Package ‘stars’

July 8, 2020

Title   Spatiotemporal Arrays, Raster and Vector Data Cubes

Version 0.4-3

Description Reading, manipulating, writing and plotting
spatiotemporal arrays (raster and vector data cubes) in 'R', using 'GDAL'
bindings provided by 'sf', and 'NetCDF' bindings by 'ncmeta' and 'RNetCDF'.

License  Apache License

URL https://r-spatial.github.io/stars/,
 https://github.com/r-spatial/stars/

BugReports  https://github.com/r-spatial/stars/issues/

Additional Repositories http://gis-bigdata.uni-muenster.de/pebesma/

Depends  R (>= 3.3.0), abind, sf (>= 0.9-0)

Imports  methods, parallel, classInt (>= 0.4-1), lwgeom, rlang, units

Suggests PCICt, RNetCDF (>= 1.8-2), covr, cubelyr, digest, dplyr (>= 0.7-0), exactextractr, future.apply, ggforce, ggplot2,
ggthemes, gstat, httr, jsonlite, knitr, maps, ncdfgeom, ncmeta
(>= 0.0.3), pbapply, plm, raster, rmarkdown, s2 (>= 1.0.0), sp,
spacetime, spatstat, starsdata, testthat, viridis, xts, zoo

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote  7.1.0

Collate  'init.R' 'stars.R' 'read.R' 'sf.R' 'dimensions.R' 'values.R'
'plot.R' 'tidyverse.R' 'transform.R' 'ops.R' 'write.R'
'raster.R' 'sp.R' 'spacetime.R' 'ncdf.R' 'proxy.R' 'factors.R'
'rasterize.R' 'subset.R' 'warp.R' 'aggregate.R' 'xts.R'
'intervals.R' 'geom.R' 'mosaic.R' 'spatstat.R'
'OpenStreetMap.R' 'sample.R' 'extract.R'

NeedsCompilation  no

Author  Edzer Pebesma [aut, cre] (<https://orcid.org/0000-0001-8049-7069>),
Michael Sumner [ctb] (<https://orcid.org/0000-0002-2471-7511>),
Etienne Racine [ctb],
Adriano Fantini [ctb],
David Blodgett [ctb]
Maintainer Edzer Pebesma <edzer.pebesma@uni-muenster.de>
Repository CRAN
Date/Publication 2020-07-08 11:20:03 UTC

R topics documented:

aggregate.stars .................................................. 3
as ................................................................. 4
c.stars .......................................................... 5
contour.stars .................................................... 6
cut_stars .......................................................... 6
dplyr ............................................................. 7
geom_stars ........................................................ 8
make_intervals .................................................... 9
ops_stars .......................................................... 10
plot .............................................................. 11
read_ncdf ......................................................... 14
read_stars ........................................................ 15
redimension ......................................................... 18
stars_subset ....................................................... 18
st_apply .......................................................... 20
st_as_sf .......................................................... 21
st_as_stars ......................................................... 23
st_contour ........................................................ 25
st_coordinates ..................................................... 26
st_crop ........................................................... 27
st_dimensions ..................................................... 29
st_dim_to_attr .................................................... 31
st_extract ......................................................... 32
st_intersects.stars ............................................... 32
st_join.stars ...................................................... 33
st_mosaic ........................................................ 34
st_rasterize ....................................................... 36
st_raster_type .................................................... 37
st_rgb ........................................................... 37
st_sfc2xy ........................................................ 38
st_transform ....................................................... 39
st_warp .......................................................... 40
st_xy2sfc ........................................................ 41
write_stars ......................................................... 42

Index 44
aggregate.stars

Description

spatially or temporally aggregate stars object, returning a data cube with lower spatial or temporal resolution

Usage

## S3 method for class 'stars'
aggregate(
  x,
  by,
  FUN,
  ..., 
  drop = FALSE,
  join = st_intersects,
  as_points = any(st_dimension(by) == 2, na.rm = TRUE),
  rightmost.closed = FALSE,
  left.open = FALSE,
  exact = FALSE
)

Arguments

x object of class stars with information to be aggregated
by object of class sf or sfc for spatial aggregation, for temporal aggregation a vector with time values (Date, POSIXct, or PCICT) that is interpreted as a sequence of left-closed, right-open time intervals or a string like "months", "5 days" or the like (see cut.POSIXt); if by is an object of class stars, it is converted to sfc by st_as_sfc(by, as_points = FALSE) thus ignoring its time component.
FUN aggregation function, such as mean
... arguments passed on to FUN, such as na.rm=TRUE
drop logical; ignored
join function; function used to find matches of x to by
as_points see st_as_sf: shall raster pixels be taken as points, or small square polygons?
rightmost.closed see findInterval
left.open logical; used for time intervals, see findInterval and cut.POSIXt
exact logical; if TRUE, use coverage_fraction to compute exact overlap fractions of polygons with raster cells
Examples

```r
# aggregate time dimension in format Date
tif = system.file("tif/L7_ETMs.tif", package = "stars")
t1 = as.Date("2018-07-31")
x = read_stars(c(tif, tif, tif, tif), along = list(time = c(t1, t1+1, t1+2, t1+3)))[,1:30,1:30]
st_get_dimension_values(x, "time")
x_agg_time = aggregate(x, by = t1 + c(0, 2, 4), FUN = max)

# aggregate time dimension in format Date - interval
by_t = "2 days"
x_agg_time2 = aggregate(x, by = by_t, FUN = max)
st_get_dimension_values(x_agg_time2, "time")
x_agg_time - x_agg_time2

# aggregate time dimension in format POSIXct
x = st_set_dimensions(x, 4, values = as.POSIXct(c("2018-07-31", "2018-08-01", "2018-08-02", "2018-08-03")), names = "time")
by_t = as.POSIXct(c("2018-07-31", "2018-08-02"))
x_agg_posix = aggregate(x, by = by_t, FUN = max)
st_get_dimension_values(x_agg_posix, "time")
x_agg_time - x_agg_posix

aggregate(x, "2 days", mean)

# Spatial aggregation, see https://github.com/r-spatial/stars/issues/299
prec_file = system.file("nc/test_stageiv_xyt.nc", package = "stars")
prec = read_ncdf(prec_file, curvilinear = c("lon", "lat"))
prec_slice = dplyr::slice(prec, index = 17, along = "time")
nc = sf::read_sf(system.file("gpkg/nc.gpkg", package = "sf"), "nc.gpkg")
nc = st_transform(nc, st_crs(prec_slice))
agg = aggregate(prec_slice, st_geometry(nc), mean)
plot(agg)
```

---

**as**

Coerce stars object into a Raster raster or brick

**Description**

Coerce stars object into a Raster raster or brick

**Arguments**

- from: object to coerce
c.stars  

**combine multiple stars objects, or combine multiple attributes in a single stars object into a single array**

Description

combine multiple stars objects, or combine multiple attributes in a single stars object into a single array

Usage

```r
## S3 method for class 'stars'
c(..., along = NA_integer_, try_hard = FALSE, nms = names(list(...)))
```

```r
## S3 method for class 'stars_proxy'
c(
  ...
  along = NA_integer_,
  along_crs = FALSE,
  try_hard = FALSE,
  nms = names(list(...))
)
```

Arguments

- `...`: object(s) of class `star`: in case of multiple arguments, these are combined into a single stars object, in case of a single argument, its attributes are combined into a single attribute. In case of multiple objects, all objects should have the same dimensionality.
- `along`: integer; see `read_stars`
- `try_hard`: logical; if `TRUE` and some arrays have different dimensions, combine those that dimensions matching to the first array
- `nms`: character; vector with array names
- `along_crs`: logical; if `TRUE`, combine arrays along a CRS dimension

Examples

```r
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
(new = c(x, x))
c(new) # collapses two arrays into one with an additional dimension
c(x, x, along = 3)
```
**contour.stars**

*plot contours of a stars object*

**Description**

plot contours of a stars object

**Usage**

```r
## S3 method for class 'stars'
contour(x, ...)
```

**Arguments**

- `x` object of class `stars`
- `...` other parameters passed on to `contour`

**Details**

this uses the R internal contour algorithm, which (by default) plots contours; `st_contour` uses the GDAL contour algorithm that returns contours as simple features.

