Package ‘terra’

December 12, 2024

Type Package

Title Spatial Data Analysis

Version 1.8-5

Date 2024-12-11

Depends R (>= 3.5.0)

Suggests parallel, tinytest, ncdf4, sf (>= 0.9-8), deldir, XML, leaflet (>= 2.2.1), htmlwidgets

LinkingTo Rcpp

Imports methods, Rcpp (>= 1.0-10)

SystemRequirements C++17, GDAL (>= 2.2.3), GEOS (>= 3.4.0), PROJ (>= 4.9.3), sqlite3

Encoding UTF-8

Language en-US

Maintainer Robert J. Hijmans <r.hijmans@gmail.com>

Description
Methods for spatial data analysis with vector (points, lines, polygons) and raster (grid) data. Methods for vector data include geometric operations such as intersect and buffer. Raster methods include local, focal, global, zonal and geometric operations. The predict and interpolate methods facilitate the use of regression type (interpolation, machine learning) models for spatial prediction, including with satellite remote sensing data. Processing of very large files is supported. See the manual and tutorials on <https://rspatial.org/> to get started. 'terra' replaces the 'raster' package ('terra' can do more, and it is faster and easier to use).

License GPL (>= 3)

URL https://rspatial.org/, https://rspatial.github.io/terra/

BugReports https://github.com/rspatial/terra/issues/

LazyLoad yes

NeedsCompilation yes

Author Robert J. Hijmans [cre, aut] (<https://orcid.org/0000-0001-5872-2872>),
Roger Bivand [ctb] (<https://orcid.org/0000-0003-2392-6140>),
Emanuele Cordano [ctb] (<https://orcid.org/0000-0002-3508-5898>),

Krzysztof Dyba [ctb] (<https://orcid.org/0000-0002-8614-3816>),
Edzer Pebesma [ctb] (<https://orcid.org/0000-0001-8049-7069>),
Michael D. Sumner [ctb]

Repository CRAN

Date/Publication 2024-12-12 10:00:02 UTC

Contents

`terra-package` *Description of the methods in the terra package*

Description

`terra` provides methods to manipulate geographic (spatial) data in "raster" and "vector" form. Raster data divide space into rectangular grid cells and they are commonly used to represent spatially continuous phenomena, such as elevation or the weather. Satellite images also have this data structure, and in that context grid cells are often referred to as pixels. In contrast, "vector" spatial data (points, lines, polygons) are typically used to represent discrete spatial entities, such as a road, country, or bus stop.

The package implements two main classes (data types): `SpatRaster` and `SpatVector`. `SpatRaster` supports handling large raster files that cannot be loaded into memory; local, focal, zonal, and global raster operations; polygon, line and point to raster conversion; integration with modeling methods to make spatial predictions; and more. `SpatVector` supports all types of geometric operations such as intersections.

Additional classes include SpatExtent, which is used to define a spatial extent (bounding box); SpatRasterDataset, which represents a collection of sub-datasets for the same area. Each sub-dataset is a SpatRaster with possibly many layers, and may, for example, represent different weather variables; and SpatRasterCollection and SpatVectorCollection that are equivalent to lists of SpatRaster or SpatVector objects. There is also a SpatGraticule class to assist in adding a longitude/latitude lines and labels to a map with another coordinate reference system.

These classes hold a C++ pointer to the data "reference class" and that creates some limitations. They cannot be recovered from a saved R session either or directly passed to nodes on a computer cluster. Generally, you should use writeRaster to save SpatRaster objects to disk (and pass a filename or cell values to cluster nodes). Also see wrap. Also, package developers should not directly access this pointer, as its user-interface is not stable.

The terra package is conceived as a replacement of the raster package. terra has a very similar, but simpler, interface, and it is faster than raster. At the bottom of this page there is a table that shows differences in the methods between the two packages.

Below is a list of some of the most important methods grouped by theme.

SpatRaster

I. Creating, combining and sub-setting

rast	Create a SpatRaster from scratch, file, or another object
c	Combine SpatRasters (multiple layers)
add<-	Add a SpatRaster to another one
subset or [[, or $	Select layers of a SpatRaster
selectRange	Select cell values from different layers using an index layer

II. Changing the spatial extent or resolution

Also see the methods in section VIII

merge	Combine SpatRasters with different extents (but same origin and resolution)
mosaic	Combine SpatRasters with different extents using a function for overlapping cells
crop	Select a geographic subset of a SpatRaster
extend	Add rows and/or columns to a SpatRaster
trim	Trim a SpatRaster by removing exterior rows and/or columns that only have NAs
aggregate	Combine cells of a SpatRaster to create larger cells
disagg	Subdivide cells
resample	Resample (warp) values to a SpatRaster with a different origin and/or resolution
project	Project (warp) values to a SpatRaster with a different coordinate reference system
shift	Adjust the location of SpatRaster

flip	Flip values horizontally or vertically
rotate	Rotate values around the date-line (for lon/lat data)
t	Transpose a SpatRaster

III. Local (cell based) methods

Apply-like methods:

app	Apply a function to all cells, across layers, typically to summarize (as in base::apply)
tapp	Apply a function to groups of layers (as in base::tapply and stats::aggregate)
lapp	Apply a function to using the layers of a SpatRaster as variables
sapp	Apply a function to each layer
rapp	Apply a function to a spatially variable range of layers

Arithmetic, logical, and standard math methods:

Arith-methods	Standard arithmetic methods (+, -, *, ^, %%, %/%, /)
Compare-methods	Comparison methods for SpatRaster (==, !=, >, <, <=, >=m is.na, is.finite)
not.na	a one-step equivalent to !is.na
Summary-methods	mean, max, min, median, sum, range, prod, any, all, stdev, which.min, which.max, anyNA, noNA, allNA
Logic-methods	Boolean methods (!, &, |)
Math-methods	abs, sign, sqrt, ceiling, floor, trunc, cummax, cummin, cumprod, cumsum, log, log10, log2, log1p, acos, acosh, asin, asinh, atan, atanh, exp, expm1, cos, cosh, sin, sinh, tan, tanh, round, signif
as.bool	create a Boolean (logical) SpatRaster
as.int	create an integer (whole numbers) SpatRaster

Other methods:

approximate	Compute missing values for cells by interpolation across layers
roll	Rolling functions such as the rolling mean
clamp	Restrict cell values to a minimum and/or maximum value
cellSize	Compute the area of cells
classify	(Re-)classify values
subst	Substitute (replace) cell values
cover	First layer covers second layer except where the first layer is NA
init	Initialize cells with new values
mask	Replace values in a SpatRaster based on values in another SpatRaster
which.lyr	which is the first layer that is TRUE?
segregate	Make a 0/1 layer for each unique value
rangeFill	Make a 0/1 SpatRaster for a time series
regress	Cell-based regression models

IV. Zonal and global methods

expanse	Compute the summed area of cells
crosstab	Cross-tabulate two SpatRasters
freq	Frequency table of SpatRaster cell values
global	Summarize SpatRaster cell values with a function
quantile	Quantiles
layerCor	Correlation between layers
stretch	Stretch values
scale	Scale values
summary	Summary of the values of a SpatRaster (quartiles and mean)
unique	Get the unique values in a SpatRaster
zonal	Summarize a SpatRaster by zones in another SpatRaster

V. Situation (spatial context) based methods

adjacent	Identify cells that are adjacent to a set of cells of a SpatRaster
boundaries	Detection of boundaries (edges)
distance	Shortest distance to a cell that is not NA or to or from a vector object
gridDist	Shortest distance through adjacent grid cells
costDist	Shortest distance considering cell-varying friction
direction	Direction (azimuth) to or from cells that are not NA
focal	Focal (neighborhood; moving window) functions
focal3D	Three dimensional (row, col, lyr) focal functions
focalCpp	Faster focal by using custom C++ functions
focalReg	Regression between layers for focal areas
focalPairs	Apply a function (e.g. a correlation coefficient) to focal values for pairs of layers
patches	Find patches (clumps)
sieve	Sieve filter to remove small patches
terrain	Compute slope, aspect and other terrain characteristics from elevation data
viewshed	Compute viewshed (showing areas that are visible from a particular location
shade	Compute hill shade from slope and aspect layers
autocor	Compute global or local spatial autocorrelation

VI. Model predictions

predict	Predict a non-spatial (regression or classification) model to a SpatRaster
interpolate	Predict a spatial model to a SpatRaster
interpIDW	Inverse-distance-weighted interpolation
interpNear	Nearest neighbor interpolation
k_means	k-means clustering of SpatRaster data
princomp and prcomp	Principal Component Analysis (PCA) with raster data

VII. Accessing cell values

Apart from the function listed below, you can also use indexing with `[` with cell numbers, and row and/or column numbers

values	cell values (fails with very large rasters)
values<-	Set new values to the cells of a SpatRaster
setValues	Set new values to the cells of a SpatRaster
as.matrix	Get cell values as a matrix
as.array	Get cell values as an array
as.data.frame	get cell values as a data.frame (including class lables)
extract	Extract cell values from a SpatRaster (with cell numbers, coordinates, points, lines, or polygons)
extractAlong	Extract cell values along a line such that the values are in the right order
spatSample	Regular or random sample
minmax	Get the minimum and maximum value of the cells of a SpatRaster (if known)
setMinMax	Compute the minimum and maximum value of a SpatRaster if these are not known
extract	spatial queries of a SpatRaster with a SpatVector

VIII. Getting and setting dimensions

Get or set basic parameters of SpatRasters. If there are values associated with a SpatRaster (either in memory or via a link to a file) these are lost when you change the number of columns or rows or the resolution. This is not the case when the extent is changed (as the number of columns and rows will not be affected). Similarly, with **crs** you can set the coordinate reference system, but this does not transform the data (see project for that).

ncol	The number of columns
nrow	The number of rows
ncell	The number of cells (can not be set directly, only via ncol or nrow)
res	The resolution (x and y)
nlyr	Get or set the number of layers
names	Get or set the layer names

xres	The x resolution (can be set with res)
yres	The y resolution (can be set with res)
xmin	The minimum x coordinate (or longitude)
xmax	The maximum x coordinate (or longitude)
ymin	The minimum y coordinate (or latitude)
ymax	The maximum y coordinate (or latitude)
ext	Get or set the extent (minimum and maximum x and y coordinates ("bounding box")
origin	The origin of a SpatRaster
crs	The coordinate reference system (map projection)
is.lonlat	Test if an object has (or may have) a longitude/latitude coordinate reference system
sources	Get the filename(s) to which a SpatRaster is linked
inMemory	Are the data sources in memory (or on disk)?
toMemory	Force data sources to memory (not recommended)?
compareGeom	Compare the geometry of SpatRasters
NAflag	Set the NA value (for reading from a file with insufficient metadata)

IX. Computing row, column, cell numbers and coordinates

Cell numbers start at 1 in the upper-left corner. They increase within rows, from left to right, and then row by row from top to bottom. Likewise, row numbers start at 1 at the top of the raster, and column numbers start at 1 at the left side of the raster.

xFromCol	x-coordinates from column numbers
yFromRow	y-coordinates from row numbers
xFromCell	x-coordinates from row numbers
yFromCell	y-coordinates from cell numbers
xyFromCell	x and y coordinates from cell numbers
colFromX	Column numbers from x-coordinates (or longitude)
rowFromY	Row numbers from y-coordinates (or latitude)
rowColFromCell	Row and column numbers from cell numbers
cellFromXY	Cell numbers from x and y coordinates
cellFromRowCol	Cell numbers from row and column numbers
cellFromRowColCombine	Cell numbers from all combinations of row and column numbers
cells	Cell numbers from an SpatVector or SpatExtent

X. Time related methods

time	Get or set time
fillTime	can add empty layers in between existing layers to assure that the time step between layers is constan
mergeTime	combine multiple rasters, perhaps partly overlapping in time, into a single time series

XI. Methods for categorical rasters

`is.factor`	Are there categorical layers?
`levels`	Get active categories, or set categories
`activeCat`	Get or set the active category
`cats`	Get categories (active and inactive)
`set.cats`	Set categories in place
`concats`	Combine SpatRasters with different categories
`catalyze`	Create a layer for each category
`as.numeric`	use the active category to create a non-categorical SpatRaster
`as.factor`	Make the layers of a SpatRaster categorical

XII. Writing SpatRaster files

Basic:

`writeRaster`	Write all values of SpatRaster to disk. You can set the filetype, datatype, compression.
`writeCDF`	Write SpatRaster data to a netCDF file

Advanced:

`readStart`	Open file connections for efficient multi-chunk reading
`readValues`	Read some values from an opened file
`readStop`	Close file connections
`writeStart`	Open a file for writing
`writeValues`	Write some values to an opened file
`writeStop`	Close the file after writing
`blocks`	Get blocksize for reading files (when not writing)

XIII. Miscellaneous SpatRaster methods

`terraOptions`	Show, set, or get session options, mostly to control memory use and to set write options
`sources`	Show the data sources of a SpatRaster
`tmpFiles`	Show or remove temporary files
`mem_info`	memory needs and availability
`inMemory`	Are the cell values in memory?

XIV. SpatRasterDataset

A SpatRasterDataset contains SpatRasters that represent sub-datasets for the same area. They all have the same extent and resolution.

sds	Create a SpatRasterDataset from a file with subdatasets (ncdf or hdf) or from SpatRasters
[or $	Extract a SpatRaster
names	Get the names of the sub-datasets

XV. SpatRasterCollections

A SpatRasterCollection is a vector of SpatRaster objects. Unlike for a SpatRasterDataset, there the extent and resolution of the SpatRasters do not need to match each other.

sprc	create a SpatRasterCollection from (a list of) SpatRasters
length	how many SpatRasters does the SpatRasterCollection have?
crop	crop a SpatRasterCollection
impose	force the members of SpatRasterCollection to the same geometry
merge	merge the members of a SpatRasterCollection
mosaic	mosaic (merge with a function for overlapping areas) the members of a SpatRasterCollection
[	extract a SpatRaster

SpatVector

XVI. Create SpatVector objects

vect	Create a SpatVector from a file (for example a "shapefile") or from another object
vector_layers	list or delete layers in a vector database such as GPGK
rbind	append SpatVectors of the same geometry type
unique	remove duplicates
na.omit	remove empty geometries and/or fields that are NA
project	Project a SpatVector to a different coordinate reference system
writeVector	Write SpatVector data to disk
centroids	Get the centroids of a SpatVector
voronoi	Voronoi diagram
delaunay	Delaunay triangles
convHull	Compute the convex hull of a SpatVector
minRect	Compute the minimum minimal bounding rotated rectangle of a SpatVector
minCircle	Compute the minimal bounding circle of a SpatVector
fillHoles	Remove or extract holes from polygons

XVII. Properties of SpatVector objects

`geom`	returns the geometries as matrix or WKT
`crds`	returns the coordinates as a matrix
`linearUnits`	returns the linear units of the crs (in meter)
`ncol`	The number of columns (of the attributes)
`nrow`	The number of rows (of the geometries and attributes)
`names`	Get or set the layer names
`ext`	Get the extent (minimum and maximum x and y coordinates ("bounding box")
`crs`	The coordinate reference system (map projection)
`is.lonlat`	Test if an object has (or may have) a longitude/latitude coordinate reference system

XVIII. Geometric queries

`adjacent`	find adjacent polygons
`expanse`	computes the area covered by polygons
`nearby`	find nearby geometries
`nearest`	find the nearest geometries
`relate`	geometric relationships such as "intersects", "overlaps", and "touches"
`perim`	computes the length of the perimeter of polygons, and the length of lines

XIX. Geometric operations

`erase` or "-"	erase (parts of) geometries
`intersect` or "*"	intersect geometries
`union` or "+"	Merge geometries
`cover`	update polygons
`symdif`	symmetrical difference of two polygons
`aggregate`	dissolve smaller polygons into larger ones
`buffer`	buffer geometries
`disagg`	split multi-geometries into separate geometries
`crop`	clip geometries using a rectangle (SpatExtent) or SpatVector

XX. SpatVector attributes

We use the term "attributes" for the tabular data (data.frame) associated with vector geometries.

extract	spatial queries between SpatVector and SpatVector (e.g. point in polygons)
sel	select - interactively select geometries
click	identify attributes by clicking on a map
merge	Join a table with a SpatVector
as.data.frame	get attributes as a data.frame
as.list	get attributes as a list
values	Get the attributes of a SpatVector
values<-	Set new attributes to the geometries of a SpatRaster
sort	sort SpatVector by the values in a field

XXI. Change geometries (for display, experimentation)

shift	change the position geometries by shifting their coordinates in horizontal and/or vertical direction
spin	rotate geometries around an origin
rescale	shrink (or expand) geometries, for example to make an inset map
flip	flip geometries vertically or horizontally
t	transpose geometries (switch x and y)

XXII. Geometry properties and topology

width	the minimum diameter of the geometries
clearance	the minimum clearance of the geometries
sharedPaths	shared paths (arcs) between line or polygon geometries
simplifyGeom	simplify geometries
gaps	find gaps between polygon geometries
fillHoles	get or remove the polygon holes
makeNodes	create nodes on lines
mergeLines	connect lines to form polygons
removeDupNodes	remove duplicate nodes in geometries and optionally rounds the coordinates
is.valid	check if geometries are valid
makeValid	attempt to repair invalid geometries
snap	make boundaries of geometries identical if they are very close to each other
erase (single argument)	remove parts of geometries that overlap
union (single argument)	create new polygons such that there are no overlapping polygons
rotate	rotate to (dis-) connect them across the date-line
normalize.longitude	move geometries that are outside of the -180 to 180 degrees range.
elongate	make lines longer by extending both sides
combineGeoms	combine geometries that overlap, share a border, or are within a minimum distance of each othe

forceCCW	force counter-clockwise polygon winding

XXIII. SpatVectorCollections

A SpatVectorCollection is a vector of SpatVector objects.

svc	create a SpatVectorCollection from (a list of) SpatVector objects
length	how many SpatRasters does the SpatRasterCollection have?
[	extract a SpatVector

Other classes

XXIV. SpatExtent

ext	Create a SpatExtent object. For example to crop a Spatial dataset
intersect	Intersect two SpatExtent objects, same as -
union	Combine two SpatExtent objects, same as +
Math-methods	round/floor/ceiling of a SpatExtent
align	Align a SpatExtent with a SpatRaster
draw	Create a SpatExtent by drawing it on top of a map (plot)

XXV. SpatGraticule

graticule	Create a graticule
crop	crop a graticule
plot<SpatGraticule>	plot a graticule

General methods

XXVI. Conversion between spatial data objects from different packages

You can coerce SpatRasters to Raster* objects, after loading the `raster` package, with `as(object, "Raster")`, or `raster(object)` or `brick(object)` or `stack(object)`

`rast`	SpatRaster from matrix and other objects
`vect`	SpatVector from `sf` or `Spatial*` vector data
`sf::st_as_sf`	sf object from SpatVector
`rasterize`	Rasterizing points, lines or polygons
`rasterizeWin`	Rasterize points with a moving window
`rasterizeGeom`	Rasterize attributes of geometries such as "count", "area", or "length"
`as.points`	Create points from a SpatRaster or SpatVector
`as.lines`	Create lines from a SpatRaster or SpatVector
`as.polygons`	Create polygons from a SpatRaster
`as.contour`	Contour lines from a SpatRaster

XXVII. Plotting

Maps:

`plot`	Plot a SpatRaster or SpatVector. The main method to create a map
`panel`	Combine multiple plots
`points`	Add points to a map
`lines`	Add lines to a map
`polys`	Add polygons to a map
`text`	Add text (such as the values of a SpatRaster or SpatVector) to a map
`halo`	Add text with a halo to a map
`map.pal`	Color palettes for mapping
`image`	Alternative to plot to make a map with a SpatRaster
`plotRGB`	Combine three layers (red, green, blue channels) into a single "real color" plot
`plot<SpatGraticule>`	plot a graticule
`sbar`	Add a scale bar to a map
`north`	Add a north arrow to a map
`inset`	Add a small inset (overview) map
`add_legend`	Add a legend to a map
`add_box`	Add a bounding box to a map
`map_extent`	Get the coordinates of a map's axes positions
`dots`	Make a dot-density map
`cartogram`	Make a cartogram
`persp`	Perspective plot of a SpatRaster
`contour`	Contour plot or filled-contour plot of a SpatRaster
`colorize`	Combine three layers (red, green, blue channels) into a single layer with a color-table

Interacting with a map:

zoom	Zoom in to a part of a map by drawing a bounding box on it
click	Query values of SpatRaster or SpatVector by clicking on a map
sel	Select a spatial subset of a SpatRaster or SpatVector by drawing on a map
draw	Create a SpatExtent or SpatVector by drawing on a map

Other plots:

plot	x-y scatter plot of the values of (a sample of) the layers of two SpatRaster objects
hist	Histogram of SpatRaster values
barplot	Bar plot of a SpatRaster
density	Density plot of SpatRaster values
pairs	Pairs plot for layers in a SpatRaster
boxplot	Box plot of the values of a SpatRaster

Comparison with the raster package

XXVIII. New method names

`terra` has a single class `SpatRaster` for which `raster` has three (`RasterLayer`, `RasterStack`, `RasterBrick`). Likewise there is a single class for vector data `SpatVector` that replaces six `Spatial*` classes. Most method names are the same, but note the following important differences in methods names with the `raster` package

raster package	**terra package**
raster, brick, stack	rast
rasterFromXYZ	rast(, type="xyz")
stack, addLayer	c
addLayer	add<-
area	cellSize or expanse
approxNA	approximate
calc	app
cellFromLine, cellFromPolygon,	cells
cellsFromExtent	cells
cellStats	global
clump	patches
compareRaster	compareGeom
corLocal	focalPairs
coordinates	crds
couldBeLonLat	is.lonlat
disaggregate	disagg
distanceFromPoints	distance
drawExtent, drawPoly, drawLine	draw
dropLayer	subset

extent	ext
getValues	values
isLonLat, isGlobalLonLat	is.lonlat
layerize	segregate
layerStats	layerCor
movingFun	roll
NAvalue	NAflag
nlayers	nlyr
overlay	lapp
unstack	as.list
projectRaster	project
rasterToPoints	as.points
rasterToPolygons	as.polygons
readAll	toMemory
reclassify, subs, cut	classify
sampleRandom, sampleRegular	spatSample
shapefile	vect
stackApply	tapp
stackSelect	selectRange

XXIX. Changed behavior

Also note that even if function names are the same in `terra` and `raster`, their output can be different. In most cases this was done to get more consistency in the returned values (and thus fewer errors in the downstream code that uses them). In other cases it simply seemed better. Here are some examples:

as.polygons	By default, `terra` returns dissolved polygons
quantile	computes by cell, across layers instead of the other way around
extract	By default, `terra` returns a matrix, with the first column the sequential ID of the vectors. `raster` returns a list (for lines or polygons) or a matrix (for points, but without the ID column. You can use `list=TRUE` to get the results as a list
values	`terra` always returns a matrix. `raster` returns a vector for a `RasterLayer`
Summary-methods	With `raster`, `mean(x, y)` and `mean(stack(x, y)` return the same result, a single layer with the mean of all cell values. This is also what `terra` returns with `mean(c(x, y))`, but with `mean(x, y)` the parallel mean is returned – that is, the computation is done layer-wise, and the number of layers in the output is the same as that of `x` and `y` (or the larger of the two if they are not the same). This affects all summary functions (`sum`, `mean`, `median`, `which.min`, `which.max`, `min`, `max`, `prod`, `any`, `all`, `stdev`), except `range`, which is not implemented for this case (you can use `min` and `max` instead)

Authors

Except where indicated otherwise, the methods and functions in this package were written by Robert Hijmans. The configuration scripts were written by Roger Bivand. Some of code using the GEOS library was adapted from code by Edzer Pebesma for `sf`. Michael Sumner contributed various bits and pieces.

Acknowledgments

This package is an attempt to climb on the shoulders of giants (GDAL, PROJ, GEOS, NCDF, GeographicLib, Rcpp, R). Many people have contributed by asking questions or raising issues. Feedback and suggestions by Márcia Barbosa, Kendon Bell, Andrew Gene Brown, Jean-Luc Dupouey, Krzysztof Dyba, Sarah Endicott, Derek Friend, Alex Ilich, Gerald Nelson, Jakub Nowosad, and Monika Tomaszewska have been especially helpful.

`activeCat` *Active category*

Description

Get or set the active category of a multi-categorical SpatRaster layer

Usage

```
## S4 method for signature 'SpatRaster'
activeCat(x, layer=1)
## S4 replacement method for signature 'SpatRaster'
activeCat(x, layer=1)<-value
```

Arguments

`x`	SpatRaster
`layer`	positive integer, the layer number or name
`value`	positive integer or character, indicating which column in the categories to use. Note that when a number is used this index is zero based, and "1" refers to the second column. This is because the first column of the categories has the cell values, not categorical labels

Value

integer

See Also

`levels`, `cats`

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE) + 10
d <- data.frame(id=11:13, cover=c("forest", "water", "urban"), letters=letters[1:3], value=10:12)
levels(r) <- d

activeCat(r)
activeCat(r) <- 3
activeCat(r)
```

---

add — *Add (in place) a SpatRaster to another SpatRaster or to a SpatRasterDataset or SpatRasterCollection*

Description

Add (in place) a SpatRaster to another SpatRaster. Comparable with `c`, but without copying the object.

Usage

```
## S4 replacement method for signature 'SpatRaster,SpatRaster'
add(x)<-value

## S4 replacement method for signature 'SpatRasterDataset,SpatRaster'
add(x)<-value

## S4 replacement method for signature 'SpatRasterCollection,SpatRaster'
add(x)<-value
```

Arguments

`x`	SpatRaster, SpatRasterDataset or SpatRasterCollection
`value`	SpatRaster

Value

SpatRaster

See Also

`c`

Examples

```
r <- rast(nrows=5, ncols=9, vals=1:45)
x <- c(r, r*2)
add(x) <- r*3
x
```

---

add_box — *draw a box*

Description

Similar to box allowing adding a box around a map. This function will place the box around the mapped area.

Usage

```
add_box(...)
```

Arguments

`...`	arguments passed to lines

See Also

add_legend, add_grid, add_mtext

Examples

```
v <- vect(system.file("ex/lux.shp", package="terra"))
plot(v)
add_box(col="red", lwd=3, xpd=TRUE)
```

---

add_grid — *add a grid to a map made with terra*

Description

Adaptation of grid that allows adding a grid to a map. This function will place the legend in the locations within the mapped area as delineated by the axes.

Also see graticule

Usage

```
add_grid(nx=NULL, ny=nx, col="lightgray", lty="dotted", lwd=1)
```

Arguments

nx, ny	number of cells of the grid in x and y direction. When NULL, as per default, the grid aligns with the tick marks on the corresponding default axis (i.e., tickmarks as computed by axTicks). When NA, no grid lines are drawn in the corresponding direction
col	character or (integer) numeric; color of the grid lines
lty	character or (integer) numeric; line type of the grid lines
lwd	non-negative numeric giving line width of the grid lines

See Also

graticule, add_legend, add_box, add_grid, add_mtext

Examples

```
v <- vect(system.file("ex/lux.shp", package="terra"))
plot(v)
add_grid()
```

---

add_legend *add a custom legend*

Description

Wrapper around legend that allows adding a custom legend to a map using a keyword such as "topleft" or "bottomright". This function will place the legend in the locations within the mapped area as delineated by the axes.

Usage

```
add_legend(x, y, ...)
```

Arguments

x	The keyword to be used to position the legend (or the x coordinate)
y	The y coordinate to be used to position the legend (is x is also a coordinate)
...	arguments passed to legend

See Also

add_box, add_grid, add_mtext

Examples

```
v <- vect(system.file("ex/lux.shp", package="terra"))
plot(v)
points(centroids(v), col="red")
legend("topleft", legend = "centroids", pch = 20, xpd=NA, bg="white", col="red")
add_legend("topright", legend = "centroids", pch = 20, col="red")
```

---

add_mtext *draw a box*

Description

Similar to `mtext` allowing adding a text to the margins of a map. This function useds the margins around the mapped area; not the margins that R would use.

Usage

```
add_mtext(text, side=3, line=0, ...)
```

Arguments

`text`	character or expression vector specifying the text to be written
`side`	integer indicating the margin to use (1=bottom, 2=left, 3=top, 4=right)
`line`	numeric to move the text in or outwards.
`...`	arguments passed to `text`

See Also

`add_legend`, `add_grid`, `add_box`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)

plot(r, axes=FALSE, legend=FALSE)
add_box()
for (i in 1:4) add_mtext("margin text", i, cex=i, col=i, line=2-i)
```

`adjacent` *Adjacent cells*

Description

Identify cells that are adjacent to a set of raster cells. Or identify adjacent polygons

Usage

```
## S4 method for signature 'SpatRaster'
adjacent(x, cells, directions="rook", pairs=FALSE, include=FALSE, symmetrical=FALSE)

## S4 method for signature 'SpatVector'
adjacent(x, type="rook", pairs=TRUE, symmetrical=FALSE)
```

Arguments

`x`	SpatRaster
`cells`	vector of cell numbers for which adjacent cells should be found. Cell numbers start with 1 in the upper-left corner and increase from left to right and from top to bottom
`directions`	character or matrix to indicated the directions in which cells are considered connected. The following character values are allowed: "rook" or "4" for the horizontal and vertical neighbors; "bishop" to get the diagonal neighbors; "queen" or "8" to get the vertical, horizontal and diagonal neighbors; or "16" for knight and one-cell queen move neighbors. If `directions` is a matrix it should have odd dimensions and have logical (or 0, 1) values
`pairs`	logical. If `TRUE`, a two-column matrix of pairs of adjacent cells is returned. If `x` is a `SpatRaster` and `pairs` is `FALSE`, an `n*m` matrix is returned where the number of rows `n` is `length(cells)` and the number of columns `m` is the number of neighbors requested with `directions`
`include`	logical. Should the focal cells be included in the result?
`type`	character. One of "rook", "queen", "touches", or "intersects". "queen" and "touches" are synonyms. "rook" exclude polygons that touch at a single node only. "intersects" includes polygons that touch or overlap
`symmetrical`	logical. If `TRUE` and `pairs=TRUE`, an adjacent pair is only included once. For example, if polygon 1 is adjacent to polygon 3, the implied adjacency between 3 and 1 is not reported

Value

matrix

Note

When using global lon/lat rasters, adjacent cells at the other side of the date-line are included.

