raster()`, `gc_sf()`, `gc_sp()`

### Examples
```r
gc_ppp(input = gtGeoms$point)
```

---

**gc_raster**

Transform a spatial object to class `Raster*`

### Description
Transform a spatial object to class `Raster*`

### Usage
```r
## 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_raster()`, `gc_sf()`, `gc_sp()`

### Examples
```r
gc_raster(input = gtGeoms$point)
```
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(input = gtGeoms$point)
gc_sf(input = gtGeoms$line)
gc_sf(input = gtGeoms$polygon)
Transform a spatial object to class `Spatial`

### Description
Transform a spatial object to class `Spatial`

### Usage
```r
## 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
```r
gc_sp(input = gtGeoms$point)
gc_sp(input = gtGeoms$line)
gc_sp(input = gtGeoms$polygon)
```

---

**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.
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

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

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

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

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

Usage

```r
## 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)
```

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.
### 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**

```r
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**

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

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

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

Get the table of group attributes

Description

Get tabular information of the attributes of groups of features.

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)
### Usage

```r
## 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

```r
groups(x = gtGeoms$polygon)

# for raster objects, groups are pixels with the same value and their
# attributes are in the raster attribute table (RAT)
groups(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)
**getName**

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()`

---

**getDescription()**

Get the description(s) of a spatial object.

Usage

```r
## S4 method for signature 'ANY'
getDescription(x)

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

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

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

Arguments

- `x` the object from which to get the description.

Value

A list of the requested descriptions.

See Also

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

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)

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

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

Arguments

x the object from which to derive the resolution.

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)
getWindow

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**

```r
## 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(.)]\nnumber 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.
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

```r
# 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
cords <- 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 = cords))

# the vertices are plotted relative to the window
library(magrittr)
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
gs_point(anchor = cords, 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 = gRasters$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

```r
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_polygon

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

### Arguments

- **anchor**
  ```r
  [geom(1)|data.frame(1)]
  ```
  Object to derive the geom from. It must include column names `x`, `y` and optionally a custom `fid`.

- **window**
  ```r
  [data.frame(1)]
  ```
  In case the reference window deviates from the bounding box of `anchor` (minimum and maximum values), specify this here.

- **features**
  ```r
  [integerish(1)]
  ```
  Number of polygons to create.

- **vertices**
  ```r
  [integerish(.)]
  ```
  Number of vertices per polygon; will be recycled if it does not have as many elements as specified in `features`.

- **sketch**
  ```r
  [raster(1)]
  ```
  Raster object that serves as template to sketch polygons.

- **regular**
  ```r
  [logical(1)]
  ```
  Should the polygon be regular, i.e. point symmetric (`TRUE`) or should the vertices be selected as provided by `anchor` (`FALSE`, default)?

- **...**
  ```r
  [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

```r
# 1. create a polygon programmatically
cords <- 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 = cords))

# the vertices are plottet relative to the window
library(magrittr)
window <- data.frame(x = c(0, 80),
                    y = c(0, 80))
gs_polygon(anchor = cords, 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")
Usage

```r
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

```r
# create a squared tiling
library(magrittr)
aWindow <- data.frame(x = c(-180, 180), 
y = c(-60, 80))
gs_tiles(ancestor = 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(ancestor = coords, window = window)
visualise(`honeycomb background` = aGeom)
gs_tiles(ancestor = aGeom, width = 8, pattern = "hexagonal") %>%
visualise(., linecol = "deeppink", new = FALSE)
```

---

### gs_voronoi

Create a voronoi tiling geom

#### Description

Create a voronoi tiling geom

#### Usage

```r
gs_voronoi(ancestor = 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.
gtGeoms

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)

<table>
<thead>
<tr>
<th>gtGeoms</th>
<th>Example geom objects</th>
</tr>
</thead>
</table>

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**

```r
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**

```r
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-class

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

Description

Reflect geoms across a reflection axis.

Usage

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

Arguments

gem \[geom(.)\]
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.
gt_rotate

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(anchors = 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. |
gt_scale

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

```r
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

```r
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.
gt_skew

**See Also**

Other geometry tools: `gt Reflect()`, `gt Rotate()`, `gt Scale()`, `gt Skew()`, `gt Stretch()`, `gt Translate()`

**Examples**