**Examples**

```r
d = st_dimensions(x = 1:ncol(volcano), y = 1:nrow(volcano))
r = st_as_stars(t(volcano))
r = st_set_dimensions(r, 1, offset = 0, delta = 1)
r = st_set_dimensions(r, 2, offset = 0, delta = -1)
plot(r, reset = FALSE)
contour(r, add = TRUE)
```

---

**cut_stars**

*cut methods for stars objects*

**Description**

cut methods for stars objects

**Usage**

```r
## S3 method for class 'array'
cut(x, breaks, ...)
```

```r
## S3 method for class 'matrix'
cut(x, breaks, ...)
```

```r
## S3 method for class 'stars'
cut(x, breaks, ...)
```
Arguments

- **x**
  - see cut
- **breaks**
  - see cut
- **...**
  - see cut

Details

R’s `factor` only works for vectors, not for arrays or matrices. This is a work-around (or hack?) to keep the factor levels generated by `cut` and use them in plots.

Value

an array or matrix with a `levels` attribute; see details

Examples

```r
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
cut(x, c(0, 50, 100, 255))
cut(x[,,1], c(0, 50, 100, 255))
plot(cut(x[,,1], c(0, 50, 100, 255)))
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
(x1_cut = cut(x1, breaks = c(0, 50, 100, Inf))) # shows factor in summary
plot(x1_cut[,,c(3,6)]) # propagates through [ and plot
```
pull.stars(.data, var = -1)
pull.stars_proxy(.data, ...)
as.tbl_cube.stars(x, ...)
slice.stars(.data, along, index, ..., drop = length(index) == 1)
slice.stars_proxy(.data, ...)

Arguments

.data object of class stars
...
see filter
var see pull
x object of class stars
along name or index of dimension to which the slice should be applied
index integer value(s) for this index
drop logical; drop dimensions that only have a single index?

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
library(dplyr)
x1 %>% slice("band", 2:3)
x1 %>% slice("x", 50:100)

---

geom_stars ggplot geom for stars objects

Description

ggplot geom for stars objects

Usage

geom_stars(mapping = NULL, data = NULL, ..., downsample = 0, sf = FALSE)
theme_stars(…)
make_intervals

Arguments

mapping see geom_raster
data see geom_raster
... see geom_raster
downsample downsampling rate: e.g. 3 keeps rows and cols 1, 4, 7, 10 etc.; a value of 0 does not downsample
sf logical; if TRUE rasters will be converted to polygons and plotted using geom_sf.

Details

geom_stars returns (a call to) either geom_raster, geom_tile, or geom_sf, depending on the raster or vector geometry; for the first to, an aes call is constructed with the raster dimension names and the first array as fill variable. Further calls to coord_equal and facet_wrap are needed to control aspect ratio and the layers to be plotted; see examples.

Examples

```
system.file("tif/L7_ETMs.tif", package = "stars") %>% read_stars() -> x
library(ggplot2)
ggplot() + geom_stars(data = x) +
  coord_equal() +
  facet_wrap(~band) +
  theme_void() +
  scale_x_discrete(expand=c(0,0))+
  scale_y_discrete(expand=c(0,0))
```

make_intervals create an intervals object

Description

create an intervals object, assuming left-closed and right-open intervals

Usage

make_intervals(start, end)

Arguments

start vector with start values, or 2-column matrix with start and end values in column 1 and 2, respectively
end vector with end values
Description

Ops functions for stars objects, including comparison, product and divide, add, subtract

Usage

```r
## S3 method for class 'stars'
Ops(e1, e2)

## S3 method for class 'stars'
Math(x, ...)

## S3 method for class 'stars_proxy'
Ops(e1, e2)

## S3 method for class 'stars_proxy'
Math(x, ...)
```

Arguments

- `e1`: object of class `stars`
- `e2`: object of class `stars`
- `x`: object of class `stars`
- `...`: parameters passed on to the Math functions

Value

object of class `stars`

Examples

```r
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x * x
x / x
x + x
x + 10
all.equal(x * 10, 10 * x)
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
a = sqrt(x)
```

```r
b = log(x, base = 10)
```
plot stars object, with subplots for each level of first non-spatial dimension

Description
plot stars object, with subplots for each level of first non-spatial dimension, and customization of legend key

Usage
```r
## S3 method for class 'stars'
plot(
x,
y,
...
join_zlim = TRUE,
main = make_label(x, 1),
axes = FALSE,
downsampel = TRUE,
nbreaks = 11,
bbreaks = "quantile",
col = grey(1:(nbreaks - 1)/nbreaks),
key.pos = get_key_pos(x, ...),
key.width = lcm(1.8),
key.length = 0.618,
reset = TRUE,
box.col = grey(0.8),
center_time = FALSE,
hook = NULL
)

## S3 method for class 'stars'
image(
x,
...
band = 1,
attr = 1,
asp = NULL,
rgb = NULL,
maxColorValue = ifelse(inherits(rgb, "data.frame"), 255, max(x[[attr]], na.rm =
TRUE)),
xlab = if (!axes) "" else names(d)[1],
ylab = if (!axes) "" else names(d)[2],
xlim = st_bbox(extent)$xlim,
ylim = st_bbox(extent)$ylim,
text_values = FALSE,
```
plot

axes = FALSE,
interpolate = FALSE,
as_points = FALSE,
key.pos = NULL,
logz = FALSE,
key.width = lcm(1.8),
key.length = 0.618,
add.geom = NULL,
border = NA,
useRaster = dev.capabilities("rasterImage")$rasterImage == "yes",
extent = x
)

## S3 method for class 'stars_proxy'
plot(x, y, ..., downsample = get_downsample(dim(x)))

Arguments

x
  object of class stars

y
  ignored

...
  further arguments: for plot, passed on to image.stars; for image, passed on to image.default or rasterImage.

join_zlim
  logical; if TRUE, compute a single, joint zlim (color scale) for all subplots from x

main
  character; subplot title prefix; use "" to get only time, use NULL to suppress subplot titles

axes
  logical; should axes and box be added to the plot?

downsample
  logical or numeric; if TRUE will try to plot not many more pixels than actually are visible, if FALSE, no downsampling takes place, if numeric, the downsampling rate; see Details.

nbreaks
  number of color breaks; should be one more than number of colors. If missing and col is specified, it is derived from that.

breaks
  actual color breaks, or a method name used for classIntervals.

col
  colors to use for grid cells

key.pos
  integer; side to plot a color key: 1 bottom, 2 left, 3 top, 4 right; set to NULL to omit key. Ignored if multiple columns are plotted in a single function call. Default depends on plot size, map aspect, and, if set, parameter asp.

key.width
  amount of space reserved for width of the key (labels); relative or absolute (using lcm)

key.length
  amount of space reserved for length of the key (labels); relative or absolute (using lcm)

reset
  logical; if FALSE, keep the plot in a mode that allows adding further map elements; if TRUE restore original mode after plotting; see details.

box_col
  color for box around sub-plots; use 0 to suppress plotting of boxes around sub-plots.
center_time logical; if TRUE, sub-plot titles will show the center of time intervals, otherwise their start

hook NULL or function; hook function that will be called on every sub-plot.