See Also

relate, nearby

Examples

```
r <- rast(nrows=10, ncols=10)
adjacent(r, cells=c(1, 5, 55), directions="queen")
r <- rast(nrows=10, ncols=10, crs="+proj=utm +zone=1 +datum=WGS84")
adjacent(r, cells=11, directions="rook")

#same as
rk <- matrix(c(0,1,0,1,0,1,0,1,0), 3, 3)
adjacent(r, cells=11, directions=rk)

## note that with global lat/lon data the E and W connect
r <- rast(nrows=10, ncols=10, crs="+proj=longlat +datum=WGS84")
adjacent(r, cells=11, directions="rook")

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
a <- adjacent(v, symmetrical=TRUE)
head(a)
```

---

aggregate — *Aggregate raster or vector data*

Description

Aggregate a SpatRaster to create a new SpatRaster with a lower resolution (larger cells). Aggregation groups rectangular areas to create larger cells. The value for the resulting cells is computed with a user-specified function.

You can also aggregate ("dissolve") a SpatVector. This either combines all geometries into one geometry, or it combines the geometries that have the same value for the variable(s) specified with argument by.

Usage

```
## S4 method for signature 'SpatRaster'
aggregate(x, fact=2, fun="mean", ..., cores=1, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatVector'
aggregate(x, by=NULL, dissolve=TRUE, fun="mean", count=TRUE, ...)
```

Arguments

x	SpatRaster

fact	positive integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers)
fun	function used to aggregate values. Either an actual function, or for the following, their name: "mean", "max", "min", "median", "sum", "modal", "any", "all", "prod", "which.min", "which.max", "sd" (sample standard deviation) and "std" (population standard deviation)
...	additional arguments passed to `fun`, such as `na.rm=TRUE`
cores	positive integer. If `cores` > 1, a 'parallel' package cluster with that many cores is created. Ignored for C++ level implemented functions that are listed under `fun`
filename	character. Output filename
overwrite	logical. If `TRUE`, `filename` is overwritten
wopt	list with named options for writing files as in `writeRaster`
by	character. The variable(s) used to group the geometries
dissolve	logical. Should borders between aggregated geometries be dissolved?
count	logical. If `TRUE` and by is not `NULL`, a variable "agg_n" is included that shows the number of input geometries for each output geometry

Details

Aggregation starts at the upper-left end of a SpatRaster. If a division of the number of columns or rows with `factor` does not return an integer, the extent of the resulting SpatRaster will be somewhat larger then that of the original SpatRaster. For example, if an input SpatRaster has 100 columns, and `fact=12`, the output SpatRaster will have 9 columns and the maximum x coordinate of the output SpatRaster is also adjusted.

The function `fun` should take multiple numbers, and return a single number. For example `mean`, `modal`, `min` or `max`.

It should also accept a `na.rm` argument (or ignore it as one of the 'dots' arguments).

Value

SpatRaster

See Also

`disagg` to disaggregate

Examples

```
r <- rast()
# aggregated SpatRaster, no values
ra <- aggregate(r, fact=10)

values(r) <- runif(ncell(r))
# aggregated raster, max of the values
```

```
ra <- aggregate(r, fact=10, fun=max)

# multiple layers
s <- c(r, r*2)
x <- aggregate(s, 20)


## SpatVector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
va <- aggregate(v, "ID_1")

plot(va, "NAME_1", lwd=5, plg=list(x="topright"), mar=rep(2,4))
lines(v, lwd=3, col="light gray")
lines(va)
text(v, "ID_1", halo=TRUE)
```

`align` *Align a SpatExtent*

Description

Align an SpatExtent with a SpatRaster This can be useful to create a new SpatRaster with the same origin and resolution as an existing SpatRaster. Do not use this to force data to match that really does not match (use e.g. `resample` or (dis)aggregate for this).

It is also possible to align a SpatExtent to a clean divisor.

Usage

```
## S4 method for signature 'SpatExtent,SpatRaster'
align(x, y, snap="near")

## S4 method for signature 'SpatExtent,numeric'
align(x, y)
```

Arguments

`x`	SpatExtent
`y`	SpatRaster or numeric
`snap`	Character. One of "near", "in", or "out", to determine in which direction the extent should be aligned. To the nearest border, inwards or outwards

Value

SpatExtent

See Also

`ext`, `draw`

Examples

```
r <- rast()
e <- ext(-10.1, 9.9, -20.1, 19.9)
ea <- align(e, r)
e
ext(r)
ea

align(e, 0.5)
```

all.equal *Compare two SpatRasters for equality*

Description

Compare two SpatRasters for (near) equality.

First the attributes of the objects are compared. If these are the same, a (perhaps small) sample of the raster cells is compared as well.

The sample size used can be increased with the `maxcell` argument. You can set it to `Inf`, but for large rasters your computer may not have sufficient memory. See the examples for a safe way to compare all values.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
all.equal(target, current, maxcell=100000, ...)
```

Arguments

`target`	SpatRaster
`current`	SpatRaster
`maxcell`	positive integer. The size of the regular sample used to compare cell values
`...`	additional arguments passed to `all.equal.numeric` to compare cell values

Value

Either `TRUE` or a character vector describing the differences between target and current.

See Also

`identical`, `compareGeom`

Examples

```
x <- sqrt(1:100)
mat <- matrix(x, 10, 10)
r1 <- rast(nrows=10, ncols=10, xmin=0, vals = x)
r2 <- rast(nrows=10, ncols=10, xmin=0, vals = mat)

all.equal(r1, r2)
all.equal(r1, r1*1)
all.equal(rast(r1), rast(r2))

# compare geometries
compareGeom(r1, r2)

# Compare all cell values for near equality
# as floating point number imprecision can be a problem
m <- minmax(r1 - r2)
all(abs(m) < 1e-7)

# comparison of cell values to create new SpatRaster
e <- r1 == r2
```

`animate` *Animate a SpatRaster*

Description

Animate (sequentially plot) the layers of a SpatRaster to create a movie.

This does not work with R-Studio.

Usage

```
## S4 method for signature 'SpatRaster'
animate(x, pause=0.25, main, range, maxcell=50000, n=1, ...)
```

Arguments

`x`	SpatRaster
`pause`	numeric. How long should be the pause be between layers?
`main`	title for each layer. If not supplied the z-value is used if available. Otherwise the names are used.
`range`	numeric vector of length 2. Range of values to plot
`maxcell`	positive integer. Maximum number of cells to use for the plot. If `maxcell` < `ncell(x)`, `spatSample(type="regular")` is used before plotting
`n`	integer > 0. Number of loops
`...`	Additional arguments passed to `plot`

Value

None

See Also

plot

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
animate(s, n=1)
```

---

app *Apply a function to the cells of a SpatRaster*

Description

Apply a function to the values of each cell of a SpatRaster. Similar to apply – think of each layer in a SpatRaster as a column (or row) in a matrix.

This is generally used to summarize the values of multiple layers into one layer; but this is not required.

app calls function fun with the raster data as first argument. Depending on the function supplied, the raster data is represented as either a matrix in which each layer is a column, or a vector representing a cell. The function should return a vector or matrix that is divisible by ncell(x). Thus, both "sum" and "rowSums" can be used, but "colSums" cannot be used.

You can also apply a function fun across datasets by layer of a SpatRasterDataset. In that case, summarization is by layer across SpatRasters.

Usage

```
## S4 method for signature 'SpatRaster'
app(x, fun, ..., cores=1, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterDataset'
app(x, fun, ..., cores=1, filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatRaster or SpatRasterDataset
fun	a function that operates on a vector or matrix. This can be a function that is defined in base-R or in a package, or a function you write yourself (see examples). Functions that return complex output (e.g. a list) may need to be wrapped in your own function to simplify the output to a vector or matrix. The following functions have been re-implemented in C++ for speed: "sum", "mean", "median", "modal", "which", "which.min", "which.max", "min", "max", "prod", "any", "all", "sd", "std", "first". To use the base-R function for say, "min", you could use something like `fun=function(i) min(i)` or the equivalent `fun = \(i) min(i)`

`...`	additional arguments for `fun`. These are typically numerical constants. They should *never* be another SpatRaster
`cores`	positive integer. If `cores` > 1, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under fun)
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Details

To speed things up, parallelization is supported, but this is often not helpful, and it may actually be slower. There is only a speed gain if you have many cores (> 8) and/or a very complex (slow) function `fun`. If you write `fun` yourself, consider supplying a `cppFunction` made with the Rcpp package instead (or go have a cup of tea while the computer works for you).

Value

SpatRaster

See Also

`lapp`, `tapp`, `Math-methods`, `roll`

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
x <- c(r, sqrt(r), r+50)
s <- app(x, fun=sum)
s
# for a few generic functions like
# "sum", "mean", and "max" you can also do
sum(x)

## SpatRasterDataset
sd <- sds(x, x*2, x/3)
a <- app(sd, max)
a
# same as
max(x, x*2, x/3)
# and as (but slower)
b <- app(sd, function(i) max(i))


## also works for a single layer
f <- function(i) (i+1) * 2 * i + sqrt(i)
s <- app(r, f)
# same as above, but that is not memory-safe
# and has no filename argument
```

```
s <- f(r)

## Not run:
#### multiple cores
test0 <- app(x, sqrt)
test1 <- app(x, sqrt, cores=2)

testfun <- function(i) { 2 * sqrt(i) }
test2 <- app(x, fun=testfun, cores =2)

## this fails because testfun is not exported to the nodes
# test3 <- app(x, fun=function(i) testfun(i), cores=2)
## to export it, add it as argument to fun
test3 <- app(x, fun=function(i, ff) ff(i), cores =3, ff=testfun)

## End(Not run)
```

`approximate` *Estimate values for cell values that are* `NA` *by interpolating between layers*

Description

approximate uses the `stats` function `approx` to estimate values for cells that are `NA` by interpolation across layers. Layers are considered equidistant, unless argument `z` is used, or `time(x)` returns values that are not `NA`, in which case these values are used to determine distance between layers.

For estimation based on neighboring cells see `focal`

Usage

```
## S4 method for signature 'SpatRaster'
approximate(x, method="linear", yleft, yright,
             rule=1, f=0, ties=mean, z=NULL, NArule=1,filename="",  ...)
```

Arguments

`x`	SpatRaster
`method`	specifies the interpolation method to be used. Choices are "linear" or "constant" (step function; see the example in `approx`
`yleft`	the value to be returned before a non-`NA` value is encountered. The default is defined by the value of rule given below
`yright`	the value to be returned after the last non-`NA` value is encountered. The default is defined by the value of rule given below
`rule`	an integer (of length 1 or 2) describing how interpolation is to take place at for the first and last cells (before or after any non-`NA` values are encountered). If rule is 1 then NAs are returned for such points and if it is 2, the value at the closest data extreme is used. Use, e.g., `rule = 2:1`, if the left and right side extrapolation should differ

f	for method = "constant" a number between 0 and 1 inclusive, indicating a compromise between left- and right-continuous step functions. If y0 and y1 are the values to the left and right of the point then the value is `y0*(1-f)+y1*f` so that `f = 0)` is right-continuous and `f = 1` is left-continuous
ties	Handling of tied 'z' values. Either a function with a single vector argument returning a single number result or the string "ordered"
z	numeric vector to indicate the distance between layers (e.g., depth). The default is `time(x)` if these are not `NA` or else `1:nlys(x)`
NArule	single integer used to determine what to do when only a single layer with a non-`NA` value is encountered (and linear interpolation is not possible). The default value of 1 indicates that all layers will get this value for that cell; all other values do not change the cell values
filename	character. Output filename
...	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`focal`, `fillTime`

Examples

```
r <- rast(ncols=5, nrows=5)
r1 <- setValues(r, runif(ncell(r)))
r2 <- setValues(r, runif(ncell(r)))
r3 <- setValues(r, runif(ncell(r)))
r4 <- setValues(r, runif(ncell(r)))
r5 <- setValues(r, NA)
r6 <- setValues(r, runif(ncell(r)))
r1[6:10] <- NA
r2[5:15] <- NA
r3[8:25] <- NA
s <- c(r1,r2,r3,r4,r5,r6)
s[1:5] <- NA
x1 <- approximate(s)
x2 <- approximate(s, rule=2)
x3 <- approximate(s, rule=2, z=c(1,2,3,5,14,15))
```

Arith-methods *Arithmetic*

Description

Standard arithmetic operators for computations with SpatRasters. Computations are local (applied on a cell by cell basis). If multiple SpatRasters are used, these must have the same geometry (extent and resolution). These operators have been implemented:

`+, -, *, /, ^, %%, %/%`

You can also use a SpatRaster and a vector or a matrix. If you use a SpatRaster with a vector of multiple numbers, each element in the vector is considered a layer (with a constant value). If you use a SpatRaster with a matrix, the number of columns of the matrix must match the number of layers of the SpatRaster. The rows are used to match the cells. That is, if there are two rows, these match cells 1 and 2, and they are recycled to 3 and 4, etc.

The following methods have been implemented for `(SpatExtent, SpatExtent)`: `+`, `-`, and the following for `(SpatExtent, numeric)`: `+`, `-`, `*`, `/`, `%%`

Value

SpatRaster or SpatExtent

See Also

`ifel` to conveniently combine operations and `Math-methods` or `app` to use mathematical functions not implemented by the package.

Examples

```
r1 <- rast(ncols=10, nrows=10)
v <- runif(ncell(r1))
v[10:20] <- NA
values(r1) <- v
r2 <- rast(r1)
values(r2) <- 1:ncell(r2) / ncell(r2)
r3 <- r1 + r2
r2 <- r1 / 10
r3 <- r1 * (r2 - 1 / r2)

b <- c(r1, r2, r3)
b2 <- b * 10

### SpatExtent methods
x <- ext(0.1, 2.2, 0, 3)
y <- ext(-2, 1, -2,2)
# union
x + y
# intersection
x * y
```

```
e <- x
e
e * 2
e / 2
e + 1
e - 1
```

---

as.character *Create a text representation of (the skeleton of) an object*

Description

Create a text representation of (the skeleton of) an object

Usage

```
## S4 method for signature 'SpatExtent'
as.character(x)

## S4 method for signature 'SpatRaster'
as.character(x)
```

Arguments

x	SpatRaster

Value

character

Examples

```
r <- rast()
ext(r)
ext(c(0, 20, 0, 20))
```

as.data.frame *SpatRaster or SpatVector to data.frame*

Description

Coerce a SpatRaster or SpatVector to a data.frame

Usage

```
## S4 method for signature 'SpatVector'
as.data.frame(x, row.names=NULL, optional=FALSE, geom=NULL, ...)

## S4 method for signature 'SpatRaster'
as.data.frame(x, row.names=NULL, optional=FALSE, xy=FALSE,
cells=FALSE, time=FALSE, na.rm=NA, wide=TRUE, ...)
```

Arguments

x	SpatRaster or SpatVector
geom	character or NULL. If not NULL, either "WKT" or "HEX", to get the geometry included in Well-Known-Text or hexadecimal notation. If x has point geometry, it can also be "XY" to add the coordinates of each point
xy	logical. If TRUE, the coordinates of each raster cell are included
time	logical. If TRUE, the time data is included (if available)
na.rm	logical. If TRUE, cells that have a NA value in at least one layer are removed. If the argument is set to NA only cells that have NA values in all layers are removed
cells	logical. If TRUE, the cell numbers of each raster cell are included
wide	logical. If FALSE, the data.frame returned has a "long" format
...	Additional arguments passed to the data.frame
row.names	This argument is ignored
optional	This argument is ignored

Value

data.frame

See Also

as.list, as.matrix. See geom to only extract the geometry of a SpatVector

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
as.data.frame(v)
```

as.lines *Conversion to a SpatVector of lines*

Description

Conversion of a SpatRaster, SpatVector or SpatExtent to a SpatVector of lines.

Usage

```
## S4 method for signature 'SpatRaster'
as.lines(x)

## S4 method for signature 'SpatVector'
as.lines(x)

## S4 method for signature 'SpatExtent'
as.lines(x, crs="")

## S4 method for signature 'matrix'
as.lines(x, crs="")
```

Arguments

x	SpatRaster, SpatVector, SpatExtent or matrix. If x is a matrix it should have two columns for a single line, or four columns, where each row has the start and end coordinates (x, y) for lines
crs	character. The coordinate reference system (see crs)

Value

SpatVector

See Also

as.points, as.polygons

Examples

```
r <- rast(ncols=2, nrows=2)
values(r) <- 1:ncell(r)

as.lines(r)

as.lines(ext(r), crs=crs(r))

if  (gdal() >= "3.0.0") {
p <- as.polygons(r)
as.lines(p)
```

```
}


## with a matrix
s <- cbind(1:5, 1:5)
e <- cbind(1:5, 0)

as.lines(s)
as.lines(cbind(s, e), "+proj=longlat")
```

---

as.list *Coerce a Spat* object to a list*

Description

Coerce a SpatRaster, SpatRasterCollection, SpatRasterDataset, SpatVector or SpatVectorCollection to a list. With a SpatRaster, each layer becomes a list element. With a SpatRasterCollection or SpatRasterDataset, each SpatRaster becomes a list element. With a SpatVector, each variable (attribute) becomes a list element. With a SpatVectorCollection, each SpatVector becomes a list element.

Usage

```
## S4 method for signature 'SpatRaster'
as.list(x, geom=NULL, ...)

## S4 method for signature 'SpatRasterCollection'
as.list(x, ...)

## S4 method for signature 'SpatVector'
as.list(x, geom=NULL, ...)

## S4 method for signature 'SpatVectorCollection'
as.list(x, ...)
```

Arguments

`x`	SpatRaster, SpatRasterDataset, SpatRasterCollection, or SpatVector
`geom`	character or NULL. If not NULL, and `x` is a SpatVector, it should be either "WKT" or "HEX", to get the geometry included in Well-Known-Text or hexadecimal notation. If `x` has point geometry, it can also bey "XY" to add the coordinates of each point. If `x` is a SpatRaster, any value that is not NULL will return a list with the the parameters describing the geometry of the SpatRaster are returned
`...`	additional arguments. These are ignored

Value

list

See Also

see coerce for as.data.frame with a SpatRaster; and geom to only extract the geometry of a SpatVector

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
as.list(v)


s <- rast(system.file("ex/logo.tif", package="terra")) + 1
as.list(s)
```

`as.points` *Conversion to a SpatVector of points*

Description

Conversion of a SpatRaster, SpatVector or SpatExtent to a SpatVector of points.

Usage

```
## S4 method for signature 'SpatRaster'
as.points(x, values=TRUE, na.rm=TRUE, na.all=FALSE)

## S4 method for signature 'SpatVector'
as.points(x, multi=FALSE, skiplast=TRUE)

## S4 method for signature 'SpatExtent'
as.points(x, crs="")
```

Arguments

`x`	SpatRaster, SpatVector or SpatExtent
`values`	logical; include cell values as attributes?
`multi`	logical. If `TRUE` a multi-point geometry is returned
`skiplast`	logical. If `TRUE` the last point of a polygon (which is the same as the first point) is not included
`na.rm`	logical. If `TRUE` cells that are `NA` are ignored
`na.all`	logical. If `TRUE` cells are only ignored if `na.rm=TRUE` and their value is `NA` for **all** layers instead of for any layer
`crs`	character. The coordinate reference system (see crs)

Value

SpatVector

See Also

as.lines, as.points

Examples

```
r <- rast(ncols=2, nrows=2)
values(r) <- 1:ncell(r)

as.points(r)

if  (gdal() >= "3.0.0") {
p <- as.polygons(r)
as.points(p)
}
```

---

as.polygons *Conversion to a SpatVector of polygons*

Description

Conversion of a SpatRaster, SpatVector or SpatExtent to a SpatVector of polygons.

Usage

```
## S4 method for signature 'SpatRaster'
as.polygons(x, round=TRUE, aggregate=TRUE, values=TRUE,
na.rm=TRUE, na.all=FALSE, extent=FALSE, digits=0, ...)

## S4 method for signature 'SpatVector'
as.polygons(x, extent=FALSE)

## S4 method for signature 'SpatExtent'
as.polygons(x, crs="")
```

Arguments

x	SpatRaster, SpatVector or SpatExtent
round	logical; If TRUE and aggregate=TRUE, values are rounded before aggregation. If this value is FALSE the SpatVector returned can have very many polygons and can be very large
aggregate	logical; combine cells with the same values? If TRUE only the first layer in x is processed
values	logical; include cell values as attributes?

extent	logical. if TRUE, a polygon for the extent of the SpatRaster or SpatVector is returned. If x is a SpatRaster, the polygon has vertices for each row and column, not just the four corners of the raster. This can be useful for more precise projection. If that is not required, it is more efficient to get the extent represented by only the four corners with as.polygons(ext(x), crs=crs(x))
na.rm	logical. If TRUE cells that are NA are ignored
na.all	logical. If TRUE cells are only ignored if na.rm=TRUE and their value is NA for **all** layers instead of for any layer
digits	integer. The number of digits for rounding (if round=TRUE)
crs	character. The coordinate reference system (see crs)
...	additional arguments. For backward compatibility. Will be removed in the future

Value

SpatVector

See Also

as.lines, as.points

Examples

```
r <- rast(ncols=2, nrows=2)
values(r) <- 1:ncell(r)

if  (gdal() >= "3.0.0") {
p <- as.polygons(r)
p
}
```

---

as.raster *Coerce to a "raster" object*

Description

Implementation of the generic as.raster function to create a "raster" (small r) object. Such objects can be used for plotting with the rasterImage function. NOT TO BE CONFUSED with the Raster* (big R) objects defined by the 'raster' package!

Usage

```
## S4 method for signature 'SpatRaster'
as.raster(x, maxcell=500000, col)
```

Arguments

`x`	SpatRaster
`maxcell`	positive integer. Maximum number of cells to use for the plot
`col`	vector of colors. The default is `map.pal("viridis", 100)`

Value

'raster' object

Examples

```
r <- rast(ncols=3, nrows=3)
values(r) <- 1:ncell(r)
as.raster(r)
```

---

atan2 *Two argument arc-tangent*

Description

For SpatRasters x and y, atan2(y, x) returns the angle in radians for the tangent y/x, handling the case when x is zero. See `Trig`

See `Math-methods` for other trigonometric and mathematical functions that can be used with SpatRasters.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
atan2(y, x)

## S4 method for signature 'SpatRaster,SpatRaster'
atan_2(y, x, filename, ...)
```

Arguments

`y`	SpatRaster
`x`	SpatRaster
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

See Also

`Math-methods`

Examples

```
r1 <- rast(nrows=10, ncols=10)
r2 <- rast(nrows=10, ncols=10)
values(r1) <- (runif(ncell(r1))-0.5) * 10
values(r2) <- (runif(ncell(r1))-0.5) * 10
atan2(r1, r2)
```

---

autocorrelation *Spatial autocorrelation*

Description

Compute spatial autocorrelation for a numeric vector or a SpatRaster. You can compute standard (global) Moran's I or Geary's C, or local indicators of spatial autocorrelation (Anselin, 1995).

Usage

```
## S4 method for signature 'numeric'
autocor(x, w, method="moran")

## S4 method for signature 'SpatRaster'
autocor(x, w=matrix(c(1,1,1,1,0,1,1,1,1),3), method="moran", global=TRUE)
```

Arguments

`x`	numeric or SpatRaster
`w`	Spatial weights defined by or a rectangular matrix. For a SpatRaster this matrix must the sides must have an odd length (3, 5, ...)
`global`	logical. If `TRUE` global autocorrelation is computed instead of local autocorrelation
`method`	character. If `x` is numeric or SpatRaster: "moran" for Moran's I and "geary" for Geary's C. If `x` is numeric also: "Gi", "Gi*" (the Getis-Ord statistics), locmor (local Moran's I) and "mean" (local mean)

Details

The default setting uses a 3x3 neighborhood to compute "Queen's case" indices. You can use a filter (weights matrix) to do other things, such as "Rook's case", or different lags.

Value

numeric or SpatRaster

References

Moran, P.A.P., 1950. Notes on continuous stochastic phenomena. Biometrika 37:17-23

Geary, R.C., 1954. The contiguity ratio and statistical mapping. The Incorporated Statistician 5: 115-145

Anselin, L., 1995. Local indicators of spatial association-LISA. Geographical Analysis 27:93-115

https://en.wikipedia.org/wiki/Indicators_of_spatial_association

See Also

The spdep package for additional and more general approaches for computing spatial autocorrelation

Examples

```
### raster
r <- rast(nrows=10, ncols=10, xmin=0)
values(r) <- 1:ncell(r)

autocor(r)

# rook's case neighbors
f <- matrix(c(0,1,0,1,0,1,0,1,0), nrow=3)
autocor(r, f)

# local
rc <- autocor(r, w=f, global=FALSE)

### numeric (for vector data)
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- relate(v, relation="touches")

# global
autocor(v$AREA, w)

# local
v$Gi <- autocor(v$AREA, w, "Gi")
plot(v, "Gi")
```

---

barplot *Bar plot of a SpatRaster*

Description

Create a barplot of the values of the first layer of a SpatRaster. For large datasets a regular sample with a size of approximately `maxcells` is used.

Usage

```
## S4 method for signature 'SpatRaster'
barplot(height, maxcell=1000000, digits=0, breaks=NULL, col, ...)
```

Arguments

height	SpatRaster
maxcell	integer. To regularly subsample very large datasets
digits	integer used to determine how to round the values before tabulating. Set to NULL or to a large number if you do not want any rounding
breaks	breaks used to group the data as in cut
col	a color generating function such as rainbow (the default), or a vector of colors
...	additional arguments for plotting as in barplot

Value

A numeric vector (or matrix, when beside = TRUE) of the coordinates of the bar midpoints, useful for adding to the graph. See barplot

See Also

hist, boxplot

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
barplot(r, digits=-1, las=2, ylab="Frequency")

op <- par(no.readonly = TRUE)
par(mai = c(1, 2, .5, .5))
barplot(r, breaks=10, col=c("red", "blue"), horiz=TRUE, digits=NULL, las=1)
par(op)
```

---

bestMatch *bestMatch*

Description

Determine for each grid cell which reference it is most similar to. A reference consists of a SpatVector with reference locations, or a data.frame in which each column matches a layer name in the SpatRaster.

Similarity is computed with the sum of squared differences between the cell and the reference. It may be important to first scale the input.

Usage

```
## S4 method for signature 'SpatRaster,SpatVector'
bestMatch(x, y, labels="", filename="", ...)

## S4 method for signature 'SpatRaster,data.frame'
bestMatch(x, y, labels="", filename="", ...)
```

Arguments

`x`	SpatRaster
`y`	SpatVector or data.frame
`labels`	character. labels that correspond to each class (row in `y`
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

Examples

```
f <- system.file("ex/logo.tif", package = "terra")
r <- rast(f)

# locations of interest
pts <- vect(cbind(c(25.25, 34.324, 43.003), c(54.577, 46.489, 30.905)))
pts$code <- LETTERS[1:3]

plot(r)
points(pts, pch=20, cex=2, col="red")
text(pts, "code", pos=4, halo=TRUE)

x <- scale(r)

s1 <- bestMatch(x, pts, labels=pts$code)
plot(s1)

# same result
e <- extract(x, pts, ID=FALSE)
s2 <- bestMatch(x, e, labels=c("Ap", "Nt", "Ms"))
```

`boundaries` *Detect boundaries (edges)*

Description

Detect boundaries (edges). Boundaries are cells that have more than one class in the 4 or 8 cells surrounding it, or, if `classes=FALSE`, cells with values and cells with `NA`.

Usage

```
## S4 method for signature 'SpatRaster'
boundaries(x, classes=FALSE, inner=TRUE,
         directions=8, falseval=0, filename="", ...)
```

Arguments

`x`	SpatRaster
`inner`	logical. If `TRUE`, "inner" boundaries are returned, else "outer" boundaries are returned
`classes`	character. Logical. If `TRUE` all different values are (after rounding) distinguished, as well as `NA`. If `FALSE` (the default) only edges between `NA` and non-`NA` cells are considered
`directions`	integer. Which cells are considered adjacent? Should be 8 (Queen's case) or 4 (Rook's case)
`falseval`	numeric. The value to use for cells that are not a boundary and not `NA`
`filename`	character. Output filename
`...`	options for writing files as in `writeRaster`

Value

SpatRaster. Cell values are either 1 (a border) or 0 (not a border), or `NA`

See Also

`focal`, `patches`

Examples

```
r <- rast(nrows=18, ncols=36, xmin=0)
r[150:250] <- 1
r[251:450] <- 2
bi <- boundaries(r)
bo <- boundaries(r, inner=FALSE)
bc <- boundaries(r, classes=TRUE)
#plot(bc)
```

`boxplot` *Box plot of SpatRaster data*

Description

Box plot of layers in a SpatRaster

Usage

```
## S4 method for signature 'SpatRaster'
boxplot(x, y=NULL, maxcell=100000, ...)
```

Arguments

x	SpatRaster
y	NULL or a SpatRaster. If x is a SpatRaster it used to group the values of x by "zone"
maxcell	Integer. Number of cells to sample from datasets
...	additional arguments passed to graphics::boxplot

Value

boxplot returns a list (invisibly) that can be used with bxp

See Also

pairs, hist

Examples

```
r1 <- r2 <- r3 <- rast(ncols=10, nrows=10)
set.seed(409)
values(r1) <- rnorm(ncell(r1), 100, 40)
values(r2) <- rnorm(ncell(r1), 80, 10)
values(r3) <- rnorm(ncell(r1), 120, 30)
s <- c(r1, r2, r3)
names(s) <- c("Apple", "Pear", "Cherry")

boxplot(s, notch=TRUE, col=c("red", "blue", "orange"), main="Box plot", ylab="random", las=1)

op <- par(no.readonly = TRUE)
par(mar=c(4,6,2,2))
boxplot(s, horizontal=TRUE, col="lightskyblue", axes=FALSE)
axis(1)
axis(2, at=0:3, labels=c("", names(s)), las=1, cex.axis=.9, lty=0)
par(op)

## boxplot with 2 layers
v <- vect(system.file("ex/lux.shp", package="terra"))
r <- rast(system.file("ex/elev.tif", package="terra"))
y <- rasterize(v, r, "NAME_2")
b <- boxplot(r, y)
bxp(b)
```

buffer *Create a buffer around vector geometries or raster patches*

Description

Calculate a buffer around all cells that are not NA in a SpatRaster, or around the geometries of a SpatVector.

SpatRaster cells inside the buffer distance get a value of 1.

Note that the distance unit of the buffer width parameter is meters if the CRS is (+proj=longlat), and in map units (typically also meters) if not.