```r
# 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**

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geom</td>
<td>[geom(.)]</td>
</tr>
<tr>
<td>x</td>
<td>[numeric(1)]</td>
</tr>
<tr>
<td>y</td>
<td>[numeric(1)]</td>
</tr>
<tr>
<td>fid</td>
<td>[integerish(.)]</td>
</tr>
<tr>
<td>update</td>
<td>[logical(1)]</td>
</tr>
</tbody>
</table>

**Value**

Skewed geom.
See Also

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

Examples

```r
# the original object
coords <- data.frame(x = c(30, 60, 60, 10, 40, 20),
                      y = c(40, 40, 60, 70, 10, 20),
                      fid = c(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.
gt_translate

See Also

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

Examples

```r
# 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

```r
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

```r
# the original object
doors <- 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 = doors, 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**

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

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

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

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

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

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()`

Description

Set a table of feature attributes

Usage

```r
## 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)
```
**setGroups**

**Arguments**

- **x**
  the object to which to assign a new attribute table.

- **table**
  
  ```r
  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**

```r
## 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**
  
  ```r
  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

```r
## 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()] 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

```r
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

```r
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 datastructure that contains the minimum and maximum values in x and y-dimension to which the reference window shall be set, see Details.

Details

Possible datastructures 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 raster and geom objects