band integer; which band (dimension) to plot

attr integer; which attribute to plot

asp numeric; aspect ratio of image

rgb integer; specify three bands to form an rgb composite. Experimental: rgb color table; see Details.

maxColorValue numeric; passed on to rgb

xlab character; x axis label

ylab character; y axis label

xlim x axis limits

ylim y axis limits

text_values logical; print values as text on image?

interpolate logical; when using rasterImage (rgb), should pixels be interpolated?

as_points logical; for curvilinear or sheared grids: parameter passed on to st_as_sf, determining whether raster cells will be plotted as symbols (fast, approximate) or small polygons (slow, exact)

logz logical; if TRUE, use log10-scale for the attribute variable. In that case, breaks and at need to be given as log10-values; see examples.

add.geom object of class sfc, or list with arguments to plot, that will be added to an image or sub-image

border color used for cell borders (only in case x is a curvilinear or rotated/sheared grid)

useRaster logical; use the rasterImage capabilities of the graphics device?

extent object which has a st_bbox method; sets the plotting extent

Details

Downsampling: a value for downsample of 0 or 1 causes no downsampling, 2 that every second dimension value is sampled, 3 that every third dimension value is sampled, and so on.

use of an rgb color table is experimental; see https://github.com/r-spatial/mapview/issues/208 when plotting a subsetted stars_proxy object, the default value for argument downsample will not be computed correctly, and it has to be set manually.

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
image(x, col = grey((3:9)/10))
image(x, rgb = c(1,3,5)) # rgb composite
read_ncdf

Read NetCDF into stars object

Description

Read data from a file (or source) using the NetCDF library directly.

Usage

```r
read_ncdf(
  .x,
  ..., 
  var = NULL,
  ncsub = NULL,
  curvilinear = character(0),
  eps = 1e-12,
  ignore_bounds = FALSE,
  make_time = TRUE,
  make_units = TRUE
)
```

Arguments

- `.x` NetCDF file or source
- `...` ignored
- `var` variable name or names (they must be on matching grids)
- `ncsub` matrix of start, count columns (see Details)
- `curvilinear` length two character named vector with names of variables holding longitude and latitude values for all raster cells. "stars" attempts to figure out appropriate curvilinear coordinates if they are not supplied.
- `eps` numeric; dimension value increases are considered identical when they differ less than `eps`
- `ignore_bounds` logical; should bounds values for dimensions, if present, be ignored?
- `make_time` if TRUE (the default), an attempt is made to provide a date-time class from the "time" variable
- `make_units` if TRUE (the default), an attempt is made to set the units property of each variable

Details

The following logic is applied to coordinates. If any coordinate axes have regularly spaced coordinate variables they are reduced to the offset/delta form with `affine = c(0, 0)`, otherwise the values of the coordinates are stored and used to define a rectilinear grid.

If the data has two or more dimensions and the first two are regular they are nominated as the 'raster' for plotting.
If the curvilinear argument is used it specifies the 2D arrays containing coordinate values for the first two dimensions of the data read. It is currently assumed that the coordinates are 2D and that they relate to the first two dimensions in that order.

If var is not set the first set of variables on a shared grid is used.

start and count columns of ncsb must correspond to the variable dimension (nrows) and be valid index using var.get.nc convention (start is 1-based). If the count value is NA then all steps are included. Axis order must match that of the variable/s being read.

Examples

```r
f <- system.file("nc/reduced.nc", package = "stars")
read_ncdf(f)
read_ncdf(f, var = c("anom"))
read_ncdf(f, ncsb = cbind(start = c(1, 1, 1), count = c(10, 12, 1, 1)))
```

```r
#' precipitation data in a curvilinear NetCDF
prec_file = system.file("nc/test_stageiv_xyt.nc", package = "stars")
prec = read_ncdf(prec_file, curvilinear = c("lon", "lat"), ignore_bounds = TRUE)

#plot(prec) # gives error about unique breaks
# remove NAs, zeros, and give a large number
# of breaks (used for validating in detail)
qu_0_omit = function(x, ..., n = 22) {
  x = units::drop_units(na.omit(x))
  c(0, quantile(x[x > 0], seq(0, 1, length.out = n)))
}
library(dplyr)
prec_slice = slice(prec, index = 17, along = "time")
plot(prec_slice, border = NA, breaks = qu_0_omit(prec_slice[[1]]), reset = FALSE)
nc = sf::read_sf(system.file("gpkg/nc.gpkg", package = "sf"), "nc.gpkg")
plot(st_geometry(nc), add = TRUE, reset = FALSE, col = NA)
```

read_stars  

read raster/array dataset from file or connection

Description

read raster/array dataset from file or connection

Usage

```
read_stars(
  .x,
  ...,
  options = character(0),
  driver = character(0),
  sub = TRUE,
)```
quiet = FALSE,
NA_value = NA_real_,
along = NA_integer_,
RasterIO = list(),
proxy = !length(curvilinear) && is_big(.x, sub = sub, ...),
curvilinear = character(0),
normalize_path = TRUE,
RAT = character(0)
)

is_big(x, ..., sub = sub, n_proxy = options("stars.n_proxy")[[1]] %||% 1e+08)

Arguments

.x character vector with name(s) of file(s) or data source(s) to be read

... passed on to st_as_stars if curvilinear was set

options character; opening options
driver character; driver to use for opening file. To override fixing for subdatasets and
autodetect them as well, use NULL.
sub character, integer or logical; name, index or indicator of sub-dataset(s) to be read

quiet logical; print progress output?

NA_value numeric value to be used for conversion into NA values; by default this is read
from the input file

along length-one character or integer, or list; determines how several arrays are com-
bined, see Details.

RasterIO list with named parameters for GDAL’s RasterIO, to further control the extent,
resolution and bands to be read from the data source; see details.

proxy logical; if TRUE, an object of class stars_proxy is read which contains array
metadata only; if FALSE the full array data is read in memory. Always FALSE for
curvilinear grids. If not set, defaults to TRUE when the number of cells to be read
is larger than options(stars.n_proxy, or to 1e8 if that option was not set.
curvilinear length two character vector with names of subdatasets holding longitude and
latitude values for all raster cells.

normalize_path logical; if FALSE, suppress a call to normalizePath on .x

RAT character; raster attribute table column name to use as factor levels

x object to be read with read_stars

n_proxy integer; number of cells above which .x will be read as stars proxy object, i.e.
not as in-memory arrays but left on disk

Details

In case .x contains multiple files, they will all be read and combined with c.stars. Along which
dimension, or how should objects be merged? If along is set to NA it will merge arrays as new
attributes if all objects have identical dimensions, or else try to merge along time if a dimension
called time indicates different time stamps. A single name (or positive value) for along will merge
along that dimension, or create a new one if it does not already exist. If the arrays should be arranged along one of more dimensions with values (e.g. time stamps), a named list can passed to along to specify them; see example.

RasterIO is a list with zero or more of the following named arguments: nXOff, nYOff (both 1-based: the first row/col has offset value 1), nXSize, nYSize, nBufXSize, nBufYSize, bands, coderesample. see https://www.gdal.org/classGDALDataset.html#a80d005ed10aefafa8a55dc539c2f69da for their meaning; bands is an integer vector containing the band numbers to be read (1-based: first band is 1) Note that if nBufXSize or nBufYSize are specified for downsampling an image, resulting in an adjusted geotransform. resample reflects the resampling method and has to be one of: "nearest_neighbour" (the default), "bilinear", "cubic", "cubic_spline", "lanczos", "average", "mode", or "Gauss".

Value

object of class stars

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
(x1 = read_stars(tif))
(x2 = read_stars(c(tif, tif)))
(x3 = read_stars(c(tif, tif), along = "band")
(x4 = read_stars(c(tif, tif), along = "new_dimensions")) # create 4-dimensional array

x1o = read_stars(tif, options = "OVERVIEW_LEVEL=1")
t1 = as.Date("2018-07-31")

# along is a named list indicating two dimensions:
read_stars(c(tif, tif, tif, tif), along = list(foo = c("bar1", "bar2"), time = c(t1, t1+2)))

m = matrix(1:120, nrow = 12, ncol = 10)
dim(m) = c(x = 10, y = 12) # named dim
st = st_as_stars(m)
attr(st, "dimensions")$y$delta = -1
attr(st, "dimensions")$y$offset = 12
st
tmp = tempfile(fileext = ".tif")
write_stars(st, tmp)
(red <- read_stars(tmp))

read_stars(tmp, RasterIO = list(nXOff = 1, nYOff = 1, nXSize = 10, nYSize = 12, nBufXSize = 2, nBufYSize = 2))[[1]]

red[[1]] # cell values of subsample grid:
plot(st, reset = FALSE, axes = TRUE, ylim = c(-.1,12.1), xlim = c(-.1,10.1),
main = "nBufXSize & nBufYSize demo", text_values = TRUE)
plot(st_as_sfc(red, as_points = TRUE), add = TRUE, col = 'red', pch = 16)
plot(st_as_sfc(st_as_stars(st), as_points = FALSE), add = TRUE, border = 'grey')
plot(st_as_sfc(red, as_points = FALSE), add = TRUE, border = 'green', lwd = 2)
file.remove(tmp)
redimension array, or collapse attributes into a new dimension

Description

redimension array, or collapse attributes into a new dimension

Usage

st_redimension(x, new_dims, along, ...)

## S3 method for class 'stars'
st_redimension(
    x,
    new_dims = st_dimensions(x),
    along = list(new_dim = names(x)),
    ...
)

## S3 method for class 'stars_proxy'
st_redimension(
    x,
    new_dims = st_dimensions(x),
    along = list(new_dim = names(x)),
    ...
)

Arguments

x  object of class stars
new_dims  target dimensions: either a `dimensions` object or an integer vector with the
dimensions' sizes
along  named list with new dimension name and values
...  ignored

stars_subset  subset stars objects

Description

subset stars objects
#### Usage