Usage

```
## S4 method for signature 'SpatRaster'
buffer(x, width, background=0, filename="", ...)

## S4 method for signature 'SpatVector'
buffer(x, width, quadsegs=10, capstyle="round",
joinstyle="round", mitrelimit=NA, singlesided=FALSE)
```

Arguments

x	SpatRaster or SpatVector
width	numeric. Unit is meter if x has a longitude/latitude CRS, or in the units of the coordinate reference system in other cases (typically also meter). The value should be > 0 if x is a SpatRaster. If x is a SpatVector, this argument is vectorized, meaning that you can provide a different value for each geometry in x; and you can also use the name of a variable in x that has the widths
filename	character. Output filename
...	additional arguments for writing files as in writeRaster
background	numeric. value to assign to cells outside the buffer. If this value is zero or FALSE, a boolean SpatRaster is returned
quadsegs	positive integer. Number of line segments to use to draw a quart circle
capstyle	character. One of "round", "square" or "flat". Ignored if is.lonlat(x)
joinstyle	character. One of "round", "mitre" or "bevel". Ignored if is.lonlat(x)
mitrelimit	numeric. Place an upper bound on a mitre join to avoid it from extending very far from acute angles in the input geometry. Ignored if is.lonlat(x)
singlesided	logical. If TRUE a buffer is constructed on only one side of each input line. Ignored if is.lonlat(x)

Value

Same as x

See Also

distance, elongate

Examples

```
r <- rast(ncols=36, nrows=18)
r[500] <- 1
b <- buffer(r, width=5000000)
plot(b)

v <- vect(rbind(c(10,10), c(0,60)), crs="+proj=merc")
b <- buffer(v, 20)
plot(b)
points(v)

crs(v) <- "+proj=longlat"
b <- buffer(v, 1500000)
plot(b)
points(v)
```

---

c — *Combine SpatRaster or SpatVector objects*

Description

With c you can:

– Combine SpatRaster objects. They must have the same extent and resolution. However, if x is empty (has no cell values), its geometry is ignored with a warning. Two empty SpatRasters with the same geometry can also be combined (to get a summed number of layers). Also see add<-

– Add a SpatRaster to a SpatRasterDataset or SpatRasterCollection

– Add SpatVector objects to a new or existing SpatVectorCollection

To append SpatVectors, use rbind.

Usage

```
## S4 method for signature 'SpatRaster'
c(x, ..., warn=TRUE)

## S4 method for signature 'SpatRasterDataset'
c(x, ...)

## S4 method for signature 'SpatRasterCollection'
c(x, ...)

## S4 method for signature 'SpatVector'
c(x, ...)
```

```
## S4 method for signature 'SpatVectorCollection'
c(x, ...)
```

Arguments

x	SpatRaster, SpatVector, SpatRasterDataset or SpatVectorCollection
warn	logical. If TRUE, a warning is emitted if x is an empty SpatRaster
...	as for x (you can only combine raster with raster data and vector with vector data)

Value

Same class as x

See Also

add<-

Examples

```
r <- rast(nrows=5, ncols=9)
values(r) <- 1:ncell(r)
x <- c(r, r*2, r*3)
```

---

cartogram *Cartogram*

Description

Make a cartogram, that is, a map where the area of polygons is made proportional to another variable. This can be a good way to map raw count data (e.g. votes).

Usage

```
## S4 method for signature 'SpatVector'
cartogram(x, var, type)
```

Arguments

x	SpatVector
var	character. A variable name in x
type	character. Cartogram type, only "nc" (non-contiguous) is currently supported

Value

SpatVector

See Also

plot, rescale

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$value <- 1:12
p <- cartogram(v, "value", "nc")
plot(v, col="light gray", border="gray")
lines(p, col="red", lwd=2)
```

---

catalyze   *Factors to numeric*

Description

Change a categorical layer into one or more numerical layers. With `as.numeric` you can transfer the active category values to cell values in a non-categorical SpatRaster. `catalyze` creates new layers for each category.

Usage

```
## S4 method for signature 'SpatRaster'
as.numeric(x, index=NULL, filename="", ...)

## S4 method for signature 'SpatRaster'
catalyze(x, filename="", ...)
```

Arguments

`x`	SpatRaster
`index`	positive integer or category indicating the category to use. If `NULL` the active category is used
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

activeCat, cats

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE) + 10
d <- data.frame(id=11:13, cover=c("forest", "water", "urban"), letters=letters[1:3], value=10:12)
levels(r) <- d
catalyze(r)

activeCat(r) <- 3
as.numeric(r)
```

`cells` *Get cell numbers*

Description

Get the cell numbers covered by a SpatVector or SpatExtent. Or that match values in a vector; or all non `NA` values.

Usage

```
## S4 method for signature 'SpatRaster,missing'
cells(x, y)

## S4 method for signature 'SpatRaster,numeric'
cells(x, y, pairs=FALSE)

## S4 method for signature 'SpatRaster,SpatVector'
cells(x, y, method="simple", weights=FALSE, exact=FALSE,
touches=is.lines(y), small=TRUE)

## S4 method for signature 'SpatRaster,SpatExtent'
cells(x, y)
```

Arguments

`x`	SpatRaster
`y`	SpatVector, SpatExtent, 2-column matrix representing points, numeric representing values to match, or missing
`method`	character. Method for getting cell numbers for points. The default is "simple", the alternative is "bilinear". If it is "bilinear", the four nearest cells and their weights are returned
`weights`	logical. If `TRUE` and `y` has polygons, the approximate fraction of each cell that is covered is returned as well
`pairs`	logical. If `TRUE` the cell values matched area also returned

`exact`	logical. If `TRUE` and `y` has polygons, the exact fraction of each cell that is covered is returned as well
`touches`	logical. If `TRUE`, values for all cells touched by lines or polygons are extracted, not just those on the line render path, or whose center point is within the polygon. Not relevant for points
`small`	logical. If `TRUE`, values for all cells in touched polygons are extracted if none of the cells center points is within the polygon; even if `touches=FALSE`

Value

numeric vector or matrix

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
r[c(1:25, 31:100)] <- NA
r <- ifel(r > 28, r + 10, r)

# all cell numbers of cells that are not NA
cells(r)

# cell numbers that match values
x <- cells(r, c(28,38))
x$lyr.1

# cells for points
m <- cbind(x=c(0,10,-30), y=c(40,-10,20))
cellFromXY(r, m)

v <- vect(m)
cells(r, v)
cells(r, v, method="bilinear")

# cells for polygons
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
r <- rast(v)
cv <- cells(r, v)
```

`cellSize` *Area covered by each raster cell*

Description

Compute the area covered by individual raster cells.

Computing the surface area of raster cells is especially relevant for longitude/latitude rasters.

But note that for both angular (longitude/latitude) and for planar (projected) coordinate reference systems raster cells sizes are generally not constant, unless you are using an equal-area coordinate reference system.

For planar CRSs, the area is therefore not computed based on the linear units of the coordinate reference system, but on the *actual* area by transforming cells to longitude/latitude. If you do not want that correction, you can use `transform=FALSE` or `init(x, prod(res(x)))`

Usage

```
## S4 method for signature 'SpatRaster'
cellSize(x, mask=FALSE, lyrs=FALSE, unit="m", transform=TRUE, rcx=100, filename="", ...)
```

Arguments

`x`	SpatRaster
`mask`	logical. If `TRUE`, cells that are `NA` in `x` are also `NA` in the output
`lyrs`	logical. If `TRUE` and `mask=TRUE`, the output has the same number of layers as `x`. That is only useful if cases where the layers of `x` have different cells that are `NA`
`unit`	character. One of "m", "km", or "ha"
`transform`	logical. If `TRUE`, planar CRS data are transformed to lon/lat for accuracy
`rcx`	positive integer. The maximum number of rows and columns to be used to compute area of planar data if `transform=TRUE`. If `x` has more rows and/or columns, the raster is aggregated to match this limit, and values for the original cells are estimated by bilinear interpolation (see `resample`). This can save a lot of time
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`expanse`

Examples

```
# SpatRaster
r <- rast(nrows=18, ncols=36)
v <- 1:ncell(r)
v[200:400] <- NA
values(r) <- v

# size of each raster cell
a <- cellSize(r)

# illustration of distortion
```

```
r <- rast(ncols=90, nrows=45, ymin=-80, ymax=80)
m <- project(r, "+proj=merc")

bad <- init(m, prod(res(m)) / 1000000, names="naive")
good <- cellSize(m, unit="km", names="corrected")
plot(c(good, bad), nc=1, mar=c(2,2,1,6))
```

---

centroids *Centroids*

Description

Get the centroids of polygons or lines, or centroid-like points that are guaranteed to be inside the polygons or on the lines.

Usage

```
## S4 method for signature 'SpatVector'
centroids(x, inside=FALSE)
```

Arguments

x	SpatVector
inside	logical. If TRUE the points returned are guaranteed to be inside the polygons or on the lines, but they are not the true centroids. True centroids may be outside a polygon, for example when a polygon is "bean shaped", and they are unlikely to be on their line

Value

SpatVector of points

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
x <- centroids(v)
y <- centroids(v, TRUE)
```

clamp — *Clamp values*

Description

Clamp values to a minimum and maximum value. That is, all values below a lower threshold value and above the upper threshold value become either NA, or, if values=TRUE, become the threshold value

Usage

```
## S4 method for signature 'SpatRaster'
clamp(x, lower=-Inf, upper=Inf, values=TRUE, filename="", ...)

## S4 method for signature 'numeric'
clamp(x, lower=-Inf, upper=Inf, values=TRUE, ...)
```

Arguments

x	SpatRaster
lower	numeric with the lowest acceptable value (you can specify a different value for each layer). Or a SpatRaster that has a single layer or the same number of layers as x
upper	numeric with the highest acceptable value (you can specify a different value for each layer). Or a SpatRaster that has a single layer or the same number of layers as x
values	logical. If FALSE values outside the clamping range become NA, if TRUE, they get the extreme values
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

classify, subst

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
rc <- clamp(r, 25, 75)
rc
```

clamp_ts *clamp time series data*

Description

clamp time-series datat that are S shaped. The value in layers before the minimum value in a cell can be set to that minimum value, and the value in layers after the maximum value for a cell can be set to that maximum value.

Usage

```
## S4 method for signature 'SpatRaster'
clamp_ts(x, min=FALSE, max=TRUE, filename="", ...)
```

Arguments

x	SpatRaster
min	logical. If TRUE the time-series is clamped to the minimum value
max	logical. If TRUE the time-series is clamped to the maximum value
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

clamp, cummin, cummax

Examples

```
sigm <- function(x) { .8 / (1 + exp(-(x-10))) + runif(length(x))/4 }
r <- rast(ncols=10, nrows=10, nlyr=50)
s <- seq(5.2, 15,.2)
set.seed(1)
values(r) <- t(replicate(100, sigm(s)))

x <- clamp_ts(r, TRUE, TRUE)

plot(unlist(r[4]))
lines(unlist(x[4]))
```

classify *Classify (or reclassify) cell values*

Description

Classify values of a SpatRaster. The function (re-)classifies groups of values to other values.

The classification is done based on the argument `rcl`. You can classify ranges by specifying a three-column matrix "from-to-becomes" or change specific values by using a two-column matrix "is-becomes". You can also supply a vector with "cuts" or the "number of cuts".

With "from-to-becomes" or "is-becomes" classification is done in the row order of the matrix. Thus, if there are overlapping ranges or values, the first time a number is within a range determines the reclassification value.

With "cuts" the values are sorted, so that the order in which they are provided does not matter.

Usage

```
## S4 method for signature 'SpatRaster'
classify(x, rcl, include.lowest=FALSE, right=TRUE,
     others=NULL, brackets=TRUE, filename="", ...)
```

Arguments

`x`	SpatRaster
`rcl`	matrix for classification. This matrix must have 1, 2 or 3 columns. If there are three columns, the first two columns are "from" "to" of the input values, and the third column "becomes" has the new value for that range. The two column matrix ("is", "becomes") can be useful for classifying integer values. In that case, the arguments `right` and `include.lowest` are ignored. A single column matrix (or a vector) is interpreted as a set of cuts if there is more than one value. In that case the values are classified based on their location in-between the cut-values. If a single number is provided, that is used to make that number of cuts, at equal intervals between the lowest and highest values of the SpatRaster.
`include.lowest`	logical, indicating if a value equal to the lowest value in `rcl` (or highest value in the second column, for `right=FALSE`) should be included.
`right`	logical. If `TRUE`, the intervals are closed on the right (and open on the left). If `FALSE` they are open at the right and closed at the left. "open" means that the extreme value is *not* included in the interval. Thus, right-closed and left open is `(0,1] = {x | 0 < x <= 1}`. You can also close both sides with `right=NA`, that is only meaningful if you "from-to-becomes" classification with integers. For example to classify 1-5 -> 1, 6-10 -> 2, 11-15 -> 3. That may be easier to read and write than the equivalent 1-5 -> 1, 5-10 -> 2, 10-15 -> 3 with `right=TRUE` and `include.lowest=TRUE`
`others`	numeric. If not `NULL` all values that are not matched are set to this value. Otherwise they retain their original value.

brackets — logical. If TRUE, intervals are have parenthesis or brackets around them to indicate whether they are open or closed. Only applies if rcl is a vector (or single column matrix)

filename — character. Output filename

... — Additional arguments for writing files as in writeRaster

Value

SpatRaster

Note

classify works with the "raw" values of categorical rasters, ignoring the levels (labels, categories). To change the labels of categorical rasters, use subst instead.

For model-based classification see predict

See Also

subst for simpler from-to replacement, and clamp

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- (0:99)/99

## from-to-becomes
# classify the values into three groups
# all values >= 0 and <= 0.25 become 1, etc.
m <- c(0, 0.25, 1,
       0.25, 0.5, 2,
       0.5, 1, 3)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc1 <- classify(r, rclmat, include.lowest=TRUE)

## cuts
# equivalent to the above, but now a categorical SpatRaster is returned
rc2 <- classify(r, c(0, 0.25, 0.5, 1), include.lowest=TRUE, brackets=TRUE)
freq(rc2)

## is-becomes
x <- round(r*3)
unique(x)
# replace 0 with NA
y <- classify(x, cbind(0, NA))
unique(y)

# multiple replacements
m <- rbind(c(2, 200), c(3, 300))
m

rcx1 <- classify(x, m)
```

```
unique(rcx1)

rcx2 <- classify(x, m, others=NA)
unique(rcx2)
```

`click` *Query by clicking on a map*

Description

Click on a map (plot) to get the coordinates or the values of a SpatRaster or SpatVector at that location. For a SpatRaster you can also get the coordinates and cell number of the location.

This does to work well on the default RStudio plotting device. To work around that, you can first run `dev.new(noRStudioGD = TRUE)` which will create a separate window for plotting, then use `plot()` followed by `click()` and click on the map.

Usage

```
## S4 method for signature 'SpatRaster'
click(x, n=10, id=FALSE, xy=FALSE, cell=FALSE, type="p", show=TRUE, ...)

## S4 method for signature 'SpatVector'
click(x,  n=10, id=FALSE, xy=FALSE, type="p", show=TRUE, ...)

## S4 method for signature 'missing'
click(x, n=10, id=FALSE, type="p", show=TRUE, ...)
```

Arguments

`x`	SpatRaster or SpatVector, or missing
`n`	number of clicks on the plot (map)
`id`	logical. If `TRUE`, a numeric ID is shown on the map that corresponds to the row number of the output
`xy`	logical. If `TRUE`, xy coordinates are included in the output
`cell`	logical. If `TRUE`, cell numbers are included in the output
`type`	one of "n", "p", "l" or "o". If "p" or "o" the points are plotted; if "l" or "o" they are joined by lines. See `locator`
`show`	logical. Print the values after each click?
`...`	additional graphics parameters used if type != "n" for plotting the locations. See `locator`

Value

The value(s) of `x` at the point(s) clicked on (or touched by the box drawn). A `data.frame` with the value(s) of all layers of SpatRaster `x` for the cell(s) clicked on; or with the attributes of the geometries of SpatVector `x` that intersect with the box drawn).

Note

The plot only provides the coordinates for a spatial query, the values are read from the SpatRaster or SpatVector that is passed as an argument. Thus, you can extract values from an object that has not been plotted, as long as it spatially overlaps with the extent of the plot.

Unless the process is terminated prematurely values at most n positions are determined. The identification process can be terminated, depending on how you interact with R, by hitting Esc, or by clicking the right mouse button and selecting "Stop" from the menu, or from the "Stop" menu on the graphics window.

See Also

draw

Examples

```
## Not run:
r <-rast(system.file("ex/elev.tif", package="terra"))
plot(r)
click(r, n=1)
## now click on the plot (map)

## End(Not run)
```

---

coerce *Coercion to vector, matrix or array*

Description

Coercion of a SpatRaster to a vector, matrix or array. Or coerce a SpatExtent to a vector or matrix

Usage

```
## S4 method for signature 'SpatRaster'
as.vector(x, mode='any')

## S4 method for signature 'SpatRaster'
as.matrix(x, wide=FALSE, ...)

## S4 method for signature 'SpatRaster'
as.array(x)

## S4 method for signature 'SpatExtent'
as.vector(x, mode='any')

## S4 method for signature 'SpatExtent'
as.matrix(x, ...)
```

Arguments

`x`	SpatRaster or SpatVector
`wide`	logical. If `FALSE` each layer in the SpatRaster becomes a column in the matrix and each cell in the SpatRaster becomes a row. If `TRUE` each row in the SpatRaster becomes a row in the matrix and each column in the SpatRaster becomes a column in the matrix
`mode`	this argument is ignored
`...`	additional arguments (none implemented)

Value

vector, matrix, or array

See Also

`as.data.frame` and `as.polygons`

Examples

```
r <- rast(ncols=2, nrows=2)
values(r) <- 1:ncell(r)

as.vector(r)
as.matrix(r)
as.matrix(r, wide=TRUE)
as.data.frame(r, xy=TRUE)
as.array(r)

as.vector(ext(r))
as.matrix(ext(r))
```

`colors` *Color table*

Description

Get or set color table(s) associated with a SpatRaster. Color tables are used for associating colors with values, for use in mapping (plot).

Usage

```
## S4 method for signature 'SpatRaster'
coltab(x)

## S4 replacement method for signature 'SpatRaster'
coltab(x, ..., layer=1)<-value

## S4 method for signature 'SpatRaster'
has.colors(x)
```

Arguments

`x`	SpatRaster
`layer`	positive integer, the layer number or name
`value`	a two-column data.frame (first column the cell value, the second column the color); a vector of colors (the first one is the color for value 0 and so on); or a four (value,red,green,blue) or five (including alpha) column data.frame also from 0 to n; or NULL to remove the color table. You can also supply a list of such data.frames to set a color table to all layers
`...`	additional arguments (none implemented)

Value

data.frame

Examples

```
r <- rast(ncols=3, nrows=2, vals=1:6)
coltb <- data.frame(value=1:6, col=rainbow(6, end=.9))
coltb

plot(r)

has.colors(r)
coltab(r) <- coltb
plot(r)
has.colors(r)

tb <- coltab(r)
class(tb)
dim(tb[[1]])
```

`combineGeoms` *Combine geometries*

Description

Combine the geometries of one SpatVector with those of another. Geometries can be combined based on overlap, shared boundaries and distance (in that order of operation).

The typical use-case of this method is when you are editing geometries and you have a number of small polygons in one SpatVector that should be part of the geometries of the another SpatVector; perhaps because they were small holes inbetween the borders of two SpatVectors.

To append SpatVectors use ‘rbind‘ and see methods like `intersect` and `union` for "normal" polygons combinations.

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
combineGeoms(x, y, overlap=TRUE, boundary=TRUE, distance=TRUE,
append=TRUE, minover=0.1, maxdist=Inf, dissolve=TRUE, erase=TRUE)
```

Arguments

`x`	SpatVector of polygons
`y`	SpatVector of polygons geometries that are to be combined with `x`
`overlap`	logical. If `TRUE`, a geometry is combined with the geometry it has most overlap with, if the overlap is above `minover`
`boundary`	logical. If `TRUE`, a geometry is combined with the geometry it has most shared border with
`distance`	logical. If `TRUE`, a geometry is combined with the geometry it is nearest to
`append`	logical. Should remaining geometries be appended to the output? Not relevant if `distance=TRUE`
`minover`	numeric. The fraction of the geometry in `y` that overlaps with a geometry in `x`. Below this threshold, geometries are not considered overlapping
`maxdist`	numeric. Geometries further away from each other than this distance (in meters) will not be combined
`dissolve`	logical. Should internal boundaries be dissolved?
`erase`	logical. If `TRUE` no new overlapping areas are created

Value

SpatVector

See Also

`union`, `erase`, `intersect`

`sharedPaths`, `erase`, `intersect`

Examples

```
x1 <- vect("POLYGON ((0 0, 8 0, 8 9, 0 9, 0 0))")
x2 <- vect("POLYGON ((10 4, 12 4, 12 7, 11 7, 11 6, 10 6, 10 4))")

y1 <- vect("POLYGON ((5 6, 15 6, 15 15, 5 15, 5 6))")
y2 <- vect("POLYGON ((8 2, 9 2, 9 3, 8 3, 8 2))")
y3 <- vect("POLYGON ((2 6, 3 6, 3 8, 2 8, 2 6))")
y4 <- vect("POLYGON ((2 12, 3 12, 3 13, 2 13, 2 12))")

x <- rbind(x1, x2)
values(x) <- data.frame(xid=1:2)
crs(x) <- "+proj=utm +zone=1"

y <- rbind(y1, y2, y3, y4)
```

```
values(y) <- data.frame(yid=letters[1:4])
crs(y) <- "+proj=utm +zone=1"

plot(rbind(x, y), border=c(rep("red",2), rep("blue", 4)), lwd=2)
text(x, "xid")
text(y, "yid")

v <- combineGeoms(x, y)
plot(v, col=c("red", "blue"))

v <- combineGeoms(x, y, boundary=FALSE, maxdist=1, minover=.05)
plot(v, col=rainbow(4))
```

Compare-methods *Compare and logical methods*

Description

Standard comparison and logical operators for computations with SpatRasters. Computations are local (applied on a cell by cell basis). If multiple SpatRasters are used, these must have the same geometry (extent and resolution). These operators have been implemented:

Logical: `!`, `&`, `|`, `isTRUE`, `isFALSE`

Compare: `==`, `!=`, `>`, `<`, `<=`, `>=`, `is.na`, `is.nan`, `is.finite`, `is.infinite`

See `not.na` for the inverse of `is.na`, and `noNA` to detect cells with missing value across layers.

The `compare` and `logic` methods implement these operators in a method that can return `NA` istead of `FALSE` and allows for setting an output filename.

The terra package does not distinguish between `NA` (not available) and `NaN` (not a number). In most cases this state is represented by `NaN`.

If you use a SpatRaster with a vector of multiple numbers, each element in the vector is considered a layer (with a constant value). If you use a SpatRaster with a matrix, the number of columns of the matrix must match the number of layers of the SpatRaster. The rows are used to match the cells. That is, if there are two rows, these match cells 1 and 2, and they are recycled to 3 and 4, etc.

The following method has been implemented for **(SpatExtent, SpatExtent)**: ==

Usage

```
## S4 method for signature 'SpatRaster'
compare(x, y, oper, falseNA=FALSE, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatRaster'
logic(x, oper, falseNA=FALSE, filename="", overwrite=FALSE, ...)
```

Arguments

x	SpatRaster
y	SpatRaster or numeric
oper	character. Operator name. For compare this can be one of "==", "!=", ">", "<", ">=", "<=" and for logic it can be one of "!", "is.na", "allNA", "noNA", "is.infinite", "is.finite", "iSTRUE", "isFALSE"
falseNA	logical. Should the result be TRUE, NA instead of TRUE, FALSE?
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
...	additional arguments for writing files as in writeRaster

Value

SpatRaster or SpatExtent

See Also

all.equal, Arith-methods. See ifel to conveniently combine operations and Math-methods or app to apply any R function to a SpatRaster.

Examples

```
r1 <- rast(ncols=10, nrows=10)
values(r1) <- runif(ncell(r1))
r1[10:20] <- NA
r2 <- rast(r1)
values(r2) <- 1:ncell(r2) / ncell(r2)

x <- is.na(r1)
!x
r1 == r2
compare(r1, r2, "==")
compare(r1, r2, "==", TRUE)
```

---

compareGeom *Compare geometries*

Description

Evaluate whether two SpatRasters have the same extent, number of rows and columns, projection, resolution, and origin (or a subset of these comparisons).

Or evaluate whether two SpatVectors have the same geometries, or whether a SpatVector has duplicated geometries.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
compareGeom(x, y, ..., lyrs=FALSE, crs=TRUE, warncrs=FALSE, ext=TRUE,
rowcol=TRUE, res=FALSE, stopOnError=TRUE, messages=FALSE)

## S4 method for signature 'SpatVector,SpatVector'
compareGeom(x, y, tolerance=0)

## S4 method for signature 'SpatVector,missing'
compareGeom(x, y, tolerance=0)
```

Arguments

`x`	SpatRaster or SpatVector
`y`	Same as `x`. If `x` is a SpatRaster, `y` can also be a list of SpatRasters. If `x` is a SpatVector, `y` can be missing
`...`	Additional SpatRasters
`lyrs`	logical. If `TRUE`, the number of layers is compared
`crs`	logical. If `TRUE`, coordinate reference systems are compared
`warncrs`	logical. If `TRUE`, a warning is given if the crs is different (instead of an error)
`ext`	logical. If `TRUE`, bounding boxes are compared
`rowcol`	logical. If `TRUE`, number of rows and columns of the objects are compared
`res`	logical. If `TRUE`, resolutions are compared (redundant when checking extent and rowcol)
`stopOnError`	logical. If `TRUE`, code execution stops if raster do not match
`messages`	logical. If `TRUE`, warning/error messages are printed even if `stopOnError=FALSE`
`tolerance`	numeric

Value

logical (SpatRaster) or matrix of logical (SpatVector)

Examples

```
r1 <- rast()
r2 <- rast()
r3 <- rast()
compareGeom(r1, r2, r3)
nrow(r3) <- 10


## Not run:
compareGeom(r1, r3)

## End(Not run)
```

`concats` *Concatenate categorical rasters*

Description

Combine two categorical rasters by concatenating their levels.

Usage

```
## S4 method for signature 'SpatRaster'
concats(x, y, filename="", ...)
```

Arguments

`x`	SpatRaster (with a single, categorical, layer)
`y`	SpatRaster (with a single, categorical, layer)
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`cats`

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE)
levels(r) <- data.frame(id=1:3, cover=c("forest", "water", "urban"))

rr <- rast(r)
values(rr) <- sample(1:3, ncell(rr), replace=TRUE)
levels(rr) <- data.frame(id=c(1:3), color=c("red", "green", "blue"))

x <- concats(r, rr)
x
levels(x)[[1]]
```

contour *Contour plot*

Description

Contour lines (isolines) of a SpatRaster. Use add=TRUE to add the lines to the current plot. See graphics::contour for details.

if filled=TRUE, a new filled contour plot is made. See graphics::filled.contour for details.

as.contour returns the contour lines as a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster'
contour(x, maxcells=100000, filled=FALSE, ...)

## S4 method for signature 'SpatRaster'
as.contour(x, maxcells=100000, ...)
```

Arguments

x	SpatRaster. Only the first layer is used
maxcells	maximum number of pixels used to create the contours
filled	logical. If TRUE, a filled.contour plot is made
...	any argument that can be passed to contour or filled.contour (graphics package)

See Also

plot

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
plot(r)
contour(r, add=TRUE)

v <- as.contour(r)
plot(r)
lines(v)

contour(r, filled=TRUE, nlevels=5)

## if you want a SpatVector with contour lines
template <- disagg(rast(r), 10)
rr <- resample(r, template)
rr <- floor(rr/100) * 100
v <- as.polygons(rr)
```

```
plot(v, 1, col=terrain.colors(7))

## to combine filled contours with contour lines (or other spatial data)

br <- seq(100, 600, 100)
plot(r, breaks=br)
lines(as.contour(r, levels=br))

## or
x <- classify(r,  br) |> as.polygons()
plot(x, "elevation")
```

---

convHull   *Convex hull, minimal bounding rotated rectangle, and minimal bounding circle*

Description

Get the convex hull, the minimal bounding rotated rectangle, or minimal bounding circle of a SpatVector

Usage

```
## S4 method for signature 'SpatVector'
convHull(x, by="")

## S4 method for signature 'SpatVector'
minRect(x, by="")

## S4 method for signature 'SpatVector'
minCircle(x, by="")
```

Arguments

x	SpatVector
by	character (variable name), to get a new geometry for groups of input geometries

Value

SpatVector

Examples

```
p <- vect(system.file("ex/lux.shp", package="terra"))
h <- convHull(p)

hh <- convHull(p, "NAME_1")
rr <- minRect(p, "NAME_1")
```

```
plot(rr, lwd=5, border="gray")
plot(hh, "NAME_1", col=rainbow(10, alpha=.5), lwd=3, add=TRUE, plg=list(x="topright"))
lines(aggregate(p, "NAME_1"), col="blue", lty=2, lwd=2)
```

`costDist` *Cost-distance*

Description

Use a friction (cost) surface to compute the cost-distance from any cell to the border of one or more target cells.

Distances are computed by summing local distances between cells, which are connected with their neighbors in 8 directions, and assuming that the path has to go through the centers of one of the neighboring raster cells.

Distances are multiplied with the friction, thus to get the cost-distance, the friction surface must express the cost per unit distance (speed) of travel.

Usage

```
## S4 method for signature 'SpatRaster'
costDist(x, target=0, scale=1, maxiter=50, filename="", ...)
```

Arguments

`x`	SpatRaster
`target`	numeric. value of the target cells (where to compute cost-distance to)
`scale`	numeric. Scale factor. The cost distance is divided by this number
`maxiter`	numeric. The maximum number of iterations. Increase this number if you get the warning that `costDistance did not converge`
`filename`	character. output filename (optional)
`...`	additional arguments as for `writeRaster`

Value

SpatRaster

See Also

`gridDist`, `distance`

Examples

```
r <- rast(ncols=5, nrows=5, crs="+proj=utm +zone=1 +datum=WGS84",
xmin=0, xmax=5, ymin=0, ymax=5, vals=1)
r[13] <- 0
d <- costDist(r)
plot(d)
text(d, digits=1)

r <- rast(ncols=10, nrows=10,  xmin=0, xmax=10, ymin=0, ymax=10,
   vals=10, crs="+proj=utm +zone=1 +datum=WGS84")
r[5, 1] <- -10
r[2:3, 1] <- r[1, 2:4] <- r[2, 5] <- 0
r[3, 6] <- r[2, 7] <- r[1, 8:9] <- 0
r[6, 6:10] <- NA
r[6:9, 6] <- NA

d <- costDist(r, -10)
plot(d)
text(d, digits=1, cex=.8)
```

`cover` *Replace values with values from another object*

Description

Replace `NA` or other values in SpatRaster `x` with the values of SpatRaster `y`

For polygons: areas of `x` that overlap with `y` are replaced by `y` or, if `identity=TRUE` intersected with `y`.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
cover(x, y, values=NA, filename="", ...)

## S4 method for signature 'SpatVector,SpatVector'
cover(x, y, identity=FALSE, expand=TRUE)
```

Arguments

`x`	SpatRaster or SpatVector
`y`	Same as `x`
`values`	numeric. The cell values in `x` to be replaced by the values in `y`
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`
`identity`	logical. If `TRUE` overlapping areas are intersected rather than replaced
`expand`	logical. Should parts of `y` that are outside of `x` be included?