Description

Visualise raster and geom objects

Usage

```r
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

Topic datasets
   gtGeoms, 33
   gtPPP, 34
   gtRasters, 34
   gtSF, 35
   gtSP, 35
   gtTheme, 36
   .getDecimals, 3
   .makeTinyMap, 3
   .rad, 4
   .testAnchor, 4
   .testTemplate, 5
   .testWindow, 5
   .updateOrder, 6
   .updateVertices, 6
   .updateWindow, 7

   gc_geom, 7, 9–11
   gc_geom,ppp-method (gc_geom), 7
   gc_geom,Raster-method (gc_geom), 7
   gc_geom,Spatial-method (gc_geom), 7
   gc_grob, 8, 8, 9–11
   gc_grob,geom-method (gc_grob), 8
   gc_ppp, 8, 9, 10, 11
   gc_ppp,geom-method (gc_ppp), 9
   gc_raster, 8, 9, 10, 11
   gc_raster,geom-method (gc_raster), 9
   gc_sf, 8–10, 10, 11
   gc_sf,geom-method (gc_sf), 10
   gc_sp, 8–10, 11
   gc_sp,geom-method (gc_sp), 11
   geom, 24, 26, 27
   geom (geom-class), 11
   geom-class, 11
   geometr, 13
   getCRS, 13, 15–23
   getCRS,ANY-method (getCRS), 13
   getCRS,geom-method (getCRS), 13
   getCRS,ppp-method (getCRS), 13
   getCRS,Raster-method (getCRS), 13
   getCRS,Spatial-method (getCRS), 13
   getExtent, 14, 14, 16–23
   getExtent,ANY-method (getExtent), 14
   getExtent,geom-method (getExtent), 14
   getExtent,matrix-method (getExtent), 14
   getExtent,ppp-method (getExtent), 14
   getExtent,Raster-method (getExtent), 14
   getExtent,sf-method (getExtent), 14
   getExtent,Spatial-method (getExtent), 14
   getFeatures, 14, 15, 15, 17–23
   getFeatures,ANY-method (getFeatures), 15
   getFeatures,geom-method (getFeatures), 15
   getFeatures,matrix-method (getFeatures), 15
   getFeatures,ppp-method (getFeatures), 15
   getFeatures,Raster-method (getFeatures), 15
   getFeatures,sf-method (getFeatures), 15
   getFeatures, Spatial-method (getFeatures), 15
   getGroups, 14–16, 16, 18–23
   getHistory, 14–17, 17, 19–23
   getHistory,ANY-method (getHistory), 17
   getHistory,geom-method (getHistory), 17
   getHistory,Raster-method (getHistory), 17
   getLayer, 14–18, 18, 19–23
   getLayer,ANY-method (getLayer), 18
   getLayer,geom-method (getLayer), 18
   getLayer,matrix-method (getLayer), 18
   getLayer,Raster-method (getLayer), 18
   getLayer,sf-method (getLayer), 18

   57
getLayer, Spatial-method (getLayer), 18
getName, 14–19, 19, 20–23
getName, ANY-method (getName), 19
getName, geom-method (getName), 19
gtype, Raster-method (getName), 19
getName, sf-method (getName), 19
getPoints, 14–19, 20, 21–23
getPoints, ANY-method (getPoints), 20
getPoints, geom-method (getPoints), 20
getPoints, matrix-method (getPoints), 20
getPoints, ppp-method (getPoints), 20
getPoints, Raster-method (getPoints), 20
getPoints, sf-method (getPoints), 20
getPoints, Spatial-method (getPoints), 20
getRes, 14–20, 21, 22, 23
getRes, ANY-method (getRes), 21
getRes, geom-method (getRes), 21
getRes, matrix-method (getRes), 21
getRes, Raster-method (getRes), 21
gtype, 14–21, 22, 23
gtype, ANY-method (getType), 22
gtype, geom-method (getType), 22
gtype, matrix-method (getType), 22
gtype, ppp-method (getType), 22
gtype, Raster-method (getType), 22
gtype, sf-method (getType), 22
gtype, Spatial-method (getType), 22
gWindow, 14–22, 23
gWindow, ANY-method (getWindow), 23
gWindow, geom-method (getWindow), 23
gWindow, matrix-method (getWindow), 23
gWindow, ppp-method (getWindow), 23
gWindow, Raster-method (getWindow), 23
gWindow, sf-method (getWindow), 23
gWindow, Spatial-method (getWindow), 23
gpar, 8, 25, 26, 28, 31, 32, 47
gs_hingagon (gs_polygon), 27
gs_line, 24, 26, 29, 30
gs_point, 25, 26, 29, 30
gs_polygon, 25, 26, 27, 30
gs_random, 25, 26, 29, 29
gs_rectangle (gs_polygon), 27
gs_square (gs_polygon), 27
gs_tiles, 30, 33
gs_triangle (gs_polygon), 27
gs_voronoi, 31, 32
gt_locate, 25, 26, 28, 31, 32, 37, 42
gt_reflect, 38, 40, 41, 43–46
gt_rotate, 39, 39, 41, 43–46
gt_scale, 39, 40, 40, 43–46
gt_sketch, 39–41, 41, 44–46
gt_skew, 39–41, 43, 45, 46
gt_stretch, 39–41, 43, 44, 46
gt_translate, 39–41, 43–45, 45
gtGeoms, 33
gtPPP, 34
gtRasters, 34
gtSF, 35
gtSP, 35
gtTheme, 36
gtTheme-class, 36
makeLayout, 46
makeObject, 47
plotRGB, 55
pointsGrob, 8
recordPlot, 56
setCRS, 47, 49, 50, 53
setCRS, ANY-method (setCRS), 47
setCRS, geom-method (setCRS), 47
setCRS, Raster-method (setCRS), 47
setCRS, sf-method (setCRS), 47
setCRS, Spatial-method (setCRS), 47
setFeatures, 48, 48, 49, 50, 53
setFeatures, ANY-method (setFeatures), 48
setFeatures, geom-method (setFeatures), 48
setFeatures, ppp-method (setFeatures), 48
setFeatures, sf-method (setFeatures), 48
setFeatures, Spatial-method (setFeatures), 48
setGroups, 48, 49, 49, 50, 53
setGroups, ANY-method (setGroups), 49
setGroups, geom-method (setGroups), 49
setGroups, RasterLayer-method (setGroups), 49
setHistory, 48, 49, 50, 53
setHistory, ANY-method (setHistory), 50
setHistory, geom-method (setHistory), 50
setHistory, RasterLayer-method (setHistory), 50
setWindow, 8, 36, 47, 51, 55
setWindow, 48–50, 52
INDEX

setWindow, ANY-method (setWindow), 52
setWindow, geom-method (setWindow), 52
setWindow, ppp-method (setWindow), 52
show, geom-method, 54
show, gtTheme-method, 54

themeClass (gtTheme-class), 36

visualise, 36, 55