```r
## S3 method for class 'stars'
x[i = TRUE, ..., drop = FALSE, crop = !is_curvilinear(x)]

## S3 replacement method for class 'stars'
x[i] <- value
```

```r
st_flip(x, which = 1)
```

#### Arguments

- **x**: object of class stars
- **i**: first selector: integer, logical or character vector indicating attributes to select, or object of class sf or sfc used as spatial selector; see details
- **...**: further (logical or integer vector) selectors, matched by order, to select on individual dimensions
- **drop**: logical; if TRUE, degenerate dimensions (with only one value) are dropped
- **crop**: logical; if TRUE and parameter i is a spatial geometry (sf or sfc) object, the extent (bounding box) of the result is cropped to match the extent of i using `st_crop`. Cropping curvilinear grids is not supported.
- **value**: array of dimensions equal to those in x, or a vector or value that will be recycled to such an array
- **which**: character or integer; dimension(s) to be flipped

#### Details

if i is an object of class sf, sfc or bbox, the spatial subset covering this geometry is selected, possibly followed by cropping the extent. Array values for which the cell centre is not inside the geometry are assigned NA.

In an assignment (or replacement form, [<-), argument i needs to be a stars object with dimensions identical to x, and value will be recycled to the dimensions of the arrays in x.

#### Value

`s_t_flip` flips (reverts) the array values along the chosen dimension without(s) changing the dimension properties

#### Examples

```r
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x[,1:3] # select bands
x[,1:100,100:200,] # select x and y by range
x["L7_ETMs.tif"] # select attribute
xy = structure(list(x = c(293253.999046018, 296400.196497684), y = c(9113801.64775462, 9111328.49619133)), .Names = c("x", "y"))
pts = st_as_sf(data.frame(do.call(cbind, xy)), coords = c("x", "y"), crs = st_crs(x))
```
image(x, axes = TRUE)
plot(st_as_sfc(st_bbox(pts)), col = NA, add = TRUE)
bb = st_bbox(pts)
(xx = x[bb])
image(xx)
plot(st_as_sfc(bb), add = TRUE, col = NA)
image(x)
pt = st_point(c(x = 290462.103109179, y = 9114202.32594085))
buf = st_buffer(st_sfc(pt, crs = st_crs(x)), 1500)
plot(buf, add = TRUE)
buf = st_sfc(st_polygon(list(st_buffer(pt, 1500)[[1]], st_buffer(pt, 1000)[[1]])),
  crs = st_crs(x))
image(x[buf])
plot(buf, add = TRUE, col = NA)
image(x[buf, crop=FALSE])
plot(buf, add = TRUE, col = NA)
lc = read_stars(system.file("tif/lc.tif", package = "stars"))
x = c(orig = lc,
  flip_x = st_flip(lc, "x"),
  flip_y = st_flip(lc, "y"),
  flip_xy = st_flip(lc, c("x", "y")),
  along = 3)
plot(x)

---

**st_apply**

**st_apply** apply a function to one or more array dimensions

### Description

`st_apply` apply a function to array dimensions: aggregate over space, time, or something else

### Usage