Value

SpatRaster

Examples

```
r1 <- r2 <- rast(ncols=36, nrows=18)
values(r1) <- 1:ncell(r1)
values(r2) <- runif(ncell(r2))
r2 <- classify(r2, cbind(-Inf, 0.5, NA))
r3 <- cover(r2, r1)


p <- vect(system.file("ex/lux.shp", package="terra"))
e <- as.polygons(ext(6, 6.4, 49.75, 50))
values(e) <- data.frame(y=10)

cv <- cover(p, e)
plot(cv, col=rainbow(12))
ci <- cover(p, e, identity=TRUE)
lines(e, lwd=3)

plot(ci, col=rainbow(12))
lines(e, lwd=3)
```

---

crds *Get the coordinates of SpatVector geometries or SpatRaster cells*

Description

Get the coordinates of a SpatVector or SpatRaster cells. A matrix or data.frame of the x (longitude) and y (latitude) coordinates is returned.

Usage

```
## S4 method for signature 'SpatVector'
crds(x, df=FALSE, list=FALSE)

## S4 method for signature 'SpatRaster'
crds(x, df=FALSE, na.rm=TRUE, na.all=FALSE)
```

Arguments

x	SpatRaster or SpatVector
df	logical. If `TRUE` a `data.frame` is returned instead of a matrix
list	logical. If `TRUE` a `list` is returned instead of a matrix
na.rm	logical. If `TRUE` cells that are `NA` are excluded. Ignored if the SpatRaster is a template with no associated cell values

na.all — logical. If TRUE cells are only ignored if na.rm=TRUE and their value is NA for **all** layers instead of for any layer

Value

matrix or data.frame

See Also

geom returns the complete structure of SpatVector geometries. For SpatRaster see xyFromCell

Examples

```
x1 <- rbind(c(-175,-20), c(-140,55), c(10, 0), c(-140,-60))
x2 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))
x3 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
x4 <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))
z <- rbind(cbind(object=1, part=1, x1), cbind(object=2, part=1, x2),
           cbind(object=3, part=1, x3), cbind(object=3, part=2, x4))
colnames(z)[3:4] <- c('x', 'y')
z <- cbind(z, hole=0)
z[(z[, "object"]==3 & z[,"part"]==2), "hole"] <- 1

p <- vect(z, "polygons")
crds(p)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
g <- crds(v)
head(g)
```

---

crop — *Cut out a geographic subset*

Description

Cut out a part of a SpatRaster or SpatVector.

You can crop a SpatRaster with a SpatExtent, or with another object from which an extent can be obtained. Note that the SpatRaster returned may not have the exactly the same extent as the SpatExtent supplied because you can only select entire cells (rows and columns), and you cannot add new areas. See methods like resample and disagg to force SpatRasters to align and extend to add rows and/or columns.

You can only crop rectangular areas of a SpatRaster, but see argument mask=TRUE for setting cell values within SpatRaster to NA; or use the mask method after crop for additional masking options.

You can crop a SpatVector with another SpatVector. If these are not polygons, the minimum convex hull is used. Unlike with intersect the geometries and attributes of y are not transferred to the output. You can also crop a SpatVector with a rectangle (SpatRaster, SpatExtent).

Usage

```
## S4 method for signature 'SpatRaster'
crop(x, y, snap="near", mask=FALSE, touches=TRUE, extend=FALSE, filename="", ...)

## S4 method for signature 'SpatRasterDataset'
crop(x, y, snap="near", extend=FALSE)

## S4 method for signature 'SpatRasterCollection'
crop(x, y, snap="near", extend=FALSE)

## S4 method for signature 'SpatVector'
crop(x, y, ext=FALSE)

## S4 method for signature 'SpatGraticule'
crop(x, y)
```

Arguments

`x`	SpatRaster or SpatVector
`y`	SpatRaster, SpatVector, SpatExtent, or any other object that has a SpatExtent (`ext` returns a `SpatExtent`)
`snap`	character. One of "near", "in", or "out". Used to align `y` to the geometry of `x`
`mask`	logical. Should `y` be used to mask? Only used if `y` is a SpatVector, SpatRaster or sf
`touches`	logical. If `TRUE` and `mask=TRUE`, all cells touched by lines or polygons will be masked, not just those on the line render path, or whose center point is within the polygon
`extend`	logical. Should rows and/or columns be added if `y` is beyond the extent of `x`? Also see `extend`
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`
`ext`	logical. Use the extent of `y` instead of `y`. This also changes the behavior when `y` is an extent in two ways: (1) points that are on the extent boundary are removed and (2) lon/lat extents that go beyond -180 or 180 degrees longitude are wrapped around the earth to include areas at the other end of the dateline

Value

SpatRaster

See Also

`intersect`, `extend`

See `window` for a virtual and sometimes more efficient way to crop a dataset.

Examples

```
r <- rast(xmin=0, xmax=10, ymin=0, ymax=10, nrows=25, ncols=25)
values(r) <- 1:ncell(r)
e <- ext(-5, 5, -5, 5)
rc <- crop(r, e)

# crop and mask
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
cm <- crop(r, v[9:12,], mask=TRUE)
plot(cm)
lines(v)

# crop vector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
e <- ext(6.15, 6.3, 49.7, 49.8)
x <- crop(v, e)
plot(x, "NAME_1")
```

---

crosstab *Cross-tabulate*

Description

Cross-tabulate the layers of a SpatRaster to create a contingency table.

Usage

```
## S4 method for signature 'SpatRaster,missing'
crosstab(x, digits=0, long=FALSE, useNA=FALSE)
```

Arguments

`x`	SpatRaster
`digits`	integer. The number of digits for rounding the values before cross-tabulation
`long`	logical. If `TRUE` the results are returned in 'long' format data.frame instead of a table
`useNA`	logical, indicting if the table should includes counts of `NA` values

Value

A table or data.frame

See Also

`freq`, `zonal`

Examples

```
r <- s <- rast(nc=5, nr=5)
set.seed(1)
values(r) <- runif(ncell(r)) * 2
values(s) <- runif(ncell(r)) * 3
x <- c(r, s)

crosstab(x)

rs <- r/s
r[1:5] <- NA
s[20:25] <- NA
x <- c(r, s, rs)
crosstab(x, useNA=TRUE, long=TRUE)
```

---

crs *Get or set a coordinate reference system*

Description

Get or set the coordinate reference system (CRS), also referred to as a "projection", of a SpatRaster or SpatVector.

Setting a new CRS does not change the data itself, it just changes the label. So you should only set the CRS of a dataset (if it does not come with one) to what it *is*, not to what you would *like it to be*. See `project` to *transform* an object from one CRS to another.

Usage

```
## S4 method for signature 'SpatRaster'
crs(x, proj=FALSE, describe=FALSE, parse=FALSE)

## S4 method for signature 'SpatVector'
crs(x, proj=FALSE, describe=FALSE, parse=FALSE)

## S4 method for signature 'character'
crs(x, proj=FALSE, describe=FALSE, parse=FALSE)

## S4 replacement method for signature 'SpatRaster'
crs(x, warn=FALSE)<-value

## S4 replacement method for signature 'SpatVector'
crs(x, warn=FALSE)<-value
```

Arguments

`x`	SpatRaster or SpatVector
`proj`	logical. If `TRUE` the crs is returned in PROJ-string notation

describe	logical. If TRUE the name, EPSG code, and the name and extent of the area of use are returned if known
warn	logical. If TRUE, a message is printed when the object already has a non-empty crs
value	character string describing a coordinate reference system. This can be in a WKT format, as a <authority:number> code such as "EPSG:4326", or a PROJ-string format such as "+proj=utm +zone=12" (see Note)
parse	logical. If TRUE, wkt parts are parsed into a vector (each line becomes an element)

Value

character or modified SpatRaster/Vector

Note

Projections are handled by the PROJ/GDAL libraries. The PROJ developers suggest to define a CRS with the WKT2 or <authority>:<code> notation. It is not practical to define one's own custom CRS with WKT2, and the the <authority>:<code> system only covers a handful of (commonly used) CRSs. To work around this problem it is still possible to use the deprecated PROJ-string notation (+proj=...) with one major caveat: the datum should be WGS84 (or the equivalent NAD83) – if you want to transform your data to a coordinate reference system with a different datum. Thus as long as you use WGS84, or an ellipsoid instead of a datum, you can safely use PROJ-strings to represent your CRS; including to define your own custom CRS.

You can also set the crs to "local" to get an informal coordinate system on an arbitrary Euclidean (Cartesian) plane with units in meter.

Examples

```
r <- rast()
crs(r)
crs(r, describe=TRUE, proj=TRUE)

crs(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"
crs(r)

# You can use epsg codes
crs(r)  <- "epsg:25831"
crs(r, describe=TRUE)$area

crs("epsg:25831", describe=TRUE)
```

datatype — *Data type of a SpatRaster or SpatVector*

Description

Get the data types of the fields (attributes, variables) of a SpatVector or of the file(s) associated with a SpatRaster. A (layer of a) SpatRaster has no datatype if it has no values, or if the values are in memory.

Usage

```
## S4 method for signature 'SpatRaster'
datatype(x, bylyr=TRUE)

## S4 method for signature 'SpatVector'
datatype(x)
```

Arguments

`x`	SpatRaster or SpatVector
`bylyr`	logical. If `TRUE` a value is returned for each layer. Otherwise, a value is returned for each data source (such as a file)

Value

character

See Also

`Raster data types` to check / set the type of SpatRaster values.

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
datatype(v)

f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
datatype(r)

# no data type
datatype(rast())
```

deepcopy *Deep copy*

Description

Make a deep copy of a SpatRaster or SpatVector. This is occasionally useful when wanting to use a replacement function in a shallow copy. That is a copy that was created like this: `x <- y`. If you use a replacement function to change an object, its shallow copies also change.

Usage

```
## S4 method for signature 'SpatRaster'
deepcopy(x)

## S4 method for signature 'SpatVector'
deepcopy(x)
```

Arguments

x	SpatRaster or SpatVector

Value

Same as `x`

Examples

```
r <- rast(ncols=10, nrows=10, nl=3)
tm <- as.Date("2001-05-03") + 1:3
time(r) <- tm
time(r)
x <- r
time(x) <- tm + 365
time(x)
time(r)

y <- deepcopy(r)
time(y) <- tm - 365
time(y)
time(r)

# or make a new object like this
z <- rast(r)
time(z) <- tm
time(z)
time(r)
```

densify *Add additional nodes to lines or polygons*

Description

Add additional nodes to lines or polygons. This can be useful to do prior to using `project` such that the path does not change too much.

Usage

```
## S4 method for signature 'SpatVector'
densify(x, interval, equalize=TRUE, flat=FALSE)
```

Arguments

`x`	SpatVector
`interval`	positive number, specifying the desired minimum distance between nodes. The unit is meter for lonlat data, and in the linear unit of the crs for planar data
`equalize`	logical. If `TRUE`, new nodes are spread at equal intervals between old nodes
`flat`	logical. If `TRUE`, the earth's curvature is ignored for lonlat data, and the distance unit is degrees, not meter

Value

SpatVector

Examples

```
v <- vect(rbind(c(-120,-20), c(-80,5), c(-40,-60), c(-120,-20)),
  type="polygons", crs="+proj=longlat")
vd <- densify(v, 200000)

p <- project(v, "+proj=robin")
pd <- project(vd, "+proj=robin")

# good
plot(pd, col="gray", border="red", lwd=10)
points(pd, col="gray")

# bad
lines(p, col="blue", lwd=3)
points(p, col="blue", cex=2)
plot(p, col="blue", alpha=.1, add=TRUE)
legend("topright", c("good", "bad"), col=c("red", "blue"), lty=1, lwd=3)

## the other way around does not work
## unless the original data was truly planar (e.g. derived from a map)
x <- densify(p, 250000)
```

```
y <- project(x, "+proj=longlat")
# bad
plot(y)
# good
lines(vd, col="red")
```

---

density *Density plot*

Description

Create density plots of the cell values of a SpatRaster

Usage

```
## S4 method for signature 'SpatRaster'
density(x, maxcells=100000, plot=TRUE, main, ...)
```

Arguments

`x`	SpatRaster
`maxcells`	the maximum number of (randomly sampled) cells to be used for creating the plot
`plot`	if `TRUE` produce a plot, else return a density object
`main`	character. Caption of plot(s)
`...`	additional arguments passed to `plot`

Value

density plot (and a density object, returned invisibly if `plot=TRUE`)

Examples

```
logo <- rast(system.file("ex/logo.tif", package="terra"))
density(logo)
```

deprecated *deprecated methods*

Description

These methods are deprecated and will be removed in future versions

Usage

```
## S4 method for signature 'SpatRaster'
gridDistance(x, ...)
```

Arguments

x	object
...	additional arguments

depth *depth of SpatRaster layers*

Description

Get or set the depth of the layers of a SpatRaster. Experimental.

Usage

```
## S4 method for signature 'SpatRaster'
depth(x)

## S4 replacement method for signature 'SpatRaster'
depth(x)<-value
```

Arguments

x	SpatRaster
value	numeric vector

Value

numeric

See Also

time

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))

depth(s) <- 1:3
depth(s)
```

---

describe *describe*

Description

Describe the properties of spatial data in a file as generated with the "GDALinfo" tool.

Usage

```
## S4 method for signature 'character'
describe(x, sds=FALSE, meta=FALSE, parse=FALSE, options="", print=FALSE, open_opt="")
```

Arguments

`x`	character. The name of a file with spatial data. Or a fully specified subdataset within a file such as `"NETCDF:\"AVHRR.nc\":NDVI"`
`sds`	logical. If `TRUE` the description or metadata of the subdatasets is returned (if available)
`meta`	logical. Get the file level metadata instead
`parse`	logical. If `TRUE`, metadata for subdatasets is parsed into components (if `meta=TRUE`)
`options`	character. A vector of valid options (if `meta=FALSE`) including "json", "mm", "stats", "hist", "nogcp", "nomd", "norat", "noct", "nofl", "checksum", "proj4", "listmdd", "mdd <value>" where <value> specifies a domain or 'all', "wkt_format <value>" where value is one of 'WKT1', 'WKT2', 'WKT2_2015', or 'WKT2_2018', "sd <subdataset>" where <subdataset> is the name or identifier of a sub-dataset. See https://gdal.org/en/latest/programs/gdalinfo.html. Ignored if `sds=TRUE`
`print`	logical. If `TRUE`, print the results
`open_opt`	character. Driver specific open options

Value

character (invisibly, if `print=FALSE`)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
describe(f)
describe(f, meta=TRUE)
#g <- describe(f, options=c("json", "nomd", "proj4"))
#head(g)
```

diff	*Lagged differences*

Description

Compute the difference between consecutive layers in a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
diff(x, lag=1, filename="", ...)
```

Arguments

`x`	SpatRaster
`lag`	positive integer indicating which lag to use
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
d <- diff(s)
```

dimensions	*Dimensions of a SpatRaster or SpatVector and related objects*

Description

Get the number of rows (`nrow`), columns (`ncol`), cells (`ncell`), layers (`nlyr`), sources (`nsrc`), the size `size` (`nlyr(x)*ncell(x)`), or spatial resolution of a SpatRaster.

`length` returns the number of sub-datasets in a SpatRasterDataset or SpatVectorCollection.

For a SpatVector `length(x)` is the same as `nrow(x)`.

You can also set the number of rows or columns or layers. When setting dimensions, all cell values are dropped.

Usage

```
## S4 method for signature 'SpatRaster'
ncol(x)

## S4 method for signature 'SpatRaster'
nrow(x)

## S4 method for signature 'SpatRaster'
nlyr(x)

## S4 method for signature 'SpatRaster'
ncell(x)

## S4 method for signature 'SpatRaster'
nsrc(x)

## S4 replacement method for signature 'SpatRaster,numeric'
ncol(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
nrow(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
nlyr(x)<-value

## S4 method for signature 'SpatRaster'
res(x)

## S4 replacement method for signature 'SpatRaster,numeric'
res(x)<-value

## S4 method for signature 'SpatRaster'
xres(x)

## S4 method for signature 'SpatRaster'
yres(x)

## S4 method for signature 'SpatVector'
ncol(x)

## S4 method for signature 'SpatVector'
nrow(x)

## S4 method for signature 'SpatVector'
length(x)
```

Arguments

`x`	SpatRaster or SpatVector or related objects
`value`	For ncol and nrow: positive integer. For res: one or two positive numbers

Value

integer

See Also

ext

Examples

```
r <- rast()
ncol(r)
nrow(r)
nlyr(r)
dim(r)
nsrc(r)
ncell(r)

rr  <- c(r,r)
nlyr(rr)
nsrc(rr)
ncell(rr)

nrow(r) <- 18
ncol(r) <- 36
# equivalent to
dim(r) <- c(18, 36)

dim(r)
dim(r) <- c(10, 10, 5)
dim(r)

xres(r)
yres(r)
res(r)

res(r) <- 1/120
# different xres and yres
res(r) <- c(1/120, 1/60)
```

`direction` *Direction*

Description

The direction (azimuth) to or from the nearest cell that is not NA. The direction is expressed in radians, unless you use argument degrees=TRUE.

Usage

```
## S4 method for signature 'SpatRaster'
direction(x, from=FALSE, degrees=FALSE, filename="", ...)
```

Arguments

x	SpatRaster
filename	Character. Output filename (optional)
degrees	Logical. If FALSE (the default) the unit of direction is radians.
from	Logical. Default is FALSE. If TRUE, the direction from (instead of to) the nearest cell that is not NA is returned
...	Additional arguments as for writeRaster

Value

SpatRaster

See Also

distance

Examples

```
r <- rast(ncol=36,nrow=18, crs="+proj=merc")
values(r) <- NA
r[306] <- 1
b <- direction(r, degrees=TRUE)
plot(b)

crs(r) <- "+proj=longlat"
b <- direction(r)
plot(b)
```

---

disagg *Disaggregate raster cells or vector geometries*

Description

SpatRaster: Create a SpatRaster with a higher resolution (smaller cells). The values in the new SpatRaster are the same as in the larger original cells.

SpatVector: Separate multi-objects (points, lines, polygons) into single objects; or further into segments (for lines or polygons).

Usage

```
## S4 method for signature 'SpatRaster'
disagg(x, fact, method="near", filename="", ...)

## S4 method for signature 'SpatVector'
disagg(x, segments=FALSE)
```

Arguments

`x`	SpatRaster or SpatVector
`fact`	positive integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers)
`method`	character. Either "near" for nearest or "bilinear" for bilinear interpolation
`segments`	logical. Should (poly-)lines or polygons be disaggregated into their line-segments?
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`aggregate`, `resample`

Examples

```
r <- rast(ncols=10, nrows=10)
rd <- disagg(r, fact=c(10, 2))
ncol(rd)
nrow(rd)
values(r) <- 1:ncell(r)
rd <- disagg(r, fact=c(4, 2))
```

`distance` *Geographic distance*

Description

If `x` is a **SpatRaster**:

If `y` is `missing` this method computes the distance, for all cells that are `NA` in SpatRaster `x` to the nearest cell that is not `NA` (or other values, see arguments "target" and "exclude").

If `y` is a numeric value, the cells with that value are ignored. That is, distance to or from these cells is not computed (only if `grid=FALSE`).

If y is a SpatVector, the distance to that SpatVector is computed for all cells. For lines and polygons this is done after rasterization; and only the overlapping areas of the vector and raster are considered (for now).

The distance is always expressed in meter if the coordinate reference system is longitude/latitude, and in map units otherwise. Map units are typically meter, but inspect `crs(x)` if in doubt.

Results are more precise, sometimes much more precise, when using longitude/latitude rather than a planar coordinate reference system, as these distort distance.

If x is a **SpatVector**:

If y is `missing`, a distance matrix between all objects in x is computed. A distance matrix object of class "dist" is returned.

If y is a SpatVector the geographic distance between all objects is computed (and a matrix is returned). If both sets have the same number of points, and `pairwise=TRUE`, the distance between each pair of objects is computed, and a vector is returned.

The distance is always expressed in meter, except when the coordinate reference system is longitude/latitude AND one of the SpatVector(s) consists of lines or polygons. In that case the distance is in degrees, and thus not very useful (this will be fixed soon). Otherwise, results are more precise, sometimes much more precise, when using longitude/latitude rather than a planar coordinate reference system, as these distort distance.

If x is a **matrix**:

x should consist of two columns, the first with "x" (or longitude) and the second with "y" coordinates (or latitude). If y is a also a matrix, the distance between each points in x and all points in y is computed, unless `pairwise=TRUE`

If y is missing, the distance between each points in x with all other points in x is computed, unless `sequential=TRUE`

Usage

```
## S4 method for signature 'SpatRaster,missing'
distance(x, y, target=NA, exclude=NULL, unit="m", haversine=TRUE, filename="", ...)

## S4 method for signature 'SpatRaster,SpatVector'
distance(x, y, unit="m", rasterize=FALSE, haversine=TRUE, filename="", ...)

## S4 method for signature 'SpatVector,ANY'
distance(x, y, sequential=FALSE, pairs=FALSE, symmetrical=TRUE, unit="m")

## S4 method for signature 'SpatVector,SpatVector'
distance(x, y, pairwise=FALSE, unit="m")

## S4 method for signature 'matrix,matrix'
distance(x, y, lonlat, pairwise=FALSE, unit="m")

## S4 method for signature 'matrix,missing'
distance(x, y, lonlat, sequential=FALSE, pairs=FALSE, symmetrical=TRUE, unit="m")
```

Arguments

`x`	SpatRaster, SpatVector, or two-column matrix with coordinates (x,y) or (lon,lat)
`y`	missing, numeric, SpatVector, or two-column matrix
`target`	numeric. The value of the cells for which distances to cells that are not `NA` should be computed
`exclude`	numeric. The value of the cells that should not be considered for computing distances
`unit`	character. Can be either "m" or "km"
`haversine`	logical. Use the haversine formula for lon/lat data use the haversine formula? If `FALSE`, the more precise but slower method of Karney (2003) is used
`rasterize`	logical. If `TRUE` distance is computed from the cells covered by the geometries after rasterization. This can be much faster in some cases
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`
`sequential`	logical. If `TRUE`, the distance between sequential geometries is returned
`pairwise`	logical. If `TRUE` and if x and y have the same size (number of rows), the pairwise distances are returned instead of the distances between all elements
`lonlat`	logical. If `TRUE` the coordinates are interpreted as angular (longitude/latitude). If `FALSE` they are interpreted as planar
`pairs`	logical. If `TRUE` a "from", "to", "distance" matrix is returned
`symmetrical`	logical. If `TRUE` and `pairs=TRUE`, the distance between a pair is only included once. The distance between geometry 1 and 3 is included, but the (same) distance between 3 and 1 is not

Value

SpatRaster or numeric or matrix or distance matrix (object of class "dist")

Note

A distance matrix can be coerced into a regular matrix with `as.matrix`

References

Karney, C.F.F., 2013. Algorithms for geodesics, J. Geodesy 87: 43-55. doi:10.1007/s00190-012-0578-z.

Examples

```
#lonlat
r <- rast(ncols=36, nrows=18, crs="+proj=longlat +datum=WGS84")
r[500] <- 1
d <- distance(r)
plot(d / 100000)
```

```
#planar
rr <- rast(ncols=36, nrows=18, crs="+proj=utm +zone=1 +datum=WGS84")
rr[500] <- 1
d <- distance(rr)

rr[3:10, 3:10] <- 99
e <- distance(rr, exclude=99)

p1 <- vect(rbind(c(0,0), c(90,30), c(-90,-30)), crs="+proj=longlat +datum=WGS84")
dp <- distance(r, p1)

d <- distance(p1)
d
as.matrix(d)

p2 <- vect(rbind(c(30,-30), c(25,40), c(-9,-3)), crs="+proj=longlat +datum=WGS84")
dd <- distance(p1, p2)
dd
pd <- distance(p1, p2, pairwise=TRUE)
pd
pd == diag(dd)


# polygons, lines
crs <- "+proj=utm +zone=1"
p1 <- vect("POLYGON ((0 0, 8 0, 8 9, 0 9, 0 0))", crs=crs)
p2 <- vect("POLYGON ((5 6, 15 6, 15 15, 5 15, 5 6))", crs=crs)
p3 <- vect("POLYGON ((2 12, 3 12, 3 13, 2 13, 2 12))", crs=crs)
p <- rbind(p1, p2, p3)
L1 <- vect("LINESTRING(1 11, 4 6, 10 6)", crs=crs)
L2 <- vect("LINESTRING(8 14, 12 10)", crs=crs)
L3 <- vect("LINESTRING(1 8, 12 14)", crs=crs)
lns <- rbind(L1, L2, L3)
pts <- vect(cbind(c(7,10,10), c(3,5,6)), crs=crs)

distance(p1,p3)
distance(p)
distance(p,pts)
distance(p,lns)
distance(pts,lns)
```

`dots` *Make a dot-density map*

Description

Create the dots for a dot-density map and add these to the current map. Dot-density maps are made to display count data. For example of population counts, where each dot represents n persons. The dots are returned as a `SpatVector`. It there is an active graphics device, the dots are added to it with `points`.

Usage

```
## S4 method for signature 'SpatVector'
dots(x, field, size,  ...)
```

Arguments

`x`	SpatVector
`field`	character of numeric indicating field name. Or numeric vector of the same length as `x`
`size`	positive number indicating the number of cases associated with each dot
`...`	graphical arguments passed to `points`

Value

SpatVector (invisibly)

See Also

`plot`, `cartogram`, `points`

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$population <- 1000*(1:12)^2
plot(v, lwd=3, col="light gray", border="white")
d <- dots(v, "population", 1000, col="red", cex=.75)
lines(v)
d
```

---

draw — *Draw a polygon, line, extent, or points*

Description

Draw on a plot (map) to get a SpatVector or SpatExtent object for later use. After calling the function, start clicking on the map. When you are done, press `ESC`. You can also preset the maximum number of clicks.

This does to work well on the default RStudio plotting device. To work around that, you can first run `dev.new(noRStudioGD = TRUE)` which will create a separate window for plotting, then use `plot()` followed by `click()` and click on the map.

Usage

```
## S4 method for signature 'character'
draw(x="extent", col="red", lwd=2, id=FALSE, n=1000, xpd=TRUE, ...)
```

Arguments

`x`	character. The type of object to draw. One of "extent", "polygon", "line", or "points"
`col`	the color to be used
`lwd`	the width of the lines to be drawn
`id`	logical. If `TRUE`, a numeric ID is shown on the map
`n`	the maximum number of clicks (does not apply when `x=="extent"` in which case `n` is always 2)
`xpd`	logical. If `TRUE`, you can draw outside the current plotting area
`...`	additional graphics arguments for drawing

Value

SpatVector or SpatExtent

See Also

`click`

`elongate` *elongate lines*

Description

Elongate SpatVector lines

Usage

```
## S4 method for signature 'SpatVector'
elongate(x, length=1, flat=FALSE)
```

Arguments

`x`	SpatVector
`length`	positive number indicating how much the lines should be elongated at each end. The unit is meter is the crs is lonlat and it is the same as the linear unit of the crs on other cases (also meter in most cases)
`flat`	logical. If `TRUE`, the earth's curvature is ignored for lonlat data, and the distance unit is degrees, not meter

Value

SpatVector

See Also

buffer, crop and erase

Examples

```
v <- vect(cbind(c(0,1,2), c(0,0,2)), "lines", crs="lonlat")
e <- elongate(v, 100000)
plot(e)
points(e)
geom(e)
```

---

erase *Erase parts of a SpatVector object*

Description

Erase parts of a SpatVector with another SpatVector or with a SpatExtent. You can also erase (parts of) polygons with the other polygons of the same SpatVector.

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
erase(x, y)

## S4 method for signature 'SpatVector,missing'
erase(x, sequential=TRUE)

## S4 method for signature 'SpatVector,SpatExtent'
erase(x, y)
```

Arguments

`x`	SpatVector
`y`	SpatVector or SpatExtent
`sequential`	logical. Should areas be erased sequentially? See Details

Details

If polygons are erased sequentially, everything that is covered by the first polygon is removed from all other polygons, then everything that is covered by (what is remaining of) the second polygon is removed, etc.

If polygons are not erased sequentially, all overlapping areas are erased and only the areas covered by a single geometry are returned.

Value

SpatVector or SpatExtent

See Also

crop and intersect for the inverse.

The equivalent for SpatRaster is mask

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

# polygons with polygons or extent

e <- ext(5.6, 6, 49.55, 49.7)
x <- erase(v, e)

p <- vect("POLYGON ((5.8 49.8, 6 49.9, 6.15 49.8, 6 49.6, 5.8 49.8))")
y <- erase(v, p)

# lines with polygons
lns <- as.lines(rast(v, ncol=10, nrow=10))[12:22]
eln <- erase(lns, v)
plot(v)
lines(lns, col='blue', lwd=4, lty=3)
lines(eln, col='red', lwd=2)


## self-erase
h <- convHull(v[-12], "NAME_1")
he <- erase(h)
plot(h, lwd=2, border="red", lty=2)
lines(he, col="gray", lwd=3)
```

---

expanse *Get the expanse (area) of individual polygons or for all (summed) raster cells*

Description

Compute the area covered by polygons or for all raster cells that are not NA.

This method computes areas for longitude/latitude rasters, as the size of the cells is constant in degrees, but not in square meters. But it can also be important if the coordinate reference system is planar, but not equal-area.

For vector data, the best way to compute area is to use the longitude/latitude CRS. This is contrary to (erroneous) popular belief that suggest that you should use a planar coordinate reference system. This is done automatically, if transform=TRUE.

Usage

```
## S4 method for signature 'SpatRaster'
expanse(x, unit="m", transform=TRUE, byValue=FALSE,
zones=NULL, wide=FALSE, usenames=FALSE)

## S4 method for signature 'SpatVector'
expanse(x, unit="m", transform=TRUE)
```

Arguments

x	SpatRaster or SpatVector
unit	character. Output unit of area. One of "m", "km", or "ha"
transform	logical. If TRUE, planar CRS are transformed to lon/lat for accuracy
byValue	logical. If TRUE, the area for each unique cell value is returned
zones	NULL or SpatRaster with the same geometry identifying zones in x
wide	logical. Should the results be in "wide" rather than "long" format?
usenames	logical. If TRUE layers are identified by their names instead of their numbers

Value

SpatRaster: data.frame with at least two columns ("layer" and "area") and possibly also "value" (if byValue is TRUE), and "zone" (if zones is TRUE). If x has no values, the total area of all cells is returned. Otherwise, the area of all cells that are not NA is returned.

SpatVector: numeric (one value for each (multi-) polygon geometry.

See Also

cellSize for a the size of individual cells of a raster, that can be summed with global or with zonal to get the area for different zones.

Examples

```
### SpatRaster
r <- rast(nrows=18, ncols=36)
v <- 1:ncell(r)
v[200:400] <- NA
values(r) <- v

# summed area in km2
expanse(r, unit="km")

# all cells
expanse(rast(r), unit="km")

r <- rast(ncols=90, nrows=45, ymin=-80, ymax=80)
m <- project(r, "+proj=merc")

expanse(m, unit="km")
```

```
expanse(m, unit="km", transform=FALSE)

m2 <- c(m, m)
values(m2) <- cbind(c(1,2,NA,NA), c(11:14))
expanse(m2, unit="km", byValue=TRUE, wide=TRUE)


v <- vect(system.file("ex/lux.shp", package="terra"))
r <- rast(system.file("ex/elev.tif", package="terra"))
r <- round((r-50)/100)
levels(r) <- data.frame(id=1:5, name=c("forest", "water", "urban", "crops", "grass"))
expanse(r, byValue=TRUE)

g <- rasterize(v, r, "NAME_1")
expanse(r, byValue=TRUE, zones=g, wide=TRUE)


### SpatVector
v <- vect(system.file("ex/lux.shp", package="terra"))

a <- expanse(v)
a
sum(a)
```

---

ext *Create, get or set a SpatExtent*

Description

Get a SpatExtent of a SpatRaster, SpatVector, or other spatial objects. Or create a SpatExtent from a vector (length=4; order=xmin, xmax, ymin, ymax)

You can set the extent of a SpatRaster, but you cannot set the extent of a SpatVector (see `rescale` for that). See `set.ext` to set the extent in place.

Usage

```
## S4 method for signature 'SpatRaster'
ext(x, cells=NULL)

## S4 method for signature 'SpatVector'
ext(x)

## S4 method for signature 'numeric'
ext(x, ..., xy=FALSE)

## S4 replacement method for signature 'SpatRaster,SpatExtent'
ext(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
ext(x)<-value
```

Arguments

`x`	SpatRaster
`cells`	positive integer (cell) numbers to subset the extent to area covered by these cells
`value`	SpatExtent, or numeric vector of length four (xmin, xmax, ymin, ymax)
`...`	if `x` is a single numeric value, additional numeric values for xmax, ymin, and ymax
`xy`	logical. Set this to `TRUE` to indicate that coordinates are in (xmin, ymin, xmax, ymax) order, instead of in the terra standard order of (xmin, xmax, ymin, ymax)

Value

A `SpatExtent` object.

Examples

```
r <- rast()
e <- ext(r)
as.vector(e)
as.character(e)

ext(r) <- c(0, 2.5, 0, 1.5)
r
er <- ext(r)

round(er)
# go "in"
floor(er)
# go "out"
ceiling(er)

ext(r) <- e
```

`extend` *Extend*

Description

Enlarge the spatial extent of a SpatRaster. See `crop` if you (also) want to remove rows or columns.

Note that you can only enlarge SpatRasters with entire rows and columns. Therefore, the extent of the output SpatRaster may not be exactly the same as the requested. Depending on argument `snap` it may be a bit smaller or larger.

You can also enlarge a SpatExtent with this method, or with an algebraic notation (see examples)

Usage

```
## S4 method for signature 'SpatRaster'
extend(x, y, snap="near", fill=NA, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatExtent'
extend(x, y)
```

Arguments

`x`	SpatRaster or SpatExtent
`y`	If `x` is a SpatRaster, `y` should be a SpatExtent, or an object from which it can be extracted (such as SpatRaster and SpatVector objects). Alternatively, you can provide one, two or four non-negative integers indicating the number of rows and columns that need to be added at each side (a single positive integer when the number of rows and columns to be added is equal; or 2 number (columns, rows), or four (left column, right column, bottom row, top row). If `x` is a SpatExtent, `y` should likewise be a numeric vector of 1, 2, or 4 elements
`snap`	character. One of "near", "in", or "out". Used to align `y` to the geometry of `x`
`fill`	numeric. The value used to for the new raster cells
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster or SpatExtent

See Also

`crop`, `merge`, `ext`, `resample`

Examples

```
r <- rast(xmin=-150, xmax=-120, ymin=30, ymax=60, ncols=36, nrows=18)
values(r) <- 1:ncell(r)
e <- ext(-180, -100, 40, 70)
re <- extend(r, e)

# extend with a number of rows and columns (at each side)
re2 <- extend(r, c(2,10))

# SpatExtent
e <- ext(r)
e
extend(e, 10)
extend(e, c(10, -10, 0, 20))
```

```
# add 10 columns / rows on all sides
e + 10
# double extent
e * 2
# increase extent by 25%
e * 1.25
```

extract *Extract values from a SpatRaster*

Description

Extract values from a SpatRaster for a set of locations. The locations can be a SpatVector (points, lines, polygons), a data.frame or matrix with (x, y) or (longitude, latitude – in that order!) coordinates, or a vector with cell numbers.

When argument `y` is a `SpatVector` the first column has the ID (record number) of the `SpatVector` used (unless you set `ID=FALSE`).

Alternatively, you can use `zonal` after using `rasterize` with a `SpatVector` (this may be more efficient in some cases).

Usage

```
## S4 method for signature 'SpatRaster,SpatVector'
extract(x, y, fun=NULL, method="simple", cells=FALSE, xy=FALSE,
    ID=TRUE, weights=FALSE, exact=FALSE, touches=is.lines(y), small=TRUE,
layer=NULL, bind=FALSE, raw=FALSE, ...)

## S4 method for signature 'SpatRaster,SpatExtent'
extract(x, y, cells=FALSE, xy=FALSE)

## S4 method for signature 'SpatRaster,matrix'
extract(x, y, cells=FALSE, method="simple")

## S4 method for signature 'SpatRaster,numeric'
extract(x, y, xy=FALSE, raw=FALSE)

## S4 method for signature 'SpatVector,SpatVector'
extract(x, y)
```

Arguments

`x`	SpatRaster or SpatVector of polygons
`y`	SpatVector (points, lines, or polygons). Alternatively, for points, a 2-column matrix or data.frame (x, y) or (lon, lat). Or a vector with cell numbers

`fun`	function to summarize the extracted data by line or polygon geometry. You can use `fun=table` to tabulate raster values for each line or polygon geometry. If `weights=TRUE` or `exact=TRUE` only `mean`, `sum`, `min`, `max` and `table` are accepted). Ignored if `y` has point geometry
`method`	character. method for extracting values with points ("simple" or "bilinear"). With "simple" values for the cell a point falls in are returned. With "bilinear" the returned values are interpolated from the values of the four nearest raster cells
`cells`	logical. If `TRUE` the cell numbers are also returned, unless `fun` is not `NULL`. Also see `cells`
`xy`	logical. If `TRUE` the coordinates of the cells are also returned, unless `fun` is not `NULL`. See `xyFromCell`
`ID`	logical. Should an ID column be added? If so, the first column returned has the IDs (record numbers) of `y`
`weights`	logical. If `TRUE` and `y` has polygons, the approximate fraction of each cell that is covered is returned as well, for example to compute a weighted mean
`exact`	logical. If `TRUE` and `y` has polygons, the exact fraction of each cell that is covered is returned as well, for example to compute a weighted mean
`touches`	logical. If `TRUE`, values for all cells touched by lines or polygons are extracted, not just those on the line render path, or whose center point is within the polygon. Not relevant for points; and always considered `TRUE` when `weights=TRUE` or `exact=TRUE`
`small`	logical. If `TRUE`, values for all cells in touched polygons are extracted if none of the cells center points is within the polygon; even if `touches=FALSE`
`layer`	character or numeric to select the layer to extract from for each geometry. If `layer` is a character it can be a name in `y` or a vector of layer names. If it is numeric, it must be integer values between `1` and `nlyr(x)`
`bind`	logical. If `TRUE`, a SpatVector is returned consisting of the input SpatVector `y` and the `cbind`-ed extracted values
`raw`	logical. If `TRUE`, a matrix is returned with the "raw" numeric cell values. If `FALSE`, a data.frame is returned and the cell values are transformed to factor, logical, or integer values, where appropriate
`...`	additional arguments to `fun` if `y` is a SpatVector. For example `na.rm=TRUE`. Or arguments passed to the `SpatRaster,SpatVector` method if `y` is a matrix (such as the `method` and `cells` arguments)

Value

data.frame, matrix or SpatVector

See Also

`values`, `zonal`, `extractAlong`, `extractRange`, `rapp`

Examples

```
r <- rast(ncols=5, nrows=5, xmin=0, xmax=5, ymin=0, ymax=5)
values(r) <- 1:25
xy <- rbind(c(0.5,0.5), c(2.5,2.5))
p <- vect(xy, crs="+proj=longlat +datum=WGS84")

extract(r, xy)
extract(r, p)

r[1,]
r[5]
r[,5]

r[c(0:2, 99:101)]

f <- system.file("ex/meuse.tif", package="terra")
r <- rast(f)

xy <- cbind(179000, 330000)
xy <- rbind(xy-100, xy, xy+1000)
extract(r, xy)

p <- vect(xy)
g <- geom(p)
g

extract(r, p)

x <- r + 10
extract(x, p)

i <- cellFromXY(r, xy)
x[i]
r[i]

y <- c(x,x*2,x*3)
y[i]

## extract with a polygon
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v <- v[1:2,]

rf <- system.file("ex/elev.tif", package="terra")
x <- rast(rf)
extract(x, v, mean, na.rm=TRUE)

z <- rast(v, resolution=.1, names="test")
values(z) <- 1:ncell(z)
e <- extract(z, v, ID=TRUE)
e
tapply(e[,2], e[,1], mean, na.rm=TRUE)
```

```
x <- c(z, z*2, z/3)
names(x) <- letters[1:3]

e <- extract(x, v, ID=TRUE)
de <- data.frame(e)
aggregate(de[,2:4], de[,1,drop=FALSE], mean)
```

---

extractAlong *extract values along lines*

Description

Extract raster values along a line. That is, the returned values are ordered along the line. That is not the case with extract

Usage

```
extractAlong(x, y, ID=TRUE, cells=FALSE, xy=FALSE, online=FALSE, bilinear=TRUE)
```

Arguments

x	SpatRaster
y	SpatVector with lines geometry
ID	logical. Should an ID column be added? If so, the first column returned has the IDs (record numbers) of input SpatVector y
cells	logical. If TRUE the cell numbers are also returned
xy	logical. If TRUE the coordinates of the cells traversed by y are also returned. See xyFromCell
online	logical. If TRUE the returned coordinates are snapped to y
bilinear	logical. If TRUE the returned raster values computed with bilinear interpolation from the nearest four cells. Only relevant if online=TRUE

Value

data.frame

See Also

extract

Examples

```
r <- rast(ncols=36, nrows=18, vals=1:(18*36))
cds1 <- rbind(c(-50,0), c(0,60), c(40,5), c(15,-45), c(-10,-25))
cds2 <- rbind(c(80,20), c(140,60), c(160,0), c(140,-55))
lines <- vect(list(cds1, cds2), "lines")

extractAlong(r, lines)
```

extractRange *Extract values for a range of layers from a SpatRaster*

Description

Extract values from a SpatRaster for a set of locations and a range of layers. To extract values for a single or all layers, use extract

Usage

```
## S4 method for signature 'SpatRaster'
extractRange(x, y, first, last, lyr_fun=NULL,
geom_fun=NULL, ID=FALSE, na.rm=TRUE, ...)
```

Arguments

x	SpatRaster
y	SpatVector (points, lines, or polygons). Alternatively, for points, a 2-column matrix or data.frame (x, y) or (lon, lat). Or a vector with cell numbers
first	layer name of number, indicating the first layer in the range of layers to be considered
last	layer name or number, indicating the last layer in the range to be considered
lyr_fun	function to summarize the extracted data across layers
geom_fun	function to summarize the extracted data for each line or polygon geometry. Ignored if y has point geometry
ID	logical. Should an ID column be added? If so, the first column returned has the IDs (record numbers) of y
na.rm	logical. Should missing values be ignored?
...	additional arguments passed to extract

Value

numeric or data.frame

See Also

extract

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
xy <- data.frame(c(50,80), c(30, 60))
extract(r, xy)
extract(r, xy, layer=c("red", "green"))
extractRange(r, xy, first=1:2, last=3:2, lyr_fun=sum)
```

extremes — *Get or compute the minimum and maximum cell values*

Description

The minimum and maximum value of a SpatRaster are returned or computed (from a file on disk if necessary) and stored in the object.

Usage

```
## S4 method for signature 'SpatRaster'
minmax(x, compute=FALSE)
## S4 method for signature 'SpatRaster'
hasMinMax(x)
## S4 method for signature 'SpatRaster'
setMinMax(x, force=FALSE)
```

Arguments

`x`	SpatRaster
`compute`	logical. If `TRUE` min and max values are computed if they are not available
`force`	logical. If `TRUE` min and max values are recomputed even if already available

Value

minmax: numeric matrix of minimum and maximum cell values by layer

hasMinMax: logical indicating whether the min and max values are available.

setMinMax: nothing. Used for the side-effect of computing the minimum and maximum values of a SpatRaster

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
minmax(r)
```

factors — *Categorical rasters*

Description

A SpatRaster layer can represent a categorical variable (factor). Like factors, SpatRaster categories are stored as integers that have an associated label.

The categories can be inspected with `levels` and `cats`. They are represented by a `data.frame` that must have two or more columns, the first one identifying the (integer) cell values and the other column(s) providing the category labels.

If there are multiple columns with categories, you can set the "active" category to choose the one you want to use.

`cats` returns the entire data.frame, whereas `levels` only return two columns: the index and the active category.

To set categories for the first layer of a SpatRaster, you can provide `levels<-` with a data.frame or a list with a data.frame. To set categories for multiple layers you can provide `levels<-` with a list with one element (that either has a `data.frame` or is `NULL`) for each layer. Use `categories` to set the categories for a specific layer or specific layers.

`droplevels` removes categories that are not used (declared but not present as values in the raster) if `levels=NULL`.

`addCats` adds additional categories to a layer that already is categorical. It adds new variables, not new levels of an existing categorical variable.

Usage

```
## S4 method for signature 'SpatRaster'
levels(x)

## S4 replacement method for signature 'SpatRaster'
levels(x)<-value

## S4 method for signature 'SpatRaster'
cats(x, layer)

## S4 method for signature 'SpatRaster'
categories(x, layer=1, value, active=1, ...)

## S4 method for signature 'SpatRaster'
droplevels(x, level=NULL, layer=1)

## S4 method for signature 'SpatRaster'
addCats(x, value, merge=FALSE, layer=1)
```

Arguments

`x`	SpatRaster
`layer`	the layer name or number (positive integer); or 0 for all layers
`value`	a data.frame (ID, category) that define the categories. Or `NULL` to remove them

`active`	positive integer, indicating the column in `value` to be used as the active category (zero based to skip the first column with the cell values; that is 1 is the second column in `value`)
`level`	the categories to remove for the layer specified with `layer`
`merge`	logical. If `TRUE`, the categories are combined with `merge` using the first column of `value` as ID. If `FALSE` the categories are combined with `cbind`
`...`	additional arguments (none)

Value

list of data.frames (levels, cats) or logical (is.factor)

See Also

`activeCat`, `catalyze`, `set.cats`, `as.factor`, `is.factor`

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE)
is.factor(r)

cls <- data.frame(id=1:3, cover=c("forest", "water", "urban"))
levels(r) <- cls
is.factor(r)
r

plot(r, col=c("green", "blue", "light gray"))
text(r, digits=3, cex=.75, halo=TRUE)

# raster starts at 3
x <- r + 2
is.factor(x)

# Multiple categories
d <- data.frame(id=3:5, cover=cls[,2], letters=letters[1:3], value=10:12)
levels(x) <- d
x

# get current index
activeCat(x)
# set index
activeCat(x) <- 3
activeCat(x)
activeCat(x) <- "letters"
plot(x, col=c("green", "blue", "light gray"))
text(x, digits=3, cex=.75, halo=TRUE)

r <- as.numeric(x)
r
```

```
p <- as.polygons(x)
plot(p, "letters", col=c("green", "blue", "light gray"))
```

---

fillHoles *Remove holes from polygons*

Description

Remove the holes in SpatVector polygons. If `inverse=TRUE` the holes are returned (as polygons).

Usage

```
## S4 method for signature 'SpatVector'
fillHoles(x, inverse=FALSE)
```

Arguments

`x`	SpatVector
`inverse`	logical. If `TRUE` the holes are returned as polygons

Value

SpatVector

Examples

```
x <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
hole <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))

z <- rbind(cbind(object=1, part=1, x, hole=0),
   cbind(object=1, part=1, hole, hole=1))
colnames(z)[3:4] <- c('x', 'y')
p <- vect(z, "polygons", atts=data.frame(id=1))
p

f <- fillHoles(p)
g <- fillHoles(p, inverse=TRUE)

plot(p, lwd=16, border="gray", col="light yellow")
polys(f, border="blue", lwd=3, density=4, col="orange")
polys(g, col="white", lwd=3)
```

fillTime *Fill time gaps in a SpatRaster*

Description

Add empty layers in between existing layers such that the time step between each layer is the same. See approximate to estimate values for these layer (and other missing values)

Usage

```
## S4 method for signature 'SpatRaster'
fillTime(x, filename="", ...)
```

Arguments

x	SpatRaster
filename	character. Output filename
...	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

approximate

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
s <- c(r, r)
time(s) <- as.Date("2001-01-01") + c(0:2, 5:7)
time(s)
ss <- fillTime(s)
time(ss)

a <- approximate(ss)
```

flip *Flip or reverse a raster*

Description

Flip the values of a SpatRaster by inverting the order of the rows (`vertical=TRUE`) or the columns (`vertical=FALSE`).

`rev` is the same as a horizontal *and* a vertical flip.

Usage

```
## S4 method for signature 'SpatRaster'
flip(x, direction="vertical", filename="", ...)

## S4 method for signature 'SpatVector'
flip(x, direction="vertical")

## S4 method for signature 'SpatRaster'
rev(x)
```

Arguments

`x`	SpatRaster or SpatVector
`direction`	character. Should (partially) match "vertical" to flip by rows, or "horizontal" to flip by columns
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`trans`, `rotate`

Examples

```
r <- rast(nrow=18, ncol=36)
m <- matrix(1:ncell(r), nrow=18)
values(r) <- as.vector(t(m))
rx <- flip(r, direction="h")

values(r) <- as.vector(m)
ry <- flip(r, direction="v")

v <- rev(r)
```

flowAccumulation *Flow accumulation*

Description

Computes flow accumulation or the total contributing area in terms of numbers of cells upstream of each cell.

Usage

```
## S4 method for signature 'SpatRaster'
flowAccumulation(x, weight=NULL, filename="", ...)
```

Arguments

`x`	SpatRaster with flow direction, see `terrain`.
`weight`	SpatRaster with weight/score daa. For example, cell area or precipitation
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Details

The algorithm is an adaptation of the one proposed by Zhou at al, 2019.

Value

SpatRaster

Author(s)

Emanuele Cordano

References

Zhou, G., Wei, H. & Fu, S. A fast and simple algorithm for calculating flow accumulation matrices from raster digital elevation. Front. Earth Sci. 13, 317–326 (2019). doi:10.1007/s11707-018-0725-9. Also see: https://ica-abs.copernicus.org/articles/1/434/2019/

See Also

`terrain`,`watershed`, `NIDP`

Examples

```
elev1 <- array(NA,c(9,9))
elev2 <- elev1
dx <- 1
dy <- 1
for (r in 1:nrow(elev1)) {
  y <- (r-5)*dx
  for (c in 1:ncol(elev1)) {

    x <- (c-5)*dy
    elev1[r,c] <- 5*(x^2+y^2)
    elev2[r,c] <- 10+5*(abs(x))-0.001*y
  }
}


## Elevation raster
elev1 <- rast(elev1)
elev2 <- rast(elev2)

t(array(elev1[],rev(dim(elev1)[1:2])))
t(array(elev2[],rev(dim(elev2)[1:2])))

plot(elev1)
plot(elev2)

## Flow direction raster
flowdir1<- terrain(elev1,v="flowdir")
flowdir2<- terrain(elev2,v="flowdir")

t(array(flowdir1[],rev(dim(flowdir1)[1:2])))
t(array(flowdir2[],rev(dim(flowdir2)[1:2])))

plot(flowdir1)
plot(flowdir2)

##
flow_acc1 <- flowAccumulation((flowdir1))
flow_acc2 <- flowAccumulation((flowdir2))

weight <- elev1*0+10

flow_acc1w <- flowAccumulation(flowdir1,weight)
flow_acc2w <- flowAccumulation(flowdir2,weight)

t(array(flow_acc1w[],rev(dim(flow_acc1w)[1:2])))
t(array(flow_acc2w[],rev(dim(flow_acc2w)[1:2])))

plot(flow_acc1w)
plot(flow_acc2w)
```

```
## Application wth example elevation data

elev <- rast(system.file('ex/elev.tif',package="terra"))
flowdir <- terrain(elev,"flowdir")

weight <- cellSize(elev,unit="km")
flowacc_weight <- flowAccumulation(flowdir,weight)
flowacc  <- flowAccumulation(flowdir)
```

---

focal *Focal values*

Description

Calculate focal ("moving window") values for each cell.

Usage

```
## S4 method for signature 'SpatRaster'
focal(x, w=3, fun="sum", ..., na.policy="all", fillvalue=NA,
expand=FALSE, silent=TRUE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`w`	window. The window can be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See Details.
`fun`	function that takes multiple numbers, and returns a numeric vector (one or multiple numbers). For example mean, modal, min or max
`...`	additional arguments passed to `fun` such as `na.rm`
`na.policy`	character. Can be used to determine the cells of `x` for which focal values should be computed. Must be one of "all" (compute for all cells), "only" (only for cells that are `NA`) or "omit" (skip cells that are `NA`). Note that the value of this argument does not affect which cells around each focal cell are included in the computations (use `na.rm=TRUE` to ignore cells that are `NA` for that)
`fillvalue`	numeric. The value of the cells in the virtual rows and columns outside of the raster
`expand`	logical. If `TRUE` The value of the cells in the virtual rows and columns outside of the raster are set to be the same as the value on the border. Only available for "build-in" `fun`s such as mean, sum, min and max
`silent`	logical. If `TRUE` error messages are printed that may occur when trying `fun` to determine the length of the returned value. This can be useful in debugging a `fun` that does not work
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	additional arguments for writing files as in `writeRaster`

Details

`focal` The window used must have odd dimensions. If you need even sides, you can use a matrix and add a column or row of `NA`'s to mask out values.

Window values are typically 1 or `NA` to indicate whether a value is used or ignored in computations, respectively. `NA` values in `w` can be useful for creating non-rectangular (e.g. circular) windows.

A weights matrix of numeric values can also be supplied to `w`. In the case of a weights matrix, cells with `NA` weights will be ignored, and the rest of the values in the focal window will be multiplied by the corresponding weight prior to 'fun' being applied. Note, `na.rm` does not need to be `TRUE` if `w` contains `NA` values as these cells are ignored in computations.

The "mean" function is a special case, where supplying weights to `w` will instead calculate a weighted mean.

The "sum" function returns `NA` if all focal cells are `NA` and `na.rm=TRUE`. R would normally return a zero in these cases. See the difference between `focal(x, fun=sum, na.rm=TRUE)` and `focal(x, fun=\(i) sum(i, na.rm=TRUE))`

Example weight matrices

Laplacian filter: `filter=matrix(c(0,1,0,1,-4,1,0,1,0), nrow=3)`

Sobel filters (for edge detection):

`fx=matrix(c(-1,-2,-1,0,0,0,1,2,1), nrow=3)`

`fy=matrix(c(1,0,-1,2,0,-2,1,0,-1), nrow=3)`

Value

SpatRaster

Note

When using global lon/lat rasters, the focal window "wraps around" the date-line.

See Also

focalMat, focalValues, focal3D, focalPairs, focalReg, focalCpp

Examples

```
r <- rast(ncols=10, nrows=10, ext(0, 10, 0, 10))
values(r) <- 1:ncell(r)

f <- focal(r, w=3, fun=function(x, ...) quantile(x, c(.25, .5, .75), ...), na.rm=TRUE)

f <- focal(r, w=3, fun="mean")

# the following two statements are equivalent:
a <- focal(r, w=matrix(1/9, nc=3, nr=3))
b <- focal(r, w=3, fun=mean, na.rm=FALSE)

# but this is different
d <- focal(r, w=3, fun=mean, na.rm=TRUE)
```

```
## illustrating the effect of different
## combinations of na.rm and na.policy
v <- vect(system.file("ex/lux.shp", package="terra"))
r <- rast(system.file("ex/elev.tif", package="terra"))
r[45:50, 45:50] <- NA

# also try "mean" or "min"
f <- "sum"
# na.rm=FALSE
plot(focal(r, 5, f) , fun=lines(v))

# na.rm=TRUE
plot(focal(r, 5, f, na.rm=TRUE), fun=lines(v))

# only change cells that are NA
plot(focal(r, 5, f, na.policy="only", na.rm=TRUE), fun=lines(v))

# do not change cells that are NA
plot(focal(r, 5, f, na.policy="omit", na.rm=TRUE), fun=lines(v))

# does not do anything
# focal(r, 5, f, na.policy="only", na.rm=FALSE)
```

---

focal3D *Three-dimensional focal values*

Description

Calculate focal ("moving window") values for the three-dimensional neighborhood (window) of focal cells. See `focal` for two-dimensional focal computation.

Usage

```
## S4 method for signature 'SpatRaster'
focal3D(x, w=3, fun=mean, ..., na.policy="all", fillvalue=NA, pad=FALSE,
padvalue=fillvalue, expand=FALSE, silent=TRUE,
filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`w`	window. A rectangular prism (cuboid) defined by three numbers or by a three-dimensional array. The values are used as weights, and are usually zero, one, NA, or fractions. The window used must have odd dimensions. If you desire to use even sides, you can use an array, and pad the values with rows and/or columns that contain only `NA`s.
`fun`	function that takes multiple numbers, and returns one or multiple numbers for each focal area. For example mean, modal, min or max

`...`	additional arguments passed to `fun` such as `na.rm`
`na.policy`	character. Can be used to determine the cells of `x`, in the central layer, for which focal values should be computed. Must be one of "all" (compute for all cells), "only" (only for cells that are `NA`) or "omit" (skip cells that are `NA`). Note that the value of this argument does not affect which cells around each focal cell are included in the computations (use `na.rm=TRUE` to ignore cells that are `NA` in the computation of the focal value)
`fillvalue`	numeric. The value of the cells in the virtual rows and columns outside of the raster
`pad`	logical. Add virtual layers before the first and after the last layer
`padvalue`	numeric. The value of the cells in the virtual layers
`expand`	logical. Add virtual layers before the first or after the last layer that are the same as the first or last layers. If `TRUE`, arguments `pad` and `padvalue` are ignored
`silent`	logical. If `TRUE` error messages are printed that may occur when trying `fun` to determine the length of the returned value. This can be useful in debugging a function passed to `fun` that does not work
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`focal`

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
x <- focal3D(r, c(5,5,3), na.rm=TRUE)

a <- array(c(0,1,0,1,1,1,0,1,0, rep(1,9), 0,1,0,1,1,1,0,1,0), c(3,3,3))
a[a==0] <- NA
z <- focal3D(r, a, na.rm=TRUE)
```

---

focalCpp — *Compute focal values with an iterating C++ function*

Description

Calculate focal values with a C++ function that iterates over cells to speed up computations by avoiding an R loop (with apply).

See `focal` for an easier to use method.

Usage

```
## S4 method for signature 'SpatRaster'
focalCpp(x, w=3, fun, ..., fillvalue=NA,
silent=TRUE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`w`	window. The window can be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See the Details section in `focal`
`fun`	`cppFunction` that iterates over cells. For C++ functions that operate on a single focal window, or for R functions use `focal` instead. The function must have at least three arguments. The first argument can have any name, but it must be a `Rcpp::NumericVector`, `Rcpp::IntegerVector` or a `std::vector<double>`. This is the container that receives the focal values. The other two arguments `ni` and `wi` must be of type `size_t`. `ni` represents the number of cells and `nw` represents the size of (number of elements in) the window
`...`	additional arguments to `fun`
`fillvalue`	numeric. The value of the cells in the virtual rows and columns outside of the raster
`silent`	logical. If `TRUE` error messages are printed that may occur when trying `fun` to determine the length of the returned value. This can be useful in debugging a `fun` that does not work
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`focal`, `focalValues`

Examples

```
## Not run:
library(Rcpp)
cppFunction(
'NumericVector sum_and_multiply(NumericVector x, double m, size_t ni, size_t nw) {
NumericVector out(ni);
// loop over cells
size_t start = 0;
for (size_t i=0; i<ni; i++) {
size_t end = start + nw;
// compute something for a window
```

```
double v = 0;
// loop over the values of a window
for (size_t j=start; j<end; j++) {
v += x[j];
}
out[i] = v * m;
start = end;
}
return out;
}'
)

nr <- nc <- 10
r <- rast(ncols=nc, nrows=nr, ext= c(0, nc, 0, nr))
values(r) <- 1:ncell(r)

raw <- focalCpp(r, w=3, fun=sum_and_multiply, fillvalue=0, m=10)

# same as
f1 <- focal(r, w=3, fun=sum, fillvalue=0) *10
all(values(f1) == values(raw))

# and as
ffun <- function(x, m) { sum(x) * m }
f2 <- focal(r, w=3, fun=ffun, fillvalue=0, m=10)


# You can also use an R function with focalCpp but this
# is not recommended

R_sm_iter <- function(x, m, ni, nw) {
out <- NULL
for (i in 1:ni) {
start <- (i-1) * nw + 1
out[i] <- sum(x[start:(start+nw-1)]) * m
}
out
}

fr <- focalCpp(r, w=3, fun=R_sm_iter, fillvalue=0, m=10)


## End(Not run)
```

`focalMat` *Focal weights matrix*

Description

Make a focal ("moving window") weight matrix for use in the `focal` function. The sum of the values adds up to one.