```r
## S3 method for class 'stars'
st_apply(
  X,
  MARGIN,
  FUN,
  ..., 
  CLUSTER = NULL,
  PROGRESS = FALSE,
  FUTURE = FALSE,
  rename = TRUE
)
```

### Arguments

- **X** object of class stars
st_as_sf

MARGIN: see apply; index number(s) or name(s) of the dimensions over which FUN will be applied.

FUN: see apply;

...: arguments passed on to FUN;

CLUSTER: cluster to use for parallel apply; see makeCluster;

PROGRESS: logical; if TRUE, use pbapply::pbapply to show progress bar;

FUTURE: logical; if TRUE, use future.apply::future_apply;

rename: logical; if TRUE and X has only one attribute and FUN is a simple function name, rename the attribute of the returned object to the function name.

Value

Object of class stars with accordingly reduced number of dimensions; in case FUN returns more than one value, a new dimension is created carrying the name of the function used; see the examples.

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
st_apply(x, 1:2, mean) # mean band value for each pixel
st_apply(x, c("x", "y"), mean) # equivalent to the above
st_apply(x, 3, mean) # mean of all pixels for each band
st_apply(x, "band", mean) # equivalent to the above
st_apply(x, 1:2, range) # min and max band value for each pixel

# to get a progress bar also in non-interactive mode, specify:
if (require(pbapply)) {
  pboptions(type = "timer")
}

st_as_sf

Convert stars object into an sf object

Description

Convert stars object into an sf object.

Usage

## S3 method for class 'stars'
st_as_sfc(x, ..., as_points, which = seq_len(prod(dim(x)[1:2])))

## S3 method for class 'stars'
st_as_sf(
  x,
  ..., 
  as_points = FALSE,
merge = FALSE,
na.rm = TRUE,
use_integer = is.logical(x[[1]]) || is.integer(x[[1]]),
long = FALSE,
connect8 = FALSE)

Arguments

x object of class stars
...
ignored
as_points logical; should cells be converted to points or to polygons? See details.
which linear index of cells to keep (this argument is not recommended to be used)
merge logical; if TRUE, cells with identical values are merged (using GDAL_Polygonize or GDAL_FPolygonize); if FALSE, a polygon for each raster cell is returned; see details
na.rm logical; should missing valued cells be removed, or also be converted to features?
use_integer (relevant only if merge is TRUE): if TRUE, before polygonizing values are rounded to 32-bits signed integer values (GDALPolygonize), otherwise they are converted to 32-bit floating point values (GDALFPolygonize).
long logical; if TRUE, return a long table form sf, with geometries and other dimensions recycled
connect8 logical; if TRUE, use 8 connectedness. Otherwise the 4 connectedness algorithm will be applied.

Details

If merge is TRUE, only the first attribute is converted into an sf object. If na.rm is FALSE, areas with NA values are also written out as polygons. Note that the resulting polygons are typically invalid, and use st_make_valid to create valid polygons out of them.

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x = x[,,6] # a band with lower values in it
x[[1]][x[[1]] < 30] = NA # set lower values to NA
x[[1]] = x[[1]] < 100 # make the rest binary
x
(p = st_as_sf(x)) # removes NA areas
(p = st_as_sf(x[,,1], merge = TRUE)) # glues polygons together
all(st_is_valid(p)) # not all valid, see details
# plot(p, axes = TRUE)
(p = st_as_sf(x, na.rm = FALSE, merge = TRUE)) # includes polygons with NA values
# plot(p, axes = TRUE)
**st_as_stars**

*convert objects into a stars object*

**Description**

convert objects into a stars object

**Usage**

\[
st\_as\_stars(.x, \ldots)
\]

---

**S3 method for class 'list'**

\[
st\_as\_stars(.x, \ldots, dimensions = \text{NULL})
\]

---

**Default S3 method:**

\[
st\_as\_stars(.x = \text{NULL}, \ldots, raster = \text{NULL})
\]

---

**S3 method for class 'stars'**

\[
st\_as\_stars(.x, \ldots, curvilinear = \text{NULL}, crs = \text{st\_crs(4326)})
\]

---

**S3 method for class 'bbox'**

\[
st\_as\_stars(
  .x,
  \ldots,
  nx,
  ny,
  dx = dy,
  dy = dx,
  xlim = .x[\text{c("xmin", "xmax")}],
  ylim = .x[\text{c("ymin", "ymax")}],
  values = 0,
  n = 64800,
  pretty = \text{FALSE},
  inside = \text{FALSE},
  nz
  )
\]

---

**S3 method for class 'sf'**

\[
st\_as\_stars(.x, \ldots, name = \text{attr(.x, "sf\_column")})
\]

---

**S3 method for class 'Raster'**

\[
st\_as\_stars(.x, \ldots, att = 1, ignore\_file = \text{FALSE})
\]

---

**S3 method for class 'ncdfgeom'**

\[
st\_as\_stars(.x, \ldots, sf\_geometry = \text{NA})
\]

---

**S3 method for class 'stars\_proxy'**

---
st_as_stars(
  .x,
  ...,  
  downsample = 0,
  url = attr(.x, "url"),
  envir = parent.frame()
)

## S3 method for class 'xts'
st_as_stars(.x, ..., dimensions)

## S3 method for class 'OpenStreetMap'
st_as_stars(.x, ..., as_col = FALSE)

Arguments

.x object to convert
...
in case .x is of class bbox, arguments passed on to pretty
dimensions object of class dimensions
raster character; the names of the dimensions that denote raster dimensions
curvilinear only for creating curvilinear grids: named length 2 list holding longitude and
latitude matrices; the names of this list should correspond to raster dimensions
to be replaced

object of class crs with the coordinate reference system of the values in curvilinear;

nx integer; number of cells in x direction; see details
ny integer; number of cells in y direction; see details
dx numeric; cell size in x direction; see details
dy numeric; cell size in y direction; see details
xlim length 2 numeric vector with extent (min, max) in x direction
ylim length 2 numeric vector with extent (min, max) in y direction
values value(s) to populate the raster values with

the (approximate) target number of grid cells
pretty logical; should cell coordinates have pretty values?
inside logical; should all cells entirely fall inside the bbox, potentially not covering it
completely?

integer; number of cells in z direction; if missing no z-dimension is created.
name character; name for the geometry dimensions
att see factorValues; column in the RasterLayer’s attribute table
ignore_file logical; if TRUE, ignore the Raster object file name
sf_geometry sf data.frame with geometry and attributes to be added to stars object. Must have
same number of rows as timeseries instances.
st_contour

Compute or plot contour lines or sets

Description

Compute contour lines or sets

Usage

st_contour(
  x,
  na.rm = TRUE,
  contour_lines = FALSE,
  breaks = classInt::classIntervals(na.omit(as.vector(x[[1]])))$brks
)

Arguments

x    object of class stars
na.rm logical; should missing valued cells be removed, or also be converted to features?
contour_lines logical; if FALSE, polygons are returned (contour sets), otherwise contour lines
breaks numerical; values at which to "draw" contour levels
### st_coordinates

**Retrieve Coordinates for Raster or Vector Cube Cells**

#### Description

Retrieve coordinates for raster or vector cube cells.

#### Usage

```r
# S3 method for class 'stars'
st_coordinates(x, ..., add_max = FALSE, center = TRUE)
```

```r
# S3 method for class 'stars'
as.data.frame(x, ..., add_max = FALSE, center = NA)
```

```r
as_tibble.stars(.x, ..., add_max = FALSE, center = NA)
```

**Arguments**

- `x` object of class `stars`
- `...` ignored
- `add_max` logical; if TRUE, dimensions are given with a min (x) and max (x_max) value
- `center` logical; (only if `add_max` is FALSE): should grid cell center coordinates be returned (TRUE) or offset values (FALSE)? Center can be a named logical vector or list to specify values for each dimension.
- `.x` object to be converted to a tibble
**st_crop**  

*crop a stars object*

### Description

crop a stars object

### Usage

```r
## S3 method for class 'stars_proxy'
st_crop(
  x,
  y,
  ...,  
  crop = TRUE,
  epsilon = sqrt(.Machine$double.eps),
  collect = TRUE
)
```

```r
## S3 method for class 'stars'
st_crop(
  x,
  y,
  ...,  
  crop = TRUE,
  epsilon = sqrt(.Machine$double.eps),
  as_points = all(st_dimension(y) == 2, na.rm = TRUE)
)
```

### Arguments

- **x**: object of class stars
- **y**: object of class sf, sfc or bbox; see Details below.
- **...**: ignored
- **crop**: logical; if TRUE, the spatial extent of the returned object is cropped to still cover obj, if FALSE, the extent remains the same but cells outside y are given NA values.
- **epsilon**: numeric; factor to shrink the bounding box of y towards its center before cropping.
- **collect**: logical; if TRUE, repeat cropping on stars object, i.e. after data has been read
- **as_points**: logical; if FALSE, treat x as a set of points, else as a set of small polygons. Default: TRUE if y is two-dimensional, else FALSE

### Details

for raster x, st_crop selects cells for which the cell centre is inside the bounding box; see the examples below.
Examples

l7 = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))

d = st_dimensions(l7)

# area around cells 3:10 (x) and 4:11 (y):
offset = c(d["x"]$offset, d["y"]$offset)
res = c(d["x"]$delta, d["y"]$delta)
bb = st_bbox(c(xmin = offset[1] + 2 * res[1],
              ymin = offset[2] + 11 * res[2],
              xmax = offset[1] + 10 * res[1],
              ymax = offset[2] + 3 * res[2]), crs = st_crs(l7))
l7[bb]

plot(l7[1:13,1:13,1], reset = FALSE)
image(l7[bb,,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# slightly smaller bbox:
bb = st_bbox(c(xmin = offset[1] + 2.1 * res[1],
              ymin = offset[2] + 10.9 * res[2],
              xmax = offset[1] + 9.9 * res[1],
l7[bb]

plot(l7[1:13,1:13,1], reset = FALSE)
image(l7[bb,,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# slightly larger bbox:
bb = st_bbox(c(xmin = offset[1] + 1.9 * res[1],
              xmax = offset[1] + 10.1 * res[1],
              ymax = offset[2] + 2.9 * res[2]), crs = st_crs(l7))
l7[bb]

plot(l7[1:13,1:13,1], reset = FALSE)
image(l7[bb,,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# half a cell size larger bbox:
bb = st_bbox(c(xmin = offset[1] + 1.49 * res[1],
              ymin = offset[2] + 11.51 * res[2],
              xmax = offset[1] + 10.51 * res[1],
              ymax = offset[2] + 2.49 * res[2]), crs = st_crs(l7))
l7[bb]

plot(l7[1:13,1:13,1], reset = FALSE)
image(l7[bb,,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)
**st_dimensions**  

*get dimensions from stars object*

### Description

get dimensions from stars object

### Usage