Usage

```
focalMat(x, d, type=c('circle', 'Gauss', 'rectangle'), fillNA=FALSE)
```

Arguments

x	SpatRaster
d	numeric. If type=circle, the radius of the circle (in units of the crs). If type=rectangle the dimension of the rectangle (one or two numbers). If type=Gauss the size of sigma, and optionally another number to determine the size of the matrix returned (default is 3*sigma)
type	character indicating the type of filter to be returned
fillNA	logical. If TRUE, zeros are set to NA such that they are ignored in the computations. Only applies to type="circle"

Value

matrix that can be used with focal

Examples

```
r <- rast(ncols=180, nrows=180, xmin=0)
focalMat(r, 2, "circle")

focalMat(r, c(2,3), "rect")

# Gaussian filter for square cells
gf <- focalMat(r, 1, "Gauss")
```

---

focalPairs *Focal function across two layers*

Description

Calculate values such as a correlation coefficient for focal regions in two neighboring layers. A function is applied to the first and second layer, then to the second and third layer, etc.

Usage

```
## S4 method for signature 'SpatRaster'
focalPairs(x, w=3, fun, ..., fillvalue=NA,
filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatRaster with at least two layers
w	numeric or matrix to define the focal window. The window an be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See the Details section in `focal`. Note that if a matrix with numbers other than zero or one are used, the values are used as weights. For this to work, `fun` must have an argument `weights`
fun	a function with at least two arguments (one for each layer). There is a built-in function "pearson" (for both the weighted and the unweighted Pearson correlation coefficient. This function has an additional argument `na.rm=FALSE`
...	additional arguments for `fun`
fillvalue	numeric. The value of the cells in the virtual rows and columns outside of the raster
filename	character. Output filename
overwrite	logical. If `TRUE`, `filename` is overwritten
wopt	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`layerCor`, `focalReg`, `focal`, `focal3D`

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
set.seed(0)
r[[1]] <- flip(r[[1]], "horizontal")
r[[2]] <- flip(r[[2]], "vertical") + init(rast(r,1), runif)
r[[3]] <- init(rast(r,1), runif)

x <- focalPairs(r, w=5, "pearson", na.rm=TRUE)
plot(x)


# suppress warning "the standard deviation is zero"
suppressWarnings(x <- focalPairs(r, w=5, "pearson", use="complete.obs"))

z <- focalPairs(r, w=9, function(x, y) mean(x) + mean(y))
```

focalReg *Focal regression*

Description

Calculate values for a moving-window by comparing the value in one layers with the values in one to many other layers. A typical case is the computation of the coefficients for a focal linear regression model.

Usage

```
## S4 method for signature 'SpatRaster'
focalReg(x, w=3, fun="ols", ..., fillvalue=NA, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster with at least two layers. The first is the "Y" (dependent) variable and the remainder are the "X" (independent) variables
`w`	numeric or matrix to define the focal window. The window an be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See the Details section in `focal`. Note that if a matrix with numbers other than zero or one are used, the values are used as weights. For this to work, `fun` must have an argument `weights`
`fun`	a function with at least two arguments (one for each layer). There is a built-in function "ols" for both the weighted and unweighted Ordinary Least Square regression. This function has an additional argument `na.rm=FALSE` and `intercept=TRUE`
`...`	additional arguments for `fun`
`fillvalue`	numeric. The value of the cells in the virtual rows and columns outside of the raster
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`focal`, `focal3D`, focalValues

Examples

```
r <- rast(ncols=10, nrows=10, ext(0, 10, 0, 10))
values(r) <- 1:ncell(r)
x <- c(r, init(r, runif) * r)
f <- focalReg(x, 3)
```

`focalValues` *Get focal values*

Description

Get a matrix in which each row had the focal values of a cell. These are the values of a cell and a rectangular window around it.

Usage

```
## S4 method for signature 'SpatRaster'
focalValues(x, w=3, row=1, nrows=nrow(x), fill=NA)
```

Arguments

`x`	SpatRaster or SpatVector
`w`	window. The window can be defined as one (for a square) or two odd numbers (row, col); or with an odd sized matrix
`row`	positive integer. Row number to start from, should be between 1 and nrow(x)
`nrows`	positive integer. How many rows?
`fill`	numeric used as values for imaginary cells outside the raster

Value

matrix

Examples

```
r <- rast(ncol=4, nrow=4, crs="+proj=utm +zone=1 +datum=WGS84")
values(r) <- 1:ncell(r)
focalValues(r)
```

`forceCCW` *force counter-clockwise polygons*

Description

Assure that the nodes of outer rings of polygons are in counter-clockwise order.

Usage

```
## S4 method for signature 'SpatVector'
forceCCW(x)
```

Arguments

x	SpatVector of polygons

Value

SpatVector

Examples

```
p <- vect("POLYGON ((2 45, 2 55, 18 55, 18 45, 2 45))")
pcc <- forceCCW(p)
geom(pcc, wkt=TRUE)
```

---

freq — *Frequency table*

Description

Frequency table of the values of a SpatRaster. `NA`s are not counted unless `value=NA`.

You can provide a SpatVector or additional SpatRaster to define zones for which to do tabulations.

Usage

```
## S4 method for signature 'SpatRaster'
freq(x, digits=0, value=NULL, bylayer=TRUE, usenames=FALSE, zones=NULL, wide=FALSE)
```

Arguments

x	SpatRaster
digits	integer. Used for rounding the values before tabulation. Ignored if `NA`
value	numeric. An optional single value to only count the number of cells with that value. This value can be `NA`
bylayer	logical. If `TRUE` tabulation is done by layer
usenames	logical. If `TRUE` layers are identified by their names instead of their numbers Only relevant if `bylayer` is `TRUE`
zones	SpatRaster or SpatVector to define zones for which the tabulation should be done
wide	logical. Should the results by "wide" instead of "long"?

Value

A `data.frame` with 3 columns (layer, value, count) unless `bylayer=FALSE` in which case a`data.frame` with two columns is returned (value, count).

Examples

```
r <- rast(nrows=10, ncols=10)
set.seed(2)
values(r) <- sample(5, ncell(r), replace=TRUE)

freq(r)

x <- c(r, r/3)
freq(x, bylayer=FALSE)
freq(x)

freq(x, digits=1)
freq(x, digits=-1)

freq(x, value=5)
```

---

gaps — *Find gaps between polygons*

Description

Get the gaps between polygons of a SpatVector

Usage

```
## S4 method for signature 'SpatVector'
gaps(x)
```

Arguments

x	SpatVector

Value

SpatVector

See Also

`sharedPaths`, `topology`, and `fillHoles` to get or remove polygon holes

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
h <- convHull(v[-12], "NAME_1")
g <- gaps(h)
```

gdal — GDAL version, supported file formats, and cache size

Description

Set the GDAL warning level or get a data.frame with the available GDAL drivers (file formats), or, if warn=NA and drivers=FALSE, you get the version numbers of one or all of the GDAL, PROJ and GEOS libraries.

GDAL is the software library that terra builds on to read and write spatial data and for some raster data processing. PROJ is used for transformation of coordinates ("projection") and GEOS is used for geometric operations with vector data.

Usage

```
gdal(warn=NA, drivers=FALSE, lib="gdal")
gdalCache(size=NA)
setGDALconfig(option, value="")
getGDALconfig(option)
```

Arguments

warn	If NA and drivers=FALSE, the version of the library specified by lib is returned. Otherwise, the value should be an integer between 1 and 4 representing the level of GDAL warnings and errors that are passed to R. 1 = warnings and errors; 2 = errors only (recoverable errors as a warning); 3 = irrecoverable errors only; 4 = ignore all errors and warnings. The default setting is 3
drivers	logical. If TRUE a data.frame with the raster and vector data formats that are available.
lib	character. "gdal", "proj", or "geos", or any other value to get the versions numbers of all three
size	numeric. The new cache size in MB
option	character. GDAL configuration option name, or a "name=value" string (in which case the value argument is ignored
value	character. value for GDAL configuration option. Use "" to reset it to its default value

Value

character

See Also

describe for file-level metadata "GDALinfo"

Examples

```
gdal()
gdal(2)
head(gdal(drivers=TRUE))
```

---

geom *Get the geometry (coordinates) of a SpatVector*

Description

Get the geometry of a SpatVector. If `wkt=FALSE`, this is a five-column matrix or data.frame: the vector object ID, the IDs for the parts of each object (e.g. five polygons that together are one spatial object), the x (longitude) and y (latitude) coordinates, and a flag indicating whether the part is a "hole" (only relevant for polygons).

If `wkt=TRUE`, the "well-known text" representation is returned as a character vector. If `hex=TRUE`, the "hexademical" representation is returned as a character vector. If `wkb=TRUE`, the "well-known binary" representation is returned as a list of raw vectors.

Usage

```
## S4 method for signature 'SpatVector'
geom(x, wkt=FALSE, hex=FALSE, wkb=FALSE, df=FALSE, list=FALSE, xnm="x", ynm="y")
```

Arguments

`x`	SpatVector
`wkt`	logical. If `TRUE` the WKT geometry is returned (unless `hex` is also `TRUE`)
`hex`	logical. If `TRUE` the hexadecimal geometry is returned
`wkb`	logical. If `TRUE` the raw WKB geometry is returned (unless either of `hex` or `wkt` is also `TRUE`)
`df`	logical. If `TRUE` a `data.frame` is returned instead of a matrix (only if `wkt=FALSE`, `hex=FALSE`, and `list=FALSE`)
`list`	logical. If `TRUE` a nested `list` is returned with data.frames of coordinates
`xnm`	character. If `list=TRUE` the "x" column name for the coordinates data.frame
`ynm`	character. If `list=TRUE` the "y" column name for the coordinates data.frame

Value

matrix, vector, data.frame, or list

See Also

`crds`, `xyFromCell`

Examples

```
x1 <- rbind(c(-175,-20), c(-140,55), c(10, 0), c(-140,-60))
x2 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))
x3 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
x4 <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))
z <- rbind(cbind(object=1, part=1, x1), cbind(object=2, part=1, x2),
           cbind(object=3, part=1, x3), cbind(object=3, part=2,  x4))
colnames(z)[3:4] <- c('x', 'y')
z <- cbind(z, hole=0)
z[(z[, "object"]==3 & z[,"part"]==2), "hole"] <- 1

p <- vect(z, "polygons")
geom(p)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
g <- geom(v)
head(g)

w <- geom(v, wkt=TRUE)
substr(w, 1, 60)
```

---

geomtype   *Geometry type of a SpatVector*

Description

Get the geometry type (points, lines, or polygons) of a SpatVector. See datatype for the data types of the fields (attributes, variables) of a SpatVector.

Usage

```
## S4 method for signature 'SpatVector'
geomtype(x)

## S4 method for signature 'SpatVector'
is.points(x)

## S4 method for signature 'SpatVector'
is.lines(x)

## S4 method for signature 'SpatVector'
is.polygons(x)
```

Arguments

x	SpatVector

Value

character

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

geomtype(v)
is.polygons(v)
is.lines(v)
is.points(v)

names(v)
datatype(v)
```

`global` *global statistics*

Description

Compute global statistics, that is summarized values of an entire SpatRaster.

If `x` is very large `global` can fail, except when `fun` is one of these built-in functions "mean", "min", "max", "sum", "prod", "range" (min and max), "rms" (root mean square), "sd" (sample standard deviation), "std" (population standard deviation), "isNA" (number of cells that are NA), "notNA" (number of cells that are not NA), "anyNA", "anynotNA". Note that "anyNA" and "anynotNA" cannot be combined with other functions.

The reason that this can fail with large raster and a custom function is that all values need to be loaded into memory. To circumvent this problem you can run `global` with a sample of the cells.

You can compute a weighted mean or sum by providing a SpatRaster with weights.

Usage

```
## S4 method for signature 'SpatRaster'
global(x, fun="mean", weights=NULL, maxcell=Inf, ...)
```

Arguments

`x`	SpatRaster
`fun`	function to be applied to summarize the values by zone. Either as one or more of these built-in character values: "max", "min", "mean", "sum", "range", "rms" (root mean square), "sd", "std" (population sd, using `n` rather than `n-1`), "isNA", "notNA"; or a proper R function (but these may fail for very large SpatRasters unless you specify `maxcell`)
`...`	additional arguments passed on to `fun`
`weights`	NULL or SpatRaster
`maxcell`	positive integer used to take a regular sample of `x`. Ignored by the built-in functions.

Value

A `data.frame` with a row for each layer

See Also

`zonal` for "zonal" statistics, and `app` or `Summary-methods` for "local" statistics, and `extract` for summarizing values for polygons. Also see `focal` for "focal" or "moving window" operations.

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
global(r, "sum")
global(r, "mean", na.rm=TRUE)
x <- c(r, r/10)
global(x, c("sum", "mean", "sd"), na.rm=TRUE)

global(x, function(i) min(i) / max(i))
```

---

graticule — *Create a graticule*

Description

Create a graticule. That is, a grid of lon/lat lines that can be used to on a projected map.

The object returned, a SpatGraticule, can be plotted with `plot` and `lines`. There is also a `crop` method.

Usage

```
graticule(lon=30, lat=30, crs="")
```

Arguments

`lon`	numeric. Either a single number (the interval between longitudes), or a vector with longitudes
`lat`	numeric. Either a single number (the interval between latitudes), or a vector with latitudes
`crs`	character. The coordinate reference system to use

Value

SpatGraticule

See Also

`plot<SpatGraticule>`.

Examples

```
g <- graticule(60, 30, crs="+proj=robin")
g

graticule(90, c(-90, -60, -23.5, 0, 23.5, 60, 90), crs="+proj=robin")
```

---

gridDist *Distance on a grid*

Description

The function calculates the distance to cells of a SpatRaster when the path has to go through the centers of the eight neighboring raster cells.

The default distance (when `scale=1`, is meters if the coordinate reference system (CRS) of the SpatRaster is longitude/latitude (`+proj=longlat`) and in the linear units of the CRS (typically meters) in other cases.

Distances are computed by summing local distances between cells, which are connected with their neighbors in 8 directions.

The shortest distance to the cells with the `target` value is computed for all cells that are not `NA`. Cells that are `NA` cannot be traversed and are ignored, unless the target itself is `NA`, in which case the distance to the nearest cell that is not `NA` is computed for all cells that are `NA`.

Usage

```
## S4 method for signature 'SpatRaster'
gridDist(x, target=0, scale=1, maxiter=50, filename="", ...)
```

Arguments

`x`	SpatRaster
`target`	numeric. value of the target cells (where to compute distance to)
`scale`	numeric. Scale factor. For longitude/latitude data 1 = "m" and 1000 = "km". For planar data that is also the case of the distance unit of the crs is "m"
`maxiter`	numeric. The maximum number of iterations. Increase this number if you get the warning that `costDistance` did not converge. Only relevant when target is not `NA`
`filename`	character. output filename (optional)
`...`	additional arguments as for `writeRaster`

Value

SpatRaster

See Also

See distance for "as the crow flies" distance, and costDist for distance across a landscape with variable friction

Examples

```
# global lon/lat raster
r <- rast(ncol=10,nrow=10, vals=1)
r[48] <- 0
r[66:68] <- NA
d <- gridDist(r)
plot(d)


# planar
crs(r) <- "+proj=utm +zone=15 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
d <- gridDist(r)
plot(d)

# distance to cells that are not NA
rr <- classify(r, cbind(1, NA))
dd <- gridDist(rr, NA)
```

---

halo *Add halo-ed text to a plot*

Description

Add text to a plot that has a "halo". That is, a buffer around it to enhance visibility.

Usage

```
halo(x, y=NULL, labels, col="black", hc="white", hw=0.1, ...)
```

Arguments

x, y	numeric. coordinates where the text labels should be written
labels	character. The text to be written
col	character. The main color to be used
hc	character. The halo color
hw	numeric. The halo width
...	additional arguments to pass to text

See Also

text, plot

Examples

```
r <- rast(nrows=4, ncols=4)
values(r) <- 1:ncell(r)
plot(r, col="blue", legend=FALSE)

text(-100, 20, "hello", cex=2)
halo(50, 20, "hello", cex=2)

halo(0, -20, "world", font=3, hc="light blue", cex=2, hw=.2)
halo(0, 90, "world", font=2, cex=2, hw=.2, xpd=TRUE, pos=2)
halo(0, 90, "world", col="white", font=2, hc="blue", cex=2, hw=.2, xpd=TRUE, pos=4)
```

---

headtail *head and tail of a SpatRaster or SpatVector*

Description

Show the head (first values) or tail (last values) of a SpatRaster or of the attributes of a SpatVector.

Usage

```
head(x, ...)
tail(x, ...)
```

Arguments

x	SpatRaster or SpatVector
...	additional arguments passed on to other methods

Value

matrix (SpatRaster) or data.frame (SpatVector)

See Also

show, geom

Examples

```
r <- rast(nrows=25, ncols=25)
values(r) <- 1:ncell(r)
head(r)
tail(r)
```

hist — *Histogram*

Description

Create a histogram of the values of a SpatRaster. For large datasets a sample of `maxcell` is used.

Usage

```
## S4 method for signature 'SpatRaster'
hist(x, layer, maxcell=1000000, plot=TRUE, maxnl=16, main, ...)
```

Arguments

`x`	SpatRaster
`layer`	positive integer or character to indicate layer numbers (or names). If missing, all layers up to `maxnl` are used
`maxcell`	integer. To regularly sample very large objects
`plot`	logical. Plot the histogram or only return the histogram values
`maxnl`	positive integer. The maximum number of layers to use. Ignored if `layer` is not missing
`main`	character. Main title(s) for the plot. Default is the value of `names`
`...`	additional arguments. See `hist`

Value

This function is principally used for plotting a histogram, but it also returns an object of class "histogram" (invisibly if `plot=TRUE`).

See Also

`pairs`, `boxplot`

Examples

```
r1 <- r2 <- rast(nrows=50, ncols=50)
values(r1) <- runif(ncell(r1))
values(r2) <- runif(ncell(r1))
rs <- r1 + r2
rp <- r1 * r2

opar <- par(no.readonly =TRUE)
par(mfrow=c(2,2))
plot(rs, main='sum')
plot(rp, main='product')
hist(rs)
a <- hist(rp)
```

```
a
x <- c(rs, rp, sqrt(rs))
hist(x)
par(opar)
```

---

identical *Compare two SpatRasters for equality*

Description

Compare two SpatRasters for equality.

First the attributes of the objects are compared. If these are the same, a the raster cells are compared as well. This can be time consuming, and you may prefer to use a sample instead with `all.equal`

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
identical(x, y)
```

Arguments

x	SpatRaster
y	SpatRaster

Value

single logical value

See Also

`all.equal`, `compareGeom`

Examples

```
x <- sqrt(1:100)
mat <- matrix(x, 10, 10)
r1 <- rast(nrows=10, ncols=10, xmin=0, vals = x)
r2 <- rast(nrows=10, ncols=10, xmin=0, vals = t(mat))

identical(r1, r2)
identical(r1, r1*1)
identical(rast(r1), rast(r2))
```

ifel *ifelse for SpatRasters*

Description

Implementation of `ifelse` for SpatRasters. This method allows for a concise expression of what can otherwise be achieved with a combination of `classify`, `mask`, and `cover`.

`ifel` is an `R` equivalent to the `Con` method in ArcGIS (arcpy).

Usage

```
## S4 method for signature 'SpatRaster'
ifel(test, yes, no, filename="", ...)
```

Arguments

`test`	SpatRaster with logical (TRUE/FALSE) values
`yes`	SpatRaster or numeric
`no`	SpatRaster or numeric
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

Examples

```
r <- rast(nrows=5, ncols=5, xmin=0, xmax=1, ymin=0, ymax=1)
values(r) <- c(-10:0, NA, NA, NA, 0:10)

x <- ifel(r > 1, 1, r)
# same as
a <- classify(r, cbind(1, Inf, 1))
# or
b <- app(r, fun=function(i) {i[i > 1] <- 1; i})
# or
d <- clamp(r, -Inf, 1)
# or (not recommended for large datasets)
e <- r
e[e>1] <- 1

## other examples
f <- ifel(is.na(r), 100, r)

z <- ifel(r > -2 & r < 2, 100, 0)
```

```
# nested expressions
y <- ifel(r > 1, 1, ifel(r < -1, -1, r))

k <- ifel(r > 0, r+10, ifel(r < 0, r-10, 3))
```

---

image *SpatRaster image method*

Description

Plot (make a map of) the values of a SpatRaster via `image`. See `plot` if you need more fancy options such as a legend.

Usage

```
## S4 method for signature 'SpatRaster'
image(x, y=1, maxcell=500000, ...)
```

Arguments

`x`	SpatRaster
`y`	positive integer indicating the layer to be plotted, or a character indicating the name of the layer
`maxcell`	positive integer. Maximum number of cells to use for the plot
`...`	additional arguments as for `graphics::image`

See Also

`plot`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
image(r)
image(r, col=rainbow(24))
```

impose	*Impose the geometry of a SpatRaster to those in a SpatRasterCollection.*

Description

Warp the members of a SpatRasterCollection to match the geometry of a SpatRaster.

Usage

```
## S4 method for signature 'SpatRasterCollection'
impose(x, y, filename="", ...)
```

Arguments

`x`	SpatRasterCollection
`y`	SpatRaster
`filename`	character. Output filename
`...`	list with named options for writing files as in `writeRaster`

Value

SpatRaster

See Also

`resample`

initialize	*Initialize a SpatRaster with values*

Description

Create a SpatRaster with values reflecting a cell property: "x", "y", "col", "row", "cell" or "chess". Alternatively, a function can be used. In that case, cell values are initialized without reference to pre-existing values. E.g., initialize with a random number (`fun=runif`). While there are more direct ways of achieving this for small objects (see examples) for which a vector with all values can be created in memory, the `init` function will also work for SpatRasters with many cells.

Usage

```
## S4 method for signature 'SpatRaster'
init(x, fun, filename="", ...)
```

Arguments

x	SpatRaster
fun	function to be applied. This must be a either single number, multiple numbers, a function, or one of a set of known character values. A function must take the number of cells as a single argument to return a vector of values with a length equal to the number of cells, such as `fun=runif`. Allowed character values are "x", "y", "row", "col", "cell", and "chess" to get the x or y coordinate, row, col or cell number or a chessboard pattern (alternating 0 and 1 values)
filename	character. Output filename
...	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

Examples

```
r <- rast(ncols=10, nrows=5, xmin=0, xmax=10, ymin=0, ymax=5)
x <- init(r, fun="cell")
y <- init(r, fun=runif)

# initialize with a single value
z <- init(r, fun=8)
```

---

inplace *Change values in-place*

Description

These "in-place" replacement methods assign new value to an object without making a copy. That is efficient, but if there is a copy of the object that you made by standard assignment (e.g. with y <- x), that copy is also changed.

`set.names` is the in-place replacement version of `names<-`.

`set.ext` is the in-place replacement version of `ext<-`

`set.values` is the in-place replacement version of `[<-`.

`set.cats` is the in-place replacement version of `categories`

`set.crs` is the in-place replacement version of `crs<-`

Usage

```
## S4 method for signature 'SpatRaster'
set.names(x, value, index=1:nlyr(x), validate=FALSE)
## S4 method for signature 'SpatRasterDataset'
set.names(x, value, index=1:length(x), validate=FALSE)
## S4 method for signature 'SpatVector'
```

```
set.names(x, value, index=1:ncol(x), validate=FALSE)

## S4 method for signature 'SpatRaster'
set.ext(x, value)
## S4 method for signature 'SpatVector'
set.ext(x, value)

## S4 method for signature 'SpatRaster'
set.crs(x, value)
## S4 method for signature 'SpatVector'
set.crs(x, value)

## S4 method for signature 'SpatRaster'
set.values(x, cells, values, layer=0)
## S4 method for signature 'SpatRasterDataset'
set.values(x)

## S4 method for signature 'SpatRaster'
set.cats(x, layer=1, value, active=1)

## S4 method for signature 'SpatRaster'
set.RGB(x, value, type="rgb")
```

Arguments

`x`	SpatRaster
`value`	character for `set.names`. For `set.cats`: a data.frame with columns (value, category) or vector with category names. For `set.RGB` 3 or 4 numbers indicating the RGB(A) layers
`index`	positive integer indicating layer(s) to assign a name to
`validate`	logical. Make names valid and/or unique?
`cells`	cell numbers or missing
`values`	replacement values or missing to load all values into memory
`layer`	positive integer(s) indicating to which layer(s) to you want to assign these categories or to which you want to set these values. A number < 1 indicates "all layers"
`active`	positive integer indicating the active category (column number in `value`, but not counting the first column
`type`	character. The color space. One of "rgb" "hsv", "hsi" and "hsl"

Value

logical (invisibly)

Examples

```
s <- rast(ncols=5, nrows=5, nlyrs=3)
x <- s
names(s)
names(s) <- c("a", "b", "c")
names(s)
names(x)

x <- s
set.names(s, c("e", "f", "g"))
names(s)
names(x)

set.ext(x, c(0,180,0,90))

f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)

#values from file to memory
set.values(r)

# change values
set.values(r, 1:1000, 900)
```

`inset` *Make an inset map*

Description

Make an inset map or scale the extent of a SpatVector

Usage

```
## S4 method for signature 'SpatVector'
inset(x, e, loc="", scale=0.2, background="white",
perimeter=TRUE, box=NULL, pper, pbox, offset=0.1, add=TRUE, ...)

## S4 method for signature 'SpatRaster'
inset(x, e, loc="", scale=0.2, background="white",
perimeter=TRUE, box=NULL, pper, pbox, offset=0.1, add=TRUE, ...)

## S4 method for signature 'SpatVector'
inext(x, e, y=NULL, gap=0)
```

Arguments

x	SpatVector, SpatRaster
e	SpatExtent to set the size and location of the inset. Or missing

loc	character. One of "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center"
scale	numeric. The relative size of the inset, used when x is missing
background	color for the background of the inset. Use NA for no background color
perimeter	logical. If TRUE a perimeter (border) is drawn around the inset
box	SpatExtent or missing, to draw a box on the inset, e.g. to show where the map is located in a larger area
pper	list with graphical parameters (arguments) such as col and lwd for the perimeter line
pbox	list with graphical parameters (arguments) such as col and lwd for the box (line)
offset	numeric. Value between 0.1 and 1 to indicate the relative distance between what is mapped and the bounding box
add	logical. Add the inset to the map?
...	additional arguments passed to plot for the drawing of x
y	SpatVector. If not NULL, y is scaled based with the parameters for x. This is useful, for example, when x represent boundaries, and y points within these boundaries
gap	numeric to add space between the SpatVector and the SpatExtent

Value

scaled and shifted SpatVector or SpatRaster (returned invisibly)

See Also

sbar, rescale, shift

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
x <- v[v$NAME_2 == "Diekirch", ]

plot(x, density=10, col="blue")
inset(v)

# more elaborate
plot(x, density=10, col="blue")
inset(v, col = "brown", border="lightgrey", perimeter=TRUE,
pper=list(col="orange", lwd=3, lty=2),
box=ext(x), pbox=list(col="blue", lwd=2))

cols <- rep("light grey", 12)
cols[2] <- "red"
e <- ext(c(6.2, 6.3, 49.9, 50))
b <- ext(x)+0.02
inset(v, e=e, col=cols, box=b)
```

```
# with a SpatRaster
ff <- system.file("ex/elev.tif", package="terra")
r <- rast(ff)
r <- crop(r, ext(x) + .01)
plot(r, type="int", mar=c(2,2,2,2), plg=list(x="topright"))
lines(v, lwd=1.5)
lines(x, lwd=2.5)
inset(v, col=cols, loc="topleft", scale=0.15)

# a more complex one
plot(r, plg=list(title="meter\n", shrink=.2, cex=.8))
lines(v, lwd=4, col="white")
lines(v, lwd=1.5)
lines(x, lwd=2.5)
text(x, "NAME_2", cex=1.5, halo=TRUE)
sbar(6, c(6.04, 49.785), type="bar", below="km", label=c(0,3,6), cex=.8)
s <- inset(v, col=cols, box=b, scale=.2, loc="topright", background="light yellow",
pbox=list(lwd=2, lty=5, col="blue"))

# note the returned inset SpatVector
s
lines(s, col="orange")
```

`interpIDW` *Interpolate points using a moving window*

Description

Interpolate points within a moving window using inverse distance weighting. The maximum number of points used can be restricted, optionally by selecting the nearest points.

Usage

```
## S4 method for signature 'SpatRaster,SpatVector'
interpIDW(x, y, field, radius, power=2, smooth=0,
       maxPoints=Inf, minPoints=1, near=TRUE, fill=NA, filename="", ...)

## S4 method for signature 'SpatRaster,matrix'
interpIDW(x, y, radius, power=2, smooth=0,
       maxPoints=Inf, minPoints=1, near=TRUE, fill=NA, filename="", ...)
```

Arguments

`x`	SpatRaster
`y`	SpatVector or matrix with three columns (x,y,z)
`field`	character. field name in SpatVector `y`

radius	numeric. The radius of the circle (single number). If near=FALSE, it is also possible to use two or three numbers. Two numbers are interpreted as the radii of an ellipse (x and y-axis). A third number should indicated the desired, counter clockwise, rotation of the ellipse (in degrees)
power	numeric. Weighting power
smooth	numeric. Smoothing parameter
minPoints	numeric. The minimum number of points to use. If fewer points are found in a search ellipse it is considered empty and the fill value is returned
maxPoints	numeric. The maximum number of points to consider in a search area. Additional points are ignored. If fewer points are found, the fill value is returned
near	logical. Should the nearest points within the neighborhood be used if maxPoints is reached?
fill	numeric. value to use to fill empty cells
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

rasterizeWin, rasterize, interpNear, interpolate

Examples

```
r <- rast(ncol=100, nrow=100, crs="local", xmin=0, xmax=50, ymin=0, ymax=50)
set.seed(100)
x <- runif(25, 5, 45)
y <- runif(25, 5, 45)
z <- sample(25)
xyz <- cbind(x,y,z)

x <- interpIDW(r, xyz, radius=5, power=1, smooth=1, maxPoints=5)
```

---

interpNear *Nearest neighbor interpolation*

Description

Nearest neighbor interpolation of points, using a moving window

Usage

```
## S4 method for signature 'SpatRaster,SpatVector'
interpNear(x, y, field, radius, interpolate=FALSE, fill=NA, filename="", ...)

## S4 method for signature 'SpatRaster,matrix'
interpNear(x, y, radius, interpolate=FALSE, fill=NA, filename="", ...)
```

Arguments

`x`	SpatRaster
`y`	SpatVector or matrix with three columns (x,y,z)
`field`	character. field name in SpatVector `y`
`radius`	numeric. The radius of the circle (single number). If `interpolate=FALSE` it is also possible to use two or three numbers. Two numbers are interpreted as the radii of an ellipse (x and y-axis). A third number should indicated the desired, counter clockwise, rotation of the ellipse (in degrees)
`interpolate`	logical. Should the nearest neighbor values be linearly interpolated between points?
`fill`	numeric. value to use to fill empty cells
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`rasterizeWin`, `rasterize`, `interpIDW`, `interpolate`

Examples

```
r <- rast(ncol=100, nrow=100, crs="local", xmin=0, xmax=50, ymin=0, ymax=50)
set.seed(100)
x <- runif(25, 5, 45)
y <- runif(25, 5, 45)
z <- sample(25)
xyz <- cbind(x,y,z)

x <- interpNear(r, xyz, radius=5)

p <- vect(data.frame(xyz), geom=c("x", "y"))
v <- voronoi(p)

plot(x, col=rainbow(25))
lines(v)

# plot(v, col=rainbow(25)); points(p)
```

interpolation *Spatial interpolation*

Description

Make a SpatRaster with interpolated values using a fitted model object of classes such as "gstat" (gstat package) or "Krige" (fields package), or any other model that has location (e.g., "x" and "y", or "longitude" and "latitude") as predictors (independent variables). If x and y are the only predictors, it is most efficient if you provide an empty (no associated data in memory or on file) SpatRaster for which you want predictions. If there are more spatial predictor variables, provide these as a SpatRaster in the first argument of the function. If you do not have x and y locations as implicit predictors in your model you should use `predict` instead.

Usage

```
## S4 method for signature 'SpatRaster'
interpolate(object, model, fun=predict, ..., xyNames=c("x", "y"),
       factors=NULL, const=NULL, index = NULL, cores=1, cpkgs=NULL,
   na.rm=FALSE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`object`	SpatRaster
`model`	model object
`fun`	function. Default value is "predict", but can be replaced with e.g. "predict.se" (depending on the class of `model`), or a custom function (see examples)
`...`	additional arguments passed to `fun`
`xyNames`	character. variable names that the model uses for the spatial coordinates. E.g., `c("longitude", "latitude")`
`factors`	list with levels for factor variables. The list elements should be named with names that correspond to names in `object` such that they can be matched. This argument may be omitted for some models from which the levels can be extracted from the `model` object
`const`	data.frame. Can be used to add a constant for which there is no SpatRaster for model predictions. This is particularly useful if the constant is a character-like factor value
`index`	positive integer or NULL. Allows for selecting of the variable returned if the model returns multiple variables
`cores`	positive integer. If `cores` > 1, a 'parallel' package cluster with that many cores is created and used
`cpkgs`	character. The package(s) that need to be loaded on the nodes to be able to run the model.predict function (see examples in `predict`)

na.rm	logical. If TRUE, cells with NA values in the predictors are removed from the computation. This option prevents errors with models that cannot handle NA values. In most other cases this will not affect the output. An exception is when predicting with a model that returns predicted values even if some (or all!) variables are NA
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

predict, interpIDW, interpNear

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
ra <- aggregate(r, 10)
xy <- data.frame(xyFromCell(ra, 1:ncell(ra)))
v <- values(ra)
i <- !is.na(v)
xy <- xy[i,]
v <- v[i]

## Not run:
library(fields)
tps <- Tps(xy, v)
p <- rast(r)

# use model to predict values at all locations
p <- interpolate(p, tps)
p <- mask(p, r)
plot(p)

### change "fun" from predict to fields::predictSE to get the TPS standard error
## need to use "rast(p)" to remove the values
se <- interpolate(rast(p), tps, fun=predictSE)
se <- mask(se, r)
plot(se)

### another predictor variable, "e"
e <- (init(r, "x") * init(r, "y")) / 100000000
names(e) <- "e"

z <- as.matrix(extract(e, xy)[,-1])

## add as another independent variable
xyz <- cbind(xy, z)
```

```
tps2 <- Tps(xyz, v)
p2 <- interpolate(e, tps2, xyOnly=FALSE)

## as a linear covariate
tps3 <- Tps(xy, v, Z=z)

## Z is a separate argument in Krig.predict, so we need a new function
## Internally (in interpolate) a matrix is formed of x, y, and elev (Z)

pfun <- function(model, x, ...) {
   predict(model, x[,1:2], Z=x[,3], ...)
}
p3 <- interpolate(e, tps3, fun=pfun)


#### gstat examples
library(gstat)
library(sp)
data(meuse)

### inverse distance weighted (IDW)
r <- rast(system.file("ex/meuse.tif", package="terra"))
mg <- gstat(id = "zinc", formula = zinc~1, locations = ~x+y, data=meuse,
            nmax=7, set=list(idp = .5))
z <- interpolate(r, mg, debug.level=0, index=1)
z <- mask(z, r)

## with a model built with an `sf` object you need to provide custom function

library(sf)
sfmeuse <- st_as_sf(meuse, coords = c("x", "y"), crs=crs(r))
mgsf <- gstat(id = "zinc", formula = zinc~1, data=sfmeuse,  nmax=7, set=list(idp = .5))

interpolate_gstat <- function(model, x, crs, ...) {
v <- st_as_sf(x, coords=c("x", "y"), crs=crs)
p <- predict(model, v, ...)
as.data.frame(p)[,1:2]
}

zsf <- interpolate(r, mgsf, debug.level=0, fun=interpolate_gstat, crs=crs(r), index=1)
zsf <- mask(zsf, r)

### kriging

### ordinary kriging
v <- variogram(log(zinc)~1, ~x+y, data=meuse)
mv <- fit.variogram(v, vgm(1, "Sph", 300, 1))
gOK <- gstat(NULL, "log.zinc", log(zinc)~1, meuse, locations=~x+y, model=mv)
OK <- interpolate(r, gOK, debug.level=0)

## universal kriging
vu <- variogram(log(zinc)~elev, ~x+y, data=meuse)
mu <- fit.variogram(vu, vgm(1, "Sph", 300, 1))
```

```
gUK <- gstat(NULL, "log.zinc", log(zinc)~elev, meuse, locations=~x+y, model=mu)
names(r) <- "elev"
UK <- interpolate(r, gUK, debug.level=0)

## co-kriging
gCoK <- gstat(NULL, 'log.zinc', log(zinc)~1, meuse, locations=~x+y)
gCoK <- gstat(gCoK, 'elev', elev~1, meuse, locations=~x+y)
gCoK <- gstat(gCoK, 'cadmium', cadmium~1, meuse, locations=~x+y)
gCoK <- gstat(gCoK, 'copper', copper~1, meuse, locations=~x+y)
coV <- variogram(gCoK)
plot(coV, type='b', main='Co-variogram')
coV.fit <- fit.lmc(coV, gCoK, vgm(model='Sph', range=1000))
coV.fit
plot(coV, coV.fit, main='Fitted Co-variogram')
coK <- interpolate(r, coV.fit, debug.level=0)
plot(coK)

## End(Not run)
```

`intersect` *Intersection*

Description

You can intersect SpatVectors with each other or with a SpatExtent. Intersecting points with points uses the extent of `y` to get the intersection. Intersecting of points and lines is not supported because of numerical inaccuracies with that. You can use `buffer`, to create polygons from lines and use these with intersect.

You can also intersect two SpatExtents.

When intersecting two SpatRasters these need to be aligned (have the same origin and spatial resolution). The values of the returned SpatRaster are `TRUE` where both input rasters have values, `FALSE` where one has values, and `NA` in all other cells.

When intersecting a SpatExtent and a SpatRaster, the SpatExtent is first aligned to the raster cell boundaries.

See `crop` for the intersection of a SpatRaster with a SpatExtent (or the extent of a SpatRaster or SpatVector) if you want a SpatRaster (not a SpatExtent) as output.

See `is.related(x, y, "intersects")` to find out which geometries of a SpatVector intersect. You can spatially subset a SpatVector with another one with `x[y]`.

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
intersect(x, y)

## S4 method for signature 'SpatVector,SpatExtent'
intersect(x, y)
```

```
## S4 method for signature 'SpatExtent,SpatVector'
intersect(x, y)

## S4 method for signature 'SpatExtent,SpatExtent'
intersect(x, y)

## S4 method for signature 'SpatRaster,SpatRaster'
intersect(x, y)

## S4 method for signature 'SpatRaster,SpatExtent'
intersect(x, y)

## S4 method for signature 'SpatExtent,SpatRaster'
intersect(x, y)
```

Arguments

x	SpatVector, SpatExtent, or SpatRaster
y	SpatVector, SpatExtent, or SpatRaster

Value

Same as x

See Also

union, crop, relate, [

Examples

```
e1 <- ext(-10, 10, -20, 20)
e2 <- ext(0, 20, -40, 5)
intersect(e1, e2)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
e <- ext(5.6, 6, 49.55, 49.7)
x <- intersect(v, e)

p <- vect(c("POLYGON ((5.8 49.8, 6 49.9, 6.15 49.8, 6 49.6, 5.8 49.8))",
"POLYGON ((6.3 49.9, 6.2 49.7, 6.3 49.6, 6.5 49.8, 6.3 49.9))"), crs=crs(v))
values(p) <- data.frame(pid=1:2, area=expanse(p))

y <- intersect(v, p)

r <- s <- rast(ncol=5, nrow=5, xmin=1, xmax=5, ymin=1, ymax=5)
r[5:20] <- 5:20
s[11:20] <- 11:20
rs <- intersect(r, s)

u <- shift(r, .8)
```

```
us <- intersect(u, s)
```

---

is.bool *Raster value types*

Description

The values in a SpatRaster layer are by default numeric, but they can also be set to be logical (Boolean), integer, or categorical (factor).

For a SpatRaster, `as.logical` and `isTRUE` is equivalent to `as.bool`. `isFALSE` is equivalent to `!as.bool`, and `as.integer` is the same as `as.int`.

`as.bool` and `as.int` force the values into the correct range (e.g. whole integers) but in-memory cell values are still stored as numeric. They will behave like the assigned types, though, and will be written to files with that data type (if the file type supports it).

See `levels` and `cats` to create categorical layers by setting labels.

Usage

```
## S4 method for signature 'SpatRaster'
is.bool(x)

## S4 method for signature 'SpatRaster'
as.bool(x, filename, ...)

## S4 method for signature 'SpatRaster'
is.int(x)

## S4 method for signature 'SpatRaster'
as.int(x, filename, ...)

## S4 method for signature 'SpatRaster'
is.factor(x)

## S4 method for signature 'SpatRaster'
as.factor(x)
```

Arguments

`x`	SpatRaster
`filename`	character. Output filename
`...`	list with named options for writing files as in `writeRaster`

Value

The `as.*` methods return a new `SpatRaster`, whereas the `is.*` methods return a `logical` value for each layer in `x`.

See Also

levels and cats to create categorical layers (and set labels).

Examples

```
r <- rast(nrows=10, ncols=10, vals=1:100)
is.bool(r)
z <- as.bool(r)
is.bool(z)

x <- r > 25
is.bool(x)

rr <- r/2
is.int(rr)
is.int(round(rr))
```

---

is.empty — *Check if a SpatExtent or SpatVector is empty*

Description

An empty SpatExtent has no area

An empty SpatVector has no geometries.

Usage

```
## S4 method for signature 'SpatExtent'
is.empty(x)

## S4 method for signature 'SpatVector'
is.empty(x)
```

Arguments

x	SpatVector or SpatExtent

Value

logical

Examples

```
e <- ext(0,0,0,0)
is.valid(e)
is.empty(e)

v <- vect()
is.valid(v)
is.empty(v)
```

is.lonlat *Check for longitude/latitude crs*

Description

Test whether a SpatRaster or SpatVector has a longitude/latitude coordinate reference system (CRS), or perhaps has one. That is, when the CRS is unknown ("") but the x coordinates are within -181 and 181 and the y coordinates are within -90.1 and 90.1. For a SpatRaster you can also test if it has a longitude/latitude CRS and it is "global" (covers all longitudes).

A warning is given if the CRS is missing or if it is specified as longitude/latitude but the coordinates do not match that.

Usage

```
## S4 method for signature 'SpatRaster'
is.lonlat(x, perhaps=FALSE, warn=TRUE, global=FALSE)

## S4 method for signature 'SpatVector'
is.lonlat(x, perhaps=FALSE, warn=TRUE)

## S4 method for signature 'character'
is.lonlat(x, perhaps=FALSE, warn=TRUE)
```

Arguments

x	SpatRaster or SpatVector
perhaps	logical. If TRUE and the CRS is unknown, the method returns TRUE if the coordinates are plausible for longitude/latitude
warn	logical. If TRUE, a warning is given if the CRS is unknown but assumed to be lon/lat and perhaps=TRUE
global	logical. If TRUE, the method tests if the raster covers all longitudes (from -180 to 180 degrees) such that the extreme columns are in fact adjacent

Value

logical or NA

Examples

```
r <- rast()
is.lonlat(r)
is.lonlat(r, global=TRUE)

crs(r) <- ""
is.lonlat(r)
is.lonlat(r, perhaps=TRUE, warn=FALSE)
```

```
crs(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"
is.lonlat(r)
```

`is.rotated` *Check for rotation*

Description

Check if a SpatRaster is "rotated" and needs to be rectified before it can be used

See `rectify`

Usage

```
## S4 method for signature 'SpatRaster'
is.rotated(x)
```

Arguments

`x`	SpatRaster

Value

logical. One value for each raster data *source*

See Also

`rectify`

Examples

```
r <- rast(nrows=10, ncols=10, vals=1:100)
is.rotated(r)
```

`is.valid` *Check or fix polygon or extent validity*

Description

Check the validity of polygons or attempt to fix it. Or check the validity of a SpatExtent.

Usage

```
## S4 method for signature 'SpatVector'
is.valid(x, messages=FALSE, as.points=FALSE)

## S4 method for signature 'SpatVector'
makeValid(x)

## S4 method for signature 'SpatExtent'
is.valid(x)
```

Arguments

x	SpatVector or SpatExtent
messages	logical. If TRUE the error messages are returned
as.points	logical. If TRUE, it is attempted to return locations where polygons are invalid as a SpatVector or points

Value

logical

See Also

topology

Examples

```
w <- vect("POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")
is.valid(w)

w <- vect("POLYGON ((0 -5, 10 0, 10 -10, 4 -2, 0 -5))")
is.valid(w)
is.valid(w, TRUE)

plot(w)
points(cbind(4.54, -2.72), cex=2, col="red")


e <- ext(0, 1, 0, 1)
is.valid(e)

ee <- ext(0, 0, 0, 0)
is.valid(ee)
```

---

k_means *k_means*

Description

Compute k-means clusters for a SpatRaster. For large SpatRasters (with `ncell(x) > maxcell`) this is done in two steps. First a sample of the cells is used to compute the cluster centers. Then each cell is assigned to a cluster by computing the distance to these centers.

Usage

```
## S4 method for signature 'SpatRaster'
k_means(x, centers=3, ..., maxcell=1000000, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`centers`	either the number of clusters, or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) cells in `x` is chosen as the initial centres
`...`	additional arguments passed to `kmeans`
`maxcell`	positive integer. The size of the regular sample used if it is smaller than `ncell(x)`
`filename`	character. Output filename (ignored if `as.raster=FALSE`)
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`kmeans`

Examples

```
f <- system.file("ex/logo.tif", package = "terra")
r <- rast(f)
km <- k_means(r, centers=5)
km
```

lapp *Apply a function to layers of a SpatRaster, or sub-datasets of a SpatRasterDataset*

Description

Apply a function to a SpatRaster, using layers as arguments.

The number of arguments in function `fun` must match the number of layers in the SpatRaster (or the number of sub-datasets in the SpatRasterDataset). For example, if you want to multiply two layers, you could use this function: `fun=function(x,y){return(x*y)}` percentage: `fun=function(x,y){return(100 * x / y)}`. If you combine three layers you could use `fun=function(x,y,z){return((x + y) * z)}`

Before you use the function, test it to make sure that it is vectorized. That is, it should work for vectors longer than one, not only for single numbers. Or if the input SpatRaster(s) have multiple layers, it should work for a matrix (multiple cells) of input data (or matrices in the case of a SpatRasterDataSet). The function must return the same number of elements as its input vectors, or multiples of that. Also make sure that the function is `NA`-proof: it should returns the same number of values when some or all input values are `NA`. And the function must return a vector or a matrix, not a `data.frame`. To test it, run it with `do.call(fun, data)` (see examples).

Use `app` for summarize functions such as `sum`, that take any number of arguments; and `tapp` to do so for groups of layers.

Usage

```
## S4 method for signature 'SpatRaster'
lapp(x, fun, ..., usenames=FALSE, cores=1, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterDataset'
lapp(x, fun, ..., usenames=FALSE, recycle=FALSE,
    filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster or SpatRasterDataset
`fun`	a function that takes a vector and can be applied to each cell of `x`
`...`	additional arguments to be passed to `fun`
`usenames`	logical. Use the layer names (or dataset names if `x` is a SpatRasterDataset) to match the function arguments? If `FALSE`, argument matching is by position
`cores`	positive integer. If `cores > 1`, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object
`recycle`	logical. Recycle layers to match the subdataset with the largest number of layers
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Value

SpatRaster

Note

Use sapp or lapply to apply a function that takes a SpatRaster as argument to each layer of a SpatRaster (that is rarely necessary).

See Also

app, tapp, math

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra")) + 1
ss <- s[[2:1]]

fvi <- function(x, y){ (x - y ) / (x + y) }
# test the function
data <- list(c(1:5,NA), 6:1)
do.call(fvi, data)

x <- lapp(ss, fun=fvi )

# which is the same as supplying the layers to "fun"
# in some cases this will be much faster
y <- fvi(s[[2]], s[[1]])

f2 <- function(x, y, z){ (z - y + 1) / (x + y + 1) }
p1 <- lapp(s, fun=f2 )

p2 <- lapp(s[[1:2]], f2, z=200)

# the usenames argument

fvi2 <- function(red, green){ (red - green ) / (red + green) }
names(s)
x1 <- lapp(s[[1:2]], fvi2, usenames=TRUE)
x2 <- lapp(s[[2:1]], fvi2, usenames=TRUE)
# x1 and x2 are the same, despite the change in the order of the layers
# x4 is also the same, but x3 is not
x3 <- lapp(s[[2:1]], fvi2, usenames=FALSE)

# these fail because there are too many layers in s
# x4 <- lapp(s, fvi2, usenames=TRUE)
# x5 <- lapp(s, fvi2, usenames=FALSE)

pairs(c(x1, x2, x3))

## SpatRasterDataset
x <- sds(s, s[[1]]+50)
fun <- function(x, y) { x/y }
```

```
# test "fun"
data <- list(matrix(1:9, ncol=3), matrix(9:1, ncol=3))
do.call(fun, data)

lapp(x, fun, recycle=TRUE)

# the same, more concisely
z <- s / (s[[1]]+50)
```

`layerCor` *Correlation and (weighted) covariance*

Description

Compute correlation, (weighted) covariance, or similar summary statistics that compare the values of all pairs of the layers of a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
layerCor(x, fun, w, asSample=TRUE, use="everything", maxcell=Inf, ...)
```

Arguments

`x`	SpatRaster
`fun`	character. The statistic to compute: either "cov" (covariance), "weighted.cov" (weighted covariance), or "cor" (pearson correlation coefficient) or your own function that takes two vectors as argument to compute a single number
`w`	SpatRaster with the weights to compute the weighted covariance. It should have a single layer and the same geometry as `x`
`asSample`	logical. If `TRUE`, the statistic for a sample (denominator is `n-1`) is computed, rather than for the population (denominator is `n`). Only for the standard functions
`use`	character. To decide how to handle missing values. This must be (an abbreviation of) one of "everything", "complete.obs", "pairwise.complete.obs", "masked.complete". With "pairwise.complete.obs", the value for a pair of layers is computed for all cells that are not `NA` in that pair. Therefore, it may be that the (number of) cells used varies between pairs. The benefit of this approach is that all available data is used. Use "complete.obs", if you want to only use the values from cells that are not `NA` in any of the layers. By using "masked.complete" you indicate that all layers have NA values in the same cells
`maxcell`	positive integer. The maximum number of cells to be used. If this is smaller than ncell(x), a regular sample of `x` is used
`...`	additional arguments for `fun` (if it is a proper function)

Value

If fun is one of the three standard statistics, you get a list with three items: the correlation or (weighted) covariance matrix, the (weighted) means, and the number of data cells in each comparison. The means are also a matrix because they may depend on the combination of layers if different cells have missing values and these are excluded from the computation. The rows of the mean matrix represent the layer whose (weighted) mean is being calculated and the columns represent the layer it is being paired with. Only cells with non-missing observations for both layers are used in the calculation of the (weighted) mean. The diagonals of the mean and n matrices are set to missing.

If fun is a function, you get a single matrix.

References

For the weighted covariance:

- Canty, M.J. and A.A. Nielsen, 2008. Automatic radiometric normalization of multitemporal satellite imagery with the iteratively re-weighted MAD transformation. Remote Sensing of Environment 112:1025-1036.
- Nielsen, A.A., 2007. The regularized iteratively reweighted MAD method for change detection in multi- and hyperspectral data. IEEE Transactions on Image Processing 16(2):463-478.

See Also

global, cov.wt, weighted.mean

Examples

```
b <- rast(system.file("ex/logo.tif", package="terra"))
layerCor(b, "pearson")

layerCor(b, "cov")

# weigh by column number
w <- init(b, fun="col")
layerCor(b, "weighted.cov", w=w)
```

---

linearUnits *Linear units of the coordinate reference system*

Description

Get the linear units of the coordinate reference system (crs) of a SpatRaster or SpatVector expressed in m. The value returned is used internally to transform area and perimeter measures to meters. The value returned for longitude/latitude crs is zero.

Usage

```
## S4 method for signature 'SpatRaster'
linearUnits(x)

## S4 method for signature 'SpatVector'
linearUnits(x)
```

Arguments

x	SpatRaster or SpatVector

Value

numeric (meter)

See Also

`crs`

Examples

```
x <- rast()
crs(x) <- ""
linearUnits(x)

crs(x) <- "+proj=longlat +datum=WGS84"
linearUnits(x)

crs(x) <- "+proj=utm +zone=1 +units=cm"
linearUnits(x)

crs(x) <- "+proj=utm +zone=1 +units=km"
linearUnits(x)

crs(x) <- "+proj=utm +zone=1 +units=us-ft"
linearUnits(x)
```

---

lines *Add points, lines, or polygons to a map*

Description

Add a vector geometries to a plot (map) with `points`, `lines`, or `polys`.

These are simpler alternatives for `plot(x, add=TRUE)`

These methods also work for a small(!) SpatRaster. Only cells that are not NA in the first layer are used.

Usage

```
## S4 method for signature 'SpatVector'
points(x, col, cex=0.7, pch=16, alpha=1, ...)

## S4 method for signature 'SpatVector'
lines(x, y=NULL, col, lwd=1, lty=1, arrows=FALSE, alpha=1, ...)

## S4 method for signature 'SpatVector'
polys(x, col, border="black", lwd=1, lty=1, alpha=1, ...)

## S4 method for signature 'SpatRaster'
points(x, ...)

## S4 method for signature 'SpatRaster'
lines(x, mx=10000, ...)

## S4 method for signature 'SpatRaster'
polys(x, mx=10000, dissolve=TRUE, ...)

## S4 method for signature 'SpatExtent'
points(x, col="black", alpha=1, ...)

## S4 method for signature 'SpatExtent'
lines(x, col="black", alpha=1, ...)

## S4 method for signature 'SpatExtent'
polys(x, col, alpha=1, ...)
```

Arguments

`x`	SpatVector or SpatExtent
`y`	missing or SpatVector. If both `x` and `y` have point geometry and the same number of rows, lines are drawn between pairs of points
`col`	character. Colors
`border`	character. color(s) of the polygon borders. Use `NULL` or `NA` to not draw a border
`cex`	numeric. point size magnifier. See `par`
`pch`	positive integer, point type. See `points`. On some (linux) devices, the default symbol "16" is a not a very smooth circle. You can use "20" instead (it takes a bit longer to draw) or "1" for an open circle
`alpha`	number between 0 and 1 to set transparency
`lwd`	numeric, line-width. See `par`
`lty`	positive integer, line type. See `par`
`arrows`	logical. If `TRUE` and `y` is a SpatVector, arrows are drawn intead of lines. See ?`arrows` for additional arguments
`mx`	positive number. If the number of cells of SpatRaster `x` is higher, the method will fail with an error message

dissolve	logical. Should boundaries between cells with the same value be removed?
...	additional graphical arguments such as lwd, cex and pch

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

r <- rast(v)
values(r) <- 1:ncell(r)
plot(r)
lines(v)
points(v)
```

---

makeTiles *Make tiles or get their extents*

Description

Divide a SpatRaster into "tiles". The cells of another SpatRaster (normally with a much lower resolution) or a SpatVector with polygon geometry can be used to define the tiles. You can also provide one or two numbers to indicate the number of rows and columns per tile.

`getTileExtents` returns the extents of the (virtual) tiles, while `makeTiles` creates files for the tiles and returns their filenames.

Usage

```
## S4 method for signature 'SpatRaster'
makeTiles(x, y, filename="tile_.tif", extend=FALSE,
na.rm=FALSE, buffer=0, overwrite=FALSE, ...)

## S4 method for signature 'SpatRaster'
getTileExtents(x, y, extend=FALSE, buffer=0)
```

Arguments

x	SpatRaster
y	SpatRaster or SpatVector defining the zones; or numeric specifying the number of rows and columns for each zone (1 or 2 numbers if the number of rows and columns is not the same)
filename	character. Output filename template. Filenames will be altered by adding the tile number for each tile
extend	logical. If TRUE, the extent of y is expanded to assure that it covers all of x
na.rm	logical. If TRUE, tiles with only missing values are ignored

buffer	integer. The number of additional rows and columns added to each tile. Can be a single number, or two numbers to specify a separate number of rows and columns. This allows for creating overlapping tiles that can be used for computing spatial context dependent values with e.g. focal. The expansion is only inside x, no rows or columns outside of x are added
overwrite	logical. If TRUE, existing tiles are overwritten; otherwise they are skipped (without error or warning)
...	additional arguments for writing files as in writeRaster

Value

character (filenames) or matrix (extents)

See Also

vrt to create a virtual raster from tiles and crop for sub-setting arbitrary parts of a SpatRaster.

Examples

```
r <- rast(ncols=100, nrows=100)
values(r) <- 1:ncell(r)
x <- rast(ncols=2, nrows=2)

getTileExtents(r, x)
getTileExtents(r, x, buffer=3)


filename <- paste0(tempfile(), "_.tif")
ff <- makeTiles(r, x, filename)
ff

vrt(ff)
```

---

makeVRT *Make a VRT header file*

Description

Create a VRT header file for a "flat binary" raster file that needs a header file to be able to read it, but does not have it.

Usage

```
makeVRT(filename, nrow, ncol, nlyr=1, extent, xmin, ymin, xres, yres=xres, xycenter=TRUE,
  crs="+proj=longlat", lyrnms="", datatype, NAflag=NA, bandorder="BIL", byteorder="LSB",
   toptobottom=TRUE, offset=0, scale=1)
```

Arguments

`filename`	character. raster filename (without the ".vrt" extension)
`nrow`	positive integer, the number of rows
`ncol`	positive integer, the number of columns
`nlyr`	positive integer, the number of layers
`extent`	SpatExtent or missing
`xmin`	numeric. minimum x coordinate (only used if `extent` is missing)
`ymin`	numeric. minimum y coordinate (only used if `extent` is missing)
`xres`	positive number. x resolution
`yres`	positive number. y resolution)
`xycenter`	logical. If `TRUE`, `xmin` and `xmax` represent the coordinates of the center of the extreme cell, in stead of the coordinates of the outside corner. Only used of `extent` is missing
`crs`	character. Coordinate reference system description
`lyrnms`	character. Layer names
`datatype`	character. One of "INT2S", "INT4S", "INT1U", "INT2U", "INT4U", "FLT4S", "FLT8S". If missing, this is guessed from the file size (INT1U for 1 byte per value, INT2S for 2 bytes and FLT4S for 4 bytes per value). This may be wrong because, for example, 2 bytes per value may in fact be INT2U (with the U for unsigned) values
`NAflag`	numeric. The value used as the "NA flag"
`bandorder`	character. One of "BIL", "BIP", or "BSQ". That is Band Interleaved by Line, or by Pixel, or Band SeQuential
`byteorder`	character. One of "LSB", "MSB". "MSB" is common for files generated on Linux systems, whereas "LSB" is common for files generated on windows
`toptobottom`	logical. If `FALSE`, the values are read bottom to top
`offset`	numeric. offset to be applied
`scale`	numeric. scale to be applied

Value

character (.VRT filename)

See Also

`vrt` to create a vrt for a collection of raster tiles map.pal *color palettes for mapping*

Description

Get a color palette for mapping. These palettes were copied from GRASS.

Usage

```
map.pal(name, n=50, ...)
```

Arguments

`name`	character (name of a palette, see Details), or missing (to get the available names)
`n`	numeric. The number of colors
`...`	additional arguments that are passed to `colorRamp`

Details

Name	Description
aspect	aspect oriented grey colors
bcyr	blue through cyan through yellow to red
bgyr	blue through green through yellow to red
blues	white to blue
byg	blue through yellow to green
byr	blue through yellow to red
curvature	for terrain curvatures
differences	differences oriented colors
elevation	maps relative ranges of raster values to elevation color ramp
grass	GRASS GIS green (perceptually uniform)
greens	white to green
grey	grey scale
gyr	green through yellow to red
haxby	relative colors for bathymetry or topography
inferno	perceptually uniform sequential colors inferno
magma	perceptually uniform sequential colors
oranges	white to orange
plasma	perceptually uniform sequential colors
rainbow	rainbow colors
ramp	color ramp
random	random colors
reds	white to red
roygbiv	
rstcurv	terrain curvature
ryb	red through yellow to blue

ryg	red through yellow to green
sepia	yellowish-brown through to white
viridis	perceptually uniform sequential colors
water	water depth
wave	color wave

Value

none

See Also

terrain.colors

Examples

```
map.pal("elevation", 10)

r <- rast(system.file("ex/elev.tif", package="terra"))
plot(r, col=map.pal("elevation"))

map.pal()
```

---

map_extent *Get the coordinates of the extent of a map*

Description

Helper function for creating custom map elements that are aligned with the axes of a map (base plot created with a SpatRaster and/or SpatVector). For example, you may need to know the coordinates for the upper-left corner of a map to add some information there.

Unlike the standard base plot, terra keeps the axis aligned with the data. For that reason you cannot use `par()$usr` to get these coordinates.

The coordinates returned by this function are used in, for example, `add_legend` such that a legend can be automatically placed in the a particular corner.

This function only returns meaningful results of the active plot (canvas) was create with a call to `plot` with a SpatRaster or SpatVector as first argument.

Usage