```
st_dimensions(.x, ...)  
## S3 method for class 'stars'
st_dimensions(.x, ...)

st_dimensions(x) <- value  
## S3 replacement method for class 'stars'
st_dimensions(x) <- value

## S3 replacement method for class 'list'
st_dimensions(x) <- value

## S3 method for class 'array'
st_dimensions(.x, ...)

## Default S3 method:
st_dimensions(
  .x,
  ...,  
raster,
  affine = c(0, 0),
  cell_midpoints = FALSE,
  point = FALSE
)
```

```
st_set_dimensions(
  .x,
  which,
  values = NULL,
  point = NULL,
  names = NULL,
  xy,
  ...
)
```

```
st_get_dimension_values(.x, which, ..., max = FALSE, center = NA)
```
Arguments

- .x: object to retrieve dimensions information from
- ...: further arguments
- x: object of class dimensions
- value: new object of class dimensions, with matching dimensions
- .raster: length 2 character array with names (if any) of the raster dimensions
- affine: numeric; specify parameters of the affine transformation
- cell_midpoints: logical; if TRUE AND the dimension values are strictly regular, the values are interpreted as the cell midpoint values rather than the cell offset values when calculating offset (i.e., the half-cell-size correction is applied); can have a value for each dimension, or else is recycled
- point: logical; does the pixel value (measure) refer to a point (location) value or to an area (sum) value?
- which: integer or character; index or name of the dimension to be changed
- values: values for this dimension (e.g. sfc list-column), or length-1 dimensions object
- names: character; new names vector for (all) dimensions, ignoring which
- xy: length-2 character vector; (new) names for the x and y raster dimensions
- max: logical; if TRUE return the end, rather than the beginning of an interval
- center: logical; if TRUE return the center of an interval; if NA return the center for raster dimensions, and the start of intervals in other cases

Details
dimensions can be specified in two ways. The simplest is to pass a vector with numeric values for a numeric dimension, or character values for a categorical dimension. Parameter cell_midpoints is used to specify whether numeric values refer to the offset (start) of a dimension interval (default), or to the center; the center case is only available for regular dimensions. For rectilinear numeric dimensions, one can specify either a vector with cell borders (start values), or a data.frame with two columns named "start" and "end", with the respective interval start and end values. In the first case, the end values are computed from the start values by assuming the last two intervals have equal width.

Value

the dimensions attribute of x, of class dimensions

Examples

```r
x = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))
# Landsat 7 ETM+ band semantics: https://landsat.gsfc.nasa.gov/the-enhanced-thematic-mapper-plus/
# set bands to values 1,2,3,4,5,7:
(x1 = st_set_dimensions(x, "band", values = c(1,2,3,4,5,7), names = "band_number", point = TRUE))
# set band values as bandwidth
rbind(c(0.45,0.515), c(0.525,0.605), c(0.63,0.69), c(0.775,0.90), c(1.55,1.75), c(2.08,2.35)) %>%
  units::set_units("um") -> bw # or: units::set_units(bw)
```
# set bandwidth midpoint:
(x2 = st_set_dimensions(x, "band", values = 0.5 * (bw[,1]+bw[,2]),
    names = "bandwidth_midpoint", point = TRUE))

# set bandwidth intervals:
(x3 = st_set_dimensions(x, "band", values = make_intervals(bw), names = "bandwidth"))

---

**st_dim_to_attr**

create an array with dimension values

**Description**

create an array with dimension values

**Usage**

```
st_dim_to_attr(x, which = seq_along(dim(x)))
```

**Arguments**

- **x** object of class `stars`
- **which** integer; indices of the dimensions to address (default: all)

**Value**

`stars` object with dimension values as attributes

**Examples**

```
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
(x = st_dim_to_attr(x1))
plot(x)
(x = st_dim_to_attr(x1, 2:3))
plot(x)
(x= st_dim_to_attr(x1, 3))
plot(x)
```
**st_extract**  
*Extract cell values at point locations*

**Description**
Extract cell values at point locations, possibly using interpolation

**Usage**

```r
st_extract(x, ...)
```

```r
## S3 method for class 'stars'
st_extract(x, ...)
```

```r
## S3 method for class 'stars_proxy'
st_extract(x, pts, ..., method = "near", cellsize = 1e-07, debug = FALSE)
```

**Arguments**
- `x`: object of class `stars` or `stars_proxy`
- `...`: passed on to next method
- `pts`: object of class `sf` or `sfc` with POINT geometries
- `method`: interpolation method, see `st_warp`
- `cellsize`: numeric; cellsize chosen for the sampling cell.
- `debug`: logical; if TRUE, do not remove the destination grid file and print its name;

**Value**
- if `x` has more dimensions than only x and y (raster), an object of class `stars` with POINT geometries replacing x and y raster dimensions; otherwise an object of `sf`.

---

**st_intersects.stars**  
*spatial intersect predicate for stars and sfc object*

**Description**
Spatial intersect predicate for stars and sfc object

**Usage**

```r
## S3 method for class 'stars'
st_intersects(x, y, sparse = TRUE, ..., as_points = NA, transpose = FALSE)
```

```r
## S3 method for class 'stars_proxy'
st_intersects(x, y, sparse = TRUE, ..., as_points = NA, transpose = FALSE)
```

**Arguments**
- `x`, `y`: objects of class `stars` or `stars_proxy`
- `...`: passed on to next method
- `sparse`: logical; if TRUE, return a sparse grid with all the 'true' values.
- `as_points`: logical; if TRUE, return a point feature for each intersecting feature.
- `transpose`: logical; if TRUE, transpose the output.

**Value**
- An object of class `stars` or `sf` with POINT geometries.
Arguments

x object of class stars

y object that has an 'st_geometry' method: of class 'sf' or 'sfc', or 'stars' object with an 'sfc' dimension

sparse logical; if TRUE, return the a sparse logical matrix (object of class 'sgbp'), if FALSE, return a logical matrix

... ignored, or passed on to 'st_intersects.sf' for curvilinear grids

as_points logical, should grid cells be considered as points (TRUE) or polygons (FALSE)? Default: FALSE and warning emitted

transpose logical; should the transpose of the 'sgbp' object be returned?

Details

curvilinear grids are always converted to polygons, so points on grid boundaries may intersect with two cells touched; for other grids each cell boundary or corner belongs only to one cell.

Value

'sgbp' object if sparse = TRUE, logical matrix otherwise

---

st_join.stars Spatially join a stars and an 'sf' object

Description

Spatially join a stars and an 'sf' object

Usage

## S3 method for class 'stars'
st_join(
  x,
  y,
  join = st_intersects,
  ..., 
  what = "left1",
  as_points = NA,
  warn = TRUE
)
Arguments

- **x**: object of class `stars`
- **y**: object of class `sf`, or one that can be coerced into that by `st_as_sf`
- **join**: the join function, which should return an sgbp object; see details
- **...**: arguments that will be passed on to the join function
- **what**: "left1", "right" or "inner"; see details
- **as_points**: logical; controls whether grid cells in x will be treated as points, or as cell areas; the `st_intersects.stars` method by default will derive this from x’s metadata, or else assume areas.
- **warn**: logical; if TRUE, warn on 1-to-many matches when what is "left1"

Details

When there is more than one match to a single x value, the first matching record from y is taken (and if warn is TRUE a warning is raised). If what is "inner", an object of class sf with all matching records of x and y.

Value

If what is "left1", an object of class `stars` with the (first) value of y at spatial instances of x

---

**st_mosaic**

**build mosaic (composite) of several spatially disjoint stars objects**

Description

build mosaic (composite) of several spatially disjoint stars objects

Usage

```r
st_mosaic(.x, ...)  
```

```
## S3 method for class 'stars'
st_mosaic(
  .x,
  ...,  
  dst = tempfile(fileext = file_ext),
  options = c("-vrtnodata", "-9999", "-srcnodata", "nan"),
  file_ext = ".tif"
)
```

```r
## S3 method for class 'character'
st_mosaic(
  .x,
  ...,  
)  
```
### st_mosaic

```r

dst = tempfile(fileext = file_ext),
options = c("-vrtnodata", "-9999"),
file_ext = ".tif"
)

## S3 method for class 'stars_proxy'
st_mosaic(
  .x,
  ...,
  dst = tempfile(fileext = file_ext),
options = c("-vrtnodata", "-9999"),
file_ext = ".tif"
)
```

#### Arguments

- `.x` object of class `stars`, or character vector with input dataset names
- `...` further input `stars` objects
- `dst` character; destination file name
- `options` character; options to the `gdalbuildvrt` command
- `file_ext` character; file extension, determining the format used to write to (`".tif"` implies GeoTIFF)

#### Details

The `gdal` function `buildvrt` builds a mosaic of input images; these input images can be multi-band, but not higher-dimensional data cubes or `stars` objects with multiple attributes.

Uses `gdal_utils` to internally call `buildvrt`; no executables external to R are called.

#### Value

The `stars` method returns a `stars` object with the composite of the input; the `character` method returns the file name of the file with the mosaic; see also the GDAL documentation of `gdalbuildvrt`.

#### Examples

```r
x = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))
x1 = x[,100:200,100:200,]
x2 = x[,150:300,150:300,]
plot(st_mosaic(x1, x2))
```
st_rasterize  

rasterize simple feature geometries

Description
rasterize simple feature geometries

Usage

```r
st_rasterize(
  sf,
  template = st_as_stars(st_bbox(sf), values = NA_real_, ...),
  file = tempfile(),
  driver = "GTiff",
  options = character(0),
  ...
)
```

Arguments

- **sf**  
  object of class `sf`
- **template**  
  stars object with desired target geometry
- **file**  
  temporary file name
- **driver**  
  driver for temporary file
- **options**  
  character; options vector for GDALRasterize
- **...**  
  arguments passed on to `st_as_stars`

Examples

demo(nc, echo = FALSE, ask = FALSE)
(x = st_rasterize(nc))  # default grid:
plot(x, axes = TRUE)
# a bit more customized grid:
(x = st_rasterize(nc, st_as_stars(st_bbox(nc), nx = 100, ny = 50, values = NA_real_)))
plot(x, axes = TRUE)
(ls = st_sf(a = 1:2, st_sfc(st_linestring(rbind(c(0.1, 0), c(1.1, 1))),
  st_linestring(rbind(c(0, 0.05), c(1, 0.05))))))
(grd = st_as_stars(st_bbox(ls), nx = 10, ny = 10, xlim = c(0, 1), ylim = c(0, 1),
  values = NA_real_))
# Only the left-top corner is part of the grid cell:
sf_extSoftVersion()["GDAL"]
plot(st_rasterize(ls, grd), axes = TRUE, reset = FALSE)  # ALL_TOUCHED=FALSE;
plot(ls, add = TRUE, col = "red")
plot(st_rasterize(ls, grd, options = "ALL_TOUCHED=TRUE"), axes = TRUE, reset = FALSE)
plot(ls, add = TRUE, col = "red")
# add lines to existing 0 values, summing values in case of multiple lines:
(grd = st_as_stars(st_bbox(ls), nx = 10, ny = 10, xlim = c(0, 1), ylim = c(0, 1), values = 0))
st_raster_type

get the raster type (if any) of a stars object

Description
get the raster type (if any) of a stars object

Usage
st_raster_type(x)

Arguments
x object of class stars

Value
one of NA (if the object does not have raster dimensions), "curvilinear", "rectilinear", "affine", or "regular"

Examples
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
st_raster_type(x)

st_rgb
reduce dimension to rgb (alpha) hex values

Description
reduce dimension to rgb (alpha) hex values

Usage
st_rgb(x, dimension = 3, use_alpha = FALSE, maxColorValue = 255)

Arguments
x object of class stars
dimension dimension name or number to reduce
use_alpha logical; if TRUE, the fourth band will be used as alpha values
maxColorValue integer; maximum value for colors
Details

the dimension’s bands are mapped to red, green, blue, alpha; if a different ordering is wanted, use 
stars to reorder a dimension, see examples

See Also

st_apply, rgb

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
st_rgb(x, 3)
r = st_rgb(x[,,,c(6,5,4,3)], 3, use_alpha=TRUE) # now R=6,G=5,B=4,alpha=3
if (require(ggplot2)) {
  ggplot() + geom_stars(data = r) + scale_fill_identity()
}

st_sfc2xy

replace POINT simple feature geometry list with an x y raster

Description

replace POINT simple feature geometry list with an x y raster

Usage

st_sfc2xy(x, ...)

Arguments

x object of class stars, or of class sf
... passed on to as.data.frame.stars

Value

object of class stars with a POINT list replaced by x and y raster dimensions. This only works when the points are distributed over a regular or rectilinear grid.
Description

transform features, or warp/resample grids in stars objects to a new coordinate reference system

Usage

## S3 method for class 'stars'
st_transform(x, crs, ...)

## S3 method for class 'stars'
st_transform_proj(x, crs, ...)

Arguments

x object of class stars, with either raster or simple feature geometries

crs object of class crs with target crs

... ignored

Details

For simple feature dimensions, st_transform is called, leading to lossless transformation. For grid-ded spatial data, a curvilinear grid with transformed grid cell (centers) is returned. To convert this to a regular grid in the new CRS, use st_warp.

See Also

st_warp

Examples

gematrix = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(geomatrix))
new = st_crs(4326)
y = st_transform(x, new)
plot(st_transform(st_as_sfc(st_bbox(x)), new), col = NA, border = 'red')
plot(st_as_sfc(y, as_points=FALSE), col = NA, border = 'green', axes = TRUE, add = TRUE)
image(y, col = heat.colors(12), add = TRUE)
plot(st_as_sfc(y, as_points=TRUE), pch=3, cex=.5, col = 'blue', add = TRUE)
plot(st_transform(st_as_sfc(x, as_points=FALSE), new), add = TRUE)
st_warp  

Warp (resample) grids in stars objects to a new grid, possibly in a new coordinate reference system

Description
Warp (resample) grids in stars objects to a new grid, possibly in a new coordinate reference system

Usage
st_warp(
  src,
  dest,
  ...,  
  crs = NA_crs_,
  cellsize = NA_real_,
  segments = 100,
  use_gdal = FALSE,
  options = character(0),
  no_data_value = NA_real_,
  debug = FALSE,
  method = "near"
)

Arguments
src       object of class stars with source raster
dest      object of class stars with target raster geometry
...       ignored


crs       coordinate reference system for destination grid, only used when dest is missing
cellsize  length 1 or 2 numeric; cellsize in target coordinate reference system units
segments  (total) number of segments for segmentizing the bounding box before transform


use_gdal  logical; if TRUE, use gdalwarp, through gdal_utils
options   character vector with options, passed on to gdalwarp
no_data_value  value used by gdalwarp for no_data (NA) when writing to temporary file
debug     logical; if TRUE, do not remove the temporary gdalwarp destination file, and print its name

 method    character; see details for options; methods other than near only work when use_gdal=TRUE
st_xy2sfc

Details

method should be one of near, bilinear, cubic, cubicspline, lanczos, average, mode, max, min, med, q1 or q3; see https://github.com/r-spatial/stars/issues/109

For gridded spatial data (dimensions x and y), see figure; the existing grid is transformed into a regular grid defined by dest, possibly in a new coordinate reference system. If dest is not specified, but crs is, the procedure used to choose a target grid is similar to that of projectRaster (currently only with method='ngb'). This entails: (i) the envelope (bounding box polygon) is transformed into the new crs, possibly after segmentation (red box); (ii) a grid is formed in this new crs, touching the transformed envelope on its East and North side, with (if cellsize is not given) a cellsize similar to the cell size of src, with an extent that at least covers x; (iii) for each cell center of this new grid, the matching grid cell of x is used; if there is no match, an NA value is used.

Examples

geomatrix = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(geomatrix))
new_crs = st_crs(4326)
y = st_warp(x, crs = new_crs)
plot(st_transform(st_as_sfc(st_bbox(x)), new_crs), col = NA, border = 'red')
plot(st_as_sfc(y, as_points=FALSE), col = NA, border = 'green', axes = TRUE, add = TRUE)
image(y, add = TRUE, nb breaks = 6)
plot(st_as_sfc(y, as_points=TRUE), pch=3, cex=.5, col = 'blue', add = TRUE)
plot(st_transform(st_as_sfc(x, as_points=FALSE), new_crs), add = TRUE)

# warp 0-360 raster to -180-180 raster:
r = read_stars(system.file("nc/reduced.nc", package = "stars"))
r %>% st_set_crs(4326) %>% st_warp(st_as_stars(st_bbox(), dx = 2)) -> s
plot(r, axes = TRUE) # no CRS set, so no degree symbols in labels
plot(s, axes = TRUE)

---

st_xy2sfc

replace x y raster dimensions with simple feature geometry list (points, or polygons = rasterize)

Description

replace x y raster dimensions with simple feature geometry list (points, or polygons = rasterize)

Usage

st_xy2sfc(x, as_points, ..., na.rm = TRUE)

Arguments

x object of class stars
as_points logical; if TRUE, generate points at cell centers, else generate polygons
... arguments passed on to st_as_sfc
na.rm logical; omit (remove) cells which are entirely missing valued (across other dimensions)
Value

object of class stars with x and y raster dimensions replaced by a single sfc geometry list column containing either points, or polygons. Adjacent cells with identical values are not merged; see st_rasterize for this.

Description

write stars object to gdal dataset (typically: to file)

Usage

write_stars(obj, dsn, layer, ...)

## S3 method for class 'stars'
write_stars(
  obj,
  dsn,
  layer = 1,
  ...,
  driver = detect.driver(dsn),
  options = character(0),
  type = "Float32",
  NA_value = NA_real_,
  update = FALSE,
  normalize_path = TRUE
)

## S3 method for class 'stars_proxy'
write_stars(
  obj,
  dsn,
  layer = 1,
  ...,
  driver = detect.driver(dsn),
  options = character(0),
  type = "Float32",
  NA_value = NA_real_,
  chunk_size = c(dim(obj)[1], floor(2.5e+07/dim(obj)[1])),
  progress = TRUE
)

detect.driver(filename)
Arguments

- **obj**: object of class `stars`
- **dsn**: gdal dataset (file) name
- **layer**: attribute name; if missing, the first attribute is written
- **type**: character; output binary type, one of: `Byte` for eight bit unsigned integer, `UInt16` for sixteen bit unsigned integer, `Int16` for sixteen bit signed integer, `UInt32` for thirty two bit unsigned integer, `Int32` for thirty two bit signed integer, `Float32` for thirty two bit floating point, `Float64` for sixty four bit floating point.
- **NA_value**: non-NA value that should represent R’s NA value in the target raster file; if set to NA, it will be ignored.
- **update**: logical; if TRUE, an existing file is being updated
- **normalize_path**: logical; see `read_stars`
- **chunk_size**: length two integer vector with the number of pixels (x, y) used in the read/write loop; see details.
- **progress**: logical; if TRUE, a progress bar is shown
- **filename**: character; used for guessing driver short name based on file extension; see examples

Details

`write_stars` first creates the target file, then updates it sequentially by writing blocks of `chunk_size`.

Examples

```r
detect.driver("L7_ETMs.tif")
```
Index

[.stars, 38
[.stars (stars_subset), 18
[<-.stars (stars_subset), 18

aes, 9
aggregate.stars, 3
apply, 21
as, 4
as.data.frame.stars, 38
as.data.frame.stars (st_coordinates), 26
as.tbl_cube.stars (dplyr), 7
as_tibble.stars (st_coordinates), 26
c.stars, 5, 16
c.stars_proxy (c.stars), 5
classIntervals, 12
coerce, stars, Raster-method (as), 4
coerce, stars_proxy, Raster-method (as), 4
contour, 6, 26
contour.stars, 6
coord_equal, 9
covrage_fraction, 3
cut, 7
cut.array (cut_stars), 6
cut.matrix (cut_stars), 6
cut.POSIXt, 3
cut.stars (cut_stars), 6
cut_stars, 6
detect.driver (write_stars), 42
dplyr, 7

dfacet_wrap, 9
dfactorValues, 24
dfilter, 8
dfilter.stars (dplyr), 7
dfilter.stars_proxy (dplyr), 7
dfindInterval, 3
gdal_utils, 35, 40
gdal_write, 43
gem_raster, 9
gem_sf, 9
gem_stars, 8
gem_tile, 9
image.stars (plot), 11
is_big (read_stars), 15
make_intervals, 9
makeCluster, 21
Math.stars (ops_stars), 10
Math.stars_proxy (ops_stars), 10
mutate.stars (dplyr), 7
mutate.stars_proxy (dplyr), 7
normalizePath, 16
Ops.stars (ops_stars), 10
Ops.stars_proxy (ops_stars), 10
ops_stars, 10
plot, 11
pretty, 24
projectRaster, 41
pull, 8
pull.stars (dplyr), 7
pull.stars_proxy (dplyr), 7
rasterImage, 13
read_ncdf, 14
read_stars, 5, 15, 16, 43
redimension, 18
rgb, 13, 38
select.stars (dplyr), 7
select.stars_proxy (dplyr), 7
slice.stars (dplyr), 7
slice.stars_proxy (dplyr), 7
st_apply, 20, 38
st_as_sf, 3, 13, 21, 26, 34
st_as_sfc.stars (st_as_sf), 21
st_as_stars, 16, 23, 36
st_contour, 6, 25
st_coordinates, 26
st_crop, 19, 27
st_dim_to_attr, 31
st_dimensions, 29
st_dimensions <- (st_dimensions), 29
st_drivers, 43
st_extract, 32
st_flip(stars_subset), 18
st_get_dimension_values
  (st_dimensions), 29
st_intersects.stars, 32, 34
st_join.stars, 33
st_make_valid, 22
st_mosaic, 34
st_raster_type, 37
st_rasterize, 36
st_redimension(redimension), 18
st_rgb, 37
st_set_dimensions(st_dimensions), 29
st_sfc2xy, 38
st_transform, 39, 39
st_transform_proj.stars(st_transform), 39
st_warp, 32, 39, 40
st_xy2sfc, 41
stars_subset, 18

theme_stars(geom_stars), 8

var.get.nc, 15

write_stars, 42