```
map_extent()
```

See Also

add_legend, add_grid, add_box

Examples

```
r <- rast(xmin=0, xmax=10, ymin=0, ymax=10, res=1, vals=1:100)
plot(r)

map_extent()
par()$usr
```

---

mask *Mask values in a SpatRaster or SpatVector*

Description

If `x` is a `SpatRaster`: Create a new SpatRaster that has the same values as SpatRaster `x`, except for the cells that are `NA` (or other `maskvalue`) in another SpatRaster (the 'mask'), or the cells that are not covered by a SpatVector or SpatExtent. These cells become `NA` (or another `updatevalue`).

If `x` is a SpatVector or SpatExtent: Select geometries of `x` that intersect, or not intersect, with the geometries of `y`.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
mask(x, mask, inverse=FALSE, maskvalues=NA,
   updatevalue=NA, filename="", ...)

## S4 method for signature 'SpatRaster,SpatVector'
mask(x, mask, inverse=FALSE, updatevalue=NA,
touches=TRUE, filename="", ...)

## S4 method for signature 'SpatRaster,SpatExtent'
mask(x, mask, inverse=FALSE, updatevalue=NA,
touches=TRUE, filename="", ...)

## S4 method for signature 'SpatVector,SpatVector'
mask(x, mask, inverse=FALSE)

## S4 method for signature 'SpatVector,SpatExtent'
mask(x, mask, inverse=FALSE)
```

Arguments

`x`	SpatRaster or SpatVector
`mask`	SpatRaster or SpatVector
`inverse`	logical. If `TRUE`, areas on mask that are _not_ the `maskvalue` are masked
`maskvalues`	numeric. The value(s) in `mask` that indicate which cells of `x` should be masked (change their value to `updatevalue` (default = `NA`))
`updatevalue`	numeric. The value that masked cells should become (if they are not `NA`)

touches — logical. If TRUE, all cells touched by lines or polygons will be masked, not just those on the line render path, or whose center point is within the polygon

filename — character. Output filename

... — additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

subst, crop

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
msk <- ifel(r < 400, NA, 1)

m <- mask(r, msk)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)[1,]

mv1 <- mask(r, v)
mv2 <- crop(r, v, mask=TRUE)
```

---

match *Value matching for SpatRasters*

Description

match returns a SpatRaster with the position of the matched values. The cell values are the index of the table argument.

%in% returns a 0/1 (FALSE/TRUE) SpatRaster indicating if the cells values were matched or not.

Usage

```
match(x, table, nomatch = NA_integer_, incomparables = NULL)

x %in% table
```

Arguments

x	SpatRaster
table	vector of the values to be matched against
nomatch	the value to be returned in the case when no match is found. Note that it is coerced to integer
incomparables	a vector of values that cannot be matched. Any value in x matching a value in this vector is assigned the nomatch value. For historical reasons, FALSE is equivalent to NULL

Value

SpatRaster

See Also

app, match

Examples

```
r <- rast(nrows=10, ncols=10)
values(r) <- 1:100
m <- match(r, c(5:10, 50:55))
n <- r %in% c(5:10, 50:55)
```

Math-methods *General mathematical methods*

Description

Standard mathematical methods for computations with SpatRasters. Computations are local (applied on a cell by cell basis). If multiple SpatRasters are used, these must have the same extent and resolution. These have been implemented:

`abs, sign, sqrt, ceiling, floor, trunc, cummax, cummin, cumprod, cumsum, log, log10, log2, log1p, acos, acosh, asin, asinh, atan, atanh, exp, expm1, cos, cosh, sin, sinh, tan, tanh, round, signif`

Instead of directly calling these methods, you can also provide their name to the `math` method. This is useful if you want to provide an output filename.

The following methods have been implemented for `SpatExtent`: `round, floor, ceiling`

`round` has also been implemented for `SpatVector`, to round the coordinates of the geometries.

Usage

```
## S4 method for signature 'SpatRaster'
sqrt(x)

## S4 method for signature 'SpatRaster'
log(x, base=exp(1))

## S4 method for signature 'SpatRaster'
round(x, digits=0)

## S4 method for signature 'SpatRaster'
math(x, fun, digits=0, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatVector'
round(x, digits=4)

## S4 method for signature 'SpatRaster'
cumsum(x)
```

Arguments

`x`	SpatRaster
`base`	a positive or complex number: the base with respect to which logarithms are computed
`digits`	Number of digits for rounding
`fun`	character. Math function name
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster or SpatExtent

See Also

See `app` to use mathematical functions not implemented by the package, and `Arith-methods` for arithmetical operations. Use `roll` for rolling functions.

Examples

```
r1 <- rast(ncols=10, nrows=10)
v <- runif(ncell(r1))
v[10:20] <- NA
values(r1) <- v
r2 <- rast(r1)
values(r2) <- 1:ncell(r2) / ncell(r2)
r <- c(r1, r2)
```

```
s <- sqrt(r)
# same as
math(r, "sqrt")

round(s, 1)

cumsum(r)
```

---

mem *Memory available and needed*

Description

mem_info prints the amount of RAM that is required and available to process a SpatRaster.

free_RAM returns the amount of RAM that is available

Usage

```
mem_info(x, n=1)

free_RAM()
```

Arguments

x	SpatRaster
n	positive integer. The number of copies of x that are needed

Value

free_RAM returns the amount of available RAM in kilobytes

Examples

```
mem_info(rast())

free_RAM()
```

merge *Merge SpatRasters, or merge a SpatVector with a data.frame*

Description

Merge multiple SpatRasters to create a new SpatRaster with a larger spatial extent. The SpatRasters must have the same origin and spatial resolution. In areas where the SpatRasters overlap, the values of the SpatRaster that is first in the sequence of arguments (or in the SpatRasterCollection) will be retained (unless `first=FALSE`). Cells that are `NA` are ignored unless `na.rm=FALSE`

There is also a method for merging SpatVector with a data.frame; that is, to join the data.frame to the attribute table of the SpatVector.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
merge(x, y, ..., first=TRUE, na.rm=TRUE, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterCollection,missing'
merge(x, first=TRUE, na.rm=TRUE, filename="", ...)

## S4 method for signature 'SpatVector,data.frame'
merge(x, y, ...)
```

Arguments

`x`	SpatRaster, SpatRasterCollection, or SpatVector
`y`	missing if `x` is a SpatRasterCollection. SpatRaster if `x` is a SpatRaster. data.frame if `x` is a SpatVector
`...`	if `x` is a SpatRaster: additional objects of the same class as `x`. If `x` is a SpatRaster-Collection: options for writing files as in `writeRaster`. If `x` is a SpatVector, the same arguments as in `merge`
`first`	logical. If `TRUE`, in areas where rasters overlap, the first value is used. Otherwise the last value is used
`na.rm`	logical. If `TRUE` values that are `NA` are ignored
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Value

SpatRaster or SpatVector

See Also

Combining tiles with vrt may be more efficient than using merge. See mosaic for averaging overlapping regions.

See classify to merge a SpatRaster and a data.frame and union to combine SpatExtent objects.

Examples

```
x <- rast(xmin=-110, xmax=-80, ymin=40, ymax=70, res=1, vals=1)
y <- rast(xmin=-85, xmax=-55, ymax=60, ymin=30, res=1, vals=2)
z <- rast(xmin=-60, xmax=-30, ymax=50, ymin=20, res=1, vals=3)

m1 <- merge(x, y, z)
m2 <- merge(z, y, x)
m3 <- merge(y, x, z)

# if you have many SpatRasters, it may be convenient
# to make a SpatRasterCollection
rlist <- list(x, y, z)
rsrc <- sprc(rlist)

m <- merge(rsrc)


## SpatVector with data.frame
f <- system.file("ex/lux.shp", package="terra")
p <- vect(f)
dfr <- data.frame(District=p$NAME_1, Canton=p$NAME_2, Value=round(runif(length(p), 100, 1000)))
dfr <- dfr[1:5, ]
pm <- merge(p, dfr, all.x=TRUE, by.x=c('NAME_1', 'NAME_2'), by.y=c('District', 'Canton'))
pm
values(pm)
```

---

mergeTime *merge SpatRasters by timelines to create a single timeseries*

Description

Combine SpatRasters with partly overlapping time-stamps to create a single time series. If there is no overlap between the SpatRasters there is no point in using this function (use c instead).

Also note that time gaps are not filled. You can use fillTime to do that.

Usage

```
## S4 method for signature 'SpatRasterDataset'
mergeTime(x, fun=mean, filename="", ...)
```

Arguments

x	SpatRasterDataset
fun	A function that reduces a vector to a single number, such as `mean` or `min`
filename	character. Output filename
...	list with named options for writing files as in `writeRaster`

Value

SpatRaster

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
s1 <- c(r, r)
time(s1) <- as.Date("2001-01-01") + 0:5
s1 <- s1/10
time(s1) <- as.Date("2001-01-07") + 0:5
s2 <- s1*10
time(s2) <- as.Date("2001-01-05") + 0:5
x <- sds(s1, s1, s2)

m <- mergeTime(x, mean)
```

meta *meta*

Description

Get metadata associated with the sources or layers of a SpatRaster

Usage

```
## S4 method for signature 'SpatRaster'
meta(x, layers=FALSE)
```

Arguments

x	SpatRaster
layers	logical. Should the layer level metadata be returned?

Value

list

metags *Set or get metadata*

Description

You can set arbitrary metadata to (layers of) a SpatRaster using "name=value" tags. When wring a SpatRaster to a GTiff file, these tags are written to file.

Usage

```
## S4 replacement method for signature 'SpatRaster'
metags(x, layer=NULL)<-value

## S4 method for signature 'SpatRaster'
metags(x, layer=NULL, name=NULL)
```

Arguments

x	SpatRaster
layer	NULL or positive integer. If the value is NULL, the tags assigned or returned are for the SpatRaster. Otherwise for the layer number(s)
name	character
value	character of "name=value" or two-column matrix

Value

SpatRaster (`metags<-`), or named character (`metags`)

Examples

```
r <- rast(ncol=5, nrow=5)
m <- cbind(c("one", "two", "three"), c("ABC", "123", "hello"))
metags(r) <- m
metags(r)

metags(r) <- c("another_tag=another_value", "one more=this value")
metags(r)

metags(r) <- c(another_tag="44", `one more`="that value")
metags(r)


metags(r, name="two")

# remove a tag
metags(r) <- cbind("one", "")
metags(r) <- "two="
metags(r)
```

```
# remove all metags
metags(r) <- NULL
metags(r)
```

modal	*modal value*

Description

Compute the mode for each cell across the layers of a SpatRaster. The mode, or modal value, is the most frequent value in a set of values.

Usage

```
## S4 method for signature 'SpatRaster'
modal(x, ..., ties="first", na.rm=FALSE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`...`	additional argument of the same type as `x` or numeric
`ties`	character. Indicates how to treat ties. Either "random", "lowest", "highest", "first", or "NA"
`na.rm`	logical. If `TRUE`, `NA` values are ignored. If `FALSE`, `NA` is returned if `x` has any `NA` values
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Value

SpatRaster

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
r <- c(r/2, r, r*2)
m <- modal(r)
```

mosaic *mosaic SpatRasters*

Description

Combine adjacent and (partly) overlapping SpatRasters to form a single new SpatRaster. Values in overlapping cells are averaged (by default) or can be computed with another function.

The SpatRasters must have the same origin and spatial resolution.

This method is similar to the simpler, but much faster, `merge` method.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
mosaic(x, y, ..., fun="mean", filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterCollection,missing'
mosaic(x, fun="mean", filename="", ...)
```

Arguments

`x`	SpatRaster
`y`	object of same class as `x`
`...`	additional SpatRasters
`fun`	character. One of "mean", "median", "min", "max", "modal", "sum", "first", "last"
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Value

SpatRaster

See Also

`merge`

Examples

```
x <- rast(xmin=-110, xmax=-60, ymin=40, ymax=70, res=1, vals=1)
y <- rast(xmin=-95, xmax=-45, ymax=60, ymin=30, res=1, vals=2)
z <- rast(xmin=-80, xmax=-30, ymax=50, ymin=20, res=1, vals=3)

m1 <- mosaic(x, y, z)

m2 <- mosaic(z, y, x)
```

```
# with many SpatRasters, make a SpatRasterCollection from a list
rlist <- list(x, y, z)
rsrc <- sprc(rlist)

m <- mosaic(rsrc)
```

---

na.omit *Find and remove geometries that are NA*

Description

Find geometries that are NA; or remove geometries and/or records that are NA.

Usage

```
## S4 method for signature 'SpatVector'
is.na(x)

## S4 method for signature 'SpatVector'
na.omit(object, field=NA, geom=FALSE)
```

Arguments

x	SpatVector
object	SpatVector
field	character or NA. If NA, missing values in the attributes are ignored. Other values are either one or more field (variable) names, or "" to consider all fields
geom	logical. If TRUE empty geometries are removed

Value

SpatVector

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$test <- c(1,2,NA)
nrow(v)
x <- na.omit(v, "test")
nrow(x)
```

NAflag *Set the NA flag*

Description

The main purpose of this method is to allow correct reading of a SpatRaster that is based on a file that has an incorrect NA flag. The file is not changed, but flagged value is set to NA when values are read from the file ("lazy evaluation"). In contrast, if the values are in memory the change is made immediately.

To change values, it is generally better to use `classify`

Usage

```
## S4 method for signature 'SpatRaster'
NAflag(x)

## S4 replacement method for signature 'SpatRaster'
NAflag(x)<-value
```

Arguments

`x`	SpatRaster
`value`	numeric. The value to be interpreted as NA; set this before reading the values from the file. This can be a single value, or multiple values, one for each data source (file / subdataset)

Value

none or numeric

See Also

`classify`

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))[[1]]
NAflag(s) <- 255
plot(s)
NAflag(s)
```

names — *Names of Spat* objects*

Description

Get or set the names of the layers of a SpatRaster or the attributes of a SpatVector.

See `set.names` for in-place setting of names.

Usage

```
## S4 method for signature 'SpatRaster'
names(x)

## S4 replacement method for signature 'SpatRaster'
names(x)<-value

## S4 method for signature 'SpatRasterDataset'
names(x)

## S4 replacement method for signature 'SpatRasterDataset'
names(x)<-value

## S4 method for signature 'SpatVector'
names(x)

## S4 replacement method for signature 'SpatVector'
names(x)<-value
```

Arguments

`x`	SpatRaster, SpatRasterDataset, or SpatVector
`value`	character (vector)

Value

character

Note

terra enforces neither unique nor valid names. See `make.unique` to create unique names and `make.names` to make syntactically valid names.

Examples

```
s <- rast(ncols=5, nrows=5, nlyrs=3)
nlyr(s)
names(s)
```

```
names(s) <- c("a", "b", "c")
names(s)

# SpatVector names
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
names(v)
names(v) <- paste0(substr(names(v), 1, 2), "_", 1:ncol(v))
names(v)
```

---

nearest *nearby geometries*

Description

Identify geometries that are near to each other. Either get the index of all geometries within a certain distance, or the k nearest neighbors, or (with `nearest`) get the nearest points between two geometries.

Usage

```
## S4 method for signature 'SpatVector'
nearby(x, y=NULL, distance=0, k=1, centroids=TRUE, symmetrical=TRUE)

## S4 method for signature 'SpatVector'
nearest(x, y, pairs=FALSE, centroids=TRUE, lines=FALSE)
```

Arguments

`x`	SpatVector
`y`	SpatVector or NULL
`distance`	numeric. maximum distance
`k`	positive integer. number of neighbors. Ignored if `distance > 0`
`centroids`	logical. Should the centroids of polygons be used?
`symmetrical`	logical. If `TRUE`, a near pair is only included once. That is, if geometry 1 is near to geometry 3, the implied nearness between 3 and 1 is not reported. Ignored if `k` neighbors are returned
`pairs`	logical. If `TRUE` pairwise nearest points are returned (only relevant when using at least one SpatVector of lines or polygons
`lines`	logical. If `TRUE` lines between the nearest points instead of (the nearest) points

Value

matrix

See Also

relate, adjacent

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
nearby(v, distance=12000)
```

NIDP *Number of immediate adjacent cells flowing into each cell*

Description

Compute the number of immediate adjacent cells flowing into each cell

Usage

```
## S4 method for signature 'SpatRaster'
NIDP(x, filename="",...)
```

Arguments

x	SpatRaster with flow-direction. see terrain
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Details

NDIP is computed first to compute flow-accumulation with the algorithm by Zhou at al, 2019.

Value

SpatRaster

Author(s)

Emanuele Cordano

References

Zhou, G., Wei, H. & Fu, S. A fast and simple algorithm for calculating flow accumulation matrices from raster digital elevation. Front. Earth Sci. 13, 317–326 (2019). https://doi.org/10.1007/s11707-018-0725-9 https://link.springer.com/article/10.1007/s11707-018-0725-9

See Also

flowAccumulation

Examples

```
elev1 <- array(NA,c(9,9))
elev2 <- elev1
dx <- 1
dy <- 1
for (r in 1:nrow(elev1)) {
  y <- (r-5)*dx
  for (c in 1:ncol(elev1)) {

    x <- (c-5)*dy
    elev1[r,c] <- 5*(x^2+y^2)
    elev2[r,c] <- 10+5*(abs(x))-0.001*y ### 5*(x^2+y^2)
  }
}


## Elevation Raster
elev1 <- rast(elev1)
elev2 <- rast(elev2)

t(array(elev1[],rev(dim(elev1)[1:2])))
t(array(elev2[],rev(dim(elev2)[1:2])))

plot(elev1)
plot(elev2)

## Flow Direction Raster
flowdir1<- terrain(elev1,v="flowdir")
flowdir2<- terrain(elev2,v="flowdir")


t(array(flowdir1[],rev(dim(flowdir1)[1:2])))
t(array(flowdir2[],rev(dim(flowdir2)[1:2])))

plot(flowdir1)
plot(flowdir2)

##
nidp1 <- NIDP((flowdir1))
nidp2 <- NIDP((flowdir2))

t(array(nidp1[],rev(dim(nidp1)[1:2])))
t(array(nidp2[],rev(dim(nidp2)[1:2])))

plot(nidp1)
plot(nidp2)
```

`normalize.longitude` *normalize vector data that crosses the dateline*

Description

Normalize the longitude of geometries, move them if they are outside of the -180 to 180 degrees range.

Usage

```
## S4 method for signature 'SpatVector'
normalize.longitude(x)
```

Arguments

x	SpatVector

Value

SpatVector

See Also

rotate for SpatRaster

Examples

```
p <- vect("POLYGON ((120 10, 230 75, 230 -75, 120 10))")
normalize.longitude(p)
```

---

north *North arrow*

Description

Add a (North) arrow to a map

Usage

```
north(xy=NULL, type=1, label="N", angle=0, d, head=0.1, xpd=TRUE, ...)
```

Arguments

xy	numeric. x and y coordinate to place the arrow. It can also be one of following character values: "bottomleft", "bottom", "bottomright", topleft", "top", "topright", "left", "right", or NULL
type	integer between 1 and 12, or a character (unicode) representation of a right pointing arrow such as "\u27A9"
label	character, to be printed near the arrow
angle	numeric. The angle of the arrow in degrees

d	numeric. Distance covered by the arrow in plot coordinates. Only applies to type=1
head	numeric. The size of the arrow "head", for type=1
xpd	logical. If TRUE, the scale bar or arrow can be outside the plot area
...	graphical arguments to be passed to other methods

Value

none

See Also

sbar, plot, inset

Examples

```
f <- system.file("ex/meuse.tif", package="terra")
r <- rast(f)
plot(r)
north()
north(c(178550, 332500), d=250)

## Not run:
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
plot(r, type="interval")
sbar(15, c(6.3, 50), type="bar", below="km", label=c(0,7.5,15), cex=.8)
north(type=3, cex=.8)
north(xy=c(6.7, 49.9), type=2, angle=45, label="NE")
north(xy=c(6.6, 49.7), type=5, cex=1.25)
north(xy=c(5.5, 49.6), type=9)
north(d=.05, xy=c(5.5, 50), angle=180, label="S", lwd=2, col="blue")


## all arrows
r <- rast(res=10)
values(r) <- 1
plot(r, col="white", axes=FALSE, legend=FALSE, mar=c(0,0,0,0), reset=TRUE)
for (i in 1:12) {
x = -200+i*30
north(xy=cbind(x,30), type=i)
text(x, -20, i, xpd=TRUE)
}


## End(Not run)
```

not.na *is not NA*

Description

Shortcut method to avoid the two-step `!is.na(x)`

Usage

```
## S4 method for signature 'SpatRaster'
not.na(x, falseNA, filename="", ...)
```

Arguments

`x`	SpatRaster
`falseNA`	logical. Should the result be `TRUE, NA` instead of `TRUE, FALSE`?
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`Compare-methods`

Examples

```
r <- rast(ncols=10, nrows=10, vals=1)
r[10:20] <- NA
x <- not.na(r)
```

options *Options*

Description

Get or set general options.

Usage

```
terraOptions(..., print=TRUE)
```

Arguments

`...`	option names and values (see Details). Or missing, to get or show the current options
`print`	logical. If `TRUE` the option names and values are printed

Details

The following options are available.

memfrac - value between 0 and 0.9 (larger values give a warning). The fraction of RAM that may be used by the program.

memmin - if memory required is below this threshold (in GB), the memory is assumed to be available. Otherwise, terra checks if it is available.

memmax - the maximum amount of RAM (in GB) that terra is allowed to use when processing a raster dataset. Should be less than what is detected (see `mem_info`), and higher values are ignored. Set it to a negative number or NA to not set this option. `terraOptions` only shows the value of `memmax` if it is set.

tempdir - directory where temporary files are written. The default what is returned by `tempdir()`.

datatype - default data type. See `writeRaster`

todisk - logical. If `TRUE` write all raster data to disk (temp file if no file name is specified). For debugging.

progress - non-negative integer. A progress bar is shown if the number of chunks in which the data is processed is larger than this number. No progress bar is shown if the value is zero

verbose - logical. If `TRUE` debugging info is printed for some functions

Value

list. Invisibly if `print=TRUE`

Note

It is possible to set your own default options in "etc/.Rprofile.site" of your R installation like this

`options(terra_default=list(tempdir="d:/temp", memfrac=.4))`

But that may not be a good practice. It is clearer to set your favorite options at the beginning of each script.

Examples

```
terraOptions()
terraOptions(memfrac=0.5, tempdir = "c:/temp")
terraOptions(progress=10)
terraOptions()
```

origin *Origin*

Description

Get or set the coordinates of the point of origin of a SpatRaster. This is the point closest to (0, 0) that you could get if you moved towards that point in steps of the x and y resolution.

Usage

```
## S4 method for signature 'SpatRaster'
origin(x)

## S4 replacement method for signature 'SpatRaster'
origin(x)<-value
```

Arguments

x	SpatRaster
value	numeric vector of length 1 or 2

Value

A vector of two numbers (x and y coordinates)

Examples

```
r <- rast(xmin=-0.5, xmax = 9.5, ncols=10)
origin(r)
origin(r) <- c(0,0)
r
```

pairs *Pairs plot (matrix of scatterplots)*

Description

Pair plots of layers in a SpatRaster. This is a wrapper around graphics function pairs.

Usage

```
## S4 method for signature 'SpatRaster'
pairs(x, hist=TRUE, cor=TRUE, use="pairwise.complete.obs", maxcells=100000, ...)
```

Arguments

`x`	SpatRaster
`hist`	logical. If TRUE a histogram of the values is shown on the diagonal
`cor`	logical. If TRUE the correlation coefficient is shown in the upper panels
`use`	argument passed to the `cor` function
`maxcells`	integer. Number of pixels to sample from each layer of a large SpatRaster
`...`	additional arguments (graphical parameters)

See Also

`boxplot`, `hist`

Examples

```
r <-rast(system.file("ex/elev.tif", package="terra"))
s <- c(r, 1/r, sqrt(r))
names(s) <- c("elevation", "inverse", "sqrt")
pairs(s)

# to make indvidual histograms:
hist(r)
# or scatter plots:
plot(s[[1]], s[[2]])
```

`panel` *Map panel*

Description

Show multiple maps that share a single legend.

Usage

```
## S4 method for signature 'SpatRaster'
panel(x, main, loc.main="topleft", nc, nr, maxnl=16,
maxcell=500000, box=FALSE, pax=list(), plg=list(), range=NULL, ...)
```

Arguments

`x`	SpatRaster
`main`	character. Main plot titles (one for each layer to be plotted). You can use arguments `cex.main`, `font.main`, `col.main` to change the appearance
`loc.main`	numeric of character to set the location of the main title. Either two coordinates, or a character value such as "topleft")
`nc`	positive integer. Optional. The number of columns to divide the plotting device in (when plotting multiple layers)

nr	positive integer. Optional. The number of rows to divide the plotting device in (when plotting multiple layers)
maxnl	positive integer. Maximum number of layers to plot (for a multi-layer object)
maxcell	positive integer. Maximum number of cells to use for the plot
box	logical. Should a box be drawn around the map?
plg	see plot
pax	see plot
range	numeric. minimum and maximum values to be used for the continuous legend
...	arguments passed to plot("SpatRaster", "numeric") and additional graphical arguments

See Also

plot and see rasterVis::levelplot and tidyterra::autoplot for more sophisticated panel plots.

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
v <- vect(system.file("ex/lux.shp", package="terra"))
x <- c(r, r/2, r*2, r)
names(x) <- paste0("(", LETTERS[1:4], ")")
panel(x)
panel(x, fun=\()lines(v), loc.main="topright")
```

---

patches *Detect patches (clumps) of cells*

Description

Detect patches (clumps). Patches are groups of cells that are surrounded by cells that are NA. Set zeroAsNA to TRUE to also identify patches separated by cells with values of zero.

Usage

```
## S4 method for signature 'SpatRaster'
patches(x, directions=4, values=FALSE, zeroAsNA=FALSE, allowGaps=TRUE, filename="", ...)
```

Arguments

x	SpatRaster
directions	integer indicating which cells are considered adjacent. Should be 8 (Queen's case) or 4 (Rook's case)
values	logical. If TRUE use cell values to distinguish patches. If FALSE, all cells that are not NA are considered identical

zeroAsNA logical. If TRUE treat cells that are zero as if they were NA. Ignored if byvalue=TRUE

allowGaps logical. If TRUE there may be gaps in the patch IDs (e.g. you may have patch IDs 1, 2, 3 and 5, but not 4). If it is FALSE, these numbers will be recoded from 1 to the number of patches (4 in this example)

filename character. Output filename

... options for writing files as in writeRaster

Value

SpatRaster. Cell values are patch numbers

See Also

focal, boundaries

Examples

```
r <- rast(nrows=18, ncols=36, xmin=0)
r[1:2, 5:8] <- 1
r[5:8, 2:6] <- 1
r[7:12, 22:36] <- 1
r[15:16, 18:29] <- 1
p <- patches(r)

# zero as background instead of NA
r <- rast(nrows=10, ncols=10, xmin=0, vals=0)
r[3, 3] <- 10
r[4, 4] <- 10
r[5, 5:8] <- 12
r[6, 6:9] <- 12

# treat zeros as NA

p4 <- patches(r, zeroAsNA=TRUE)
p8 <- patches(r, 8, zeroAsNA=TRUE)

### patches for different values
p <- patches(r, values=TRUE)

### patch ID values are not guaranteed to be consecutive
r <- rast(nrows=5, ncols=10, xmin=0)
set.seed(0)
values(r)<- round(runif(ncell(r))*0.7)
rp <- patches(r, directions=8, zeroAsNA=TRUE)
plot(rp, type="classes"); text(rp)

## unless you set allowGaps=FALSE
rp <- patches(r, directions=8, zeroAsNA=TRUE, allowGaps=FALSE)
plot(rp, type="classes"); text(rp)

### use zonal to remove small patches
```

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- classify(r, cbind(-Inf, 400, NA))
y <- patches(x)
# remove patches smaller than 100 ha
rz <- zonal(cellSize(y, unit="ha"), y, sum, as.raster=TRUE)
s <- ifel(rz < 250, NA, y)
```

`perim` *Perimeter or length*

Description

This method returns the length of lines or the perimeter of polygons.

When the crs is not longitude/latitude, you may get more accurate results by first unprojecting the SpatVector (you can use `project` to transform the crs to longitude/latitude)

Usage

```
## S4 method for signature 'SpatVector'
perim(x)
```

Arguments

`x`	SpatVector

Value

numeric (m)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
perim(v)
```

`persp` *Perspective plot*

Description

Perspective plot of a SpatRaster. This is an implementation of a generic function in the graphics package.

Usage

```
## S4 method for signature 'SpatRaster'
persp(x, maxcells=100000, ...)
```

Arguments

`x`	SpatRaster. Only the first layer is used
`maxcells`	integer > 0. Maximum number of cells to use for the plot. If `maxpixels < ncell(x)`, `spatSample(method="regular")` is used before plotting
`...`	Any argument that can be passed to `persp` (graphics package)

See Also

`persp`, `contour`, `plot`

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
persp(r)
```

---

pitfinder — *Pit Finder in a Flow Dir SpatRaster for Watershed Extraction*

Description

find pits (depressions with no outlet)

Usage

```
## S4 method for signature 'SpatRaster'
pitfinder(x,filename="",...)
```

Arguments

`x`	SpatRaster wih flow-direcion. See `terrain`
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

A `SpatRaster-class` (raster) map containing value 1 for the pits and value 0 elsewhere.

Author(s)

Emanuele Cordano

See Also

terrain,watershed,flowAccumulation,NIDP

Examples

```
## Creation of a Digital Elevation Model

elev <- array(NA,c(9,9))
dx <- 1
dy <- 1
for (r in 1:nrow(elev)) {
  x <- (r-5)*dx
  for (c in 1:ncol(elev)) {

    y <- (c-5)*dy
    elev[r,c] <- 10+5*(x^2+y^2)
    }
  }

elev <- cbind(elev,elev,elev,elev)
elev <- rbind(elev,elev,elev,elev)
elev <- rast(elev)

## Flow Directions

flowdir<- terrain(elev,v="flowdir")
t(array(flowdir[],rev(dim(flowdir)[1:2])))

## Pit Detect

pits <- pitfinder(flowdir)

## Application wth example DEM

elev <- rast(system.file('ex/elev.tif',package="terra"))
flowdir <- terrain(elev,"flowdir")

pits <- pitfinder(flowdir)
```

---

plet *Plot with leaflet*

Description

Plot the values of a SpatRaster or SpatVector to make an interactive leaflet map that is displayed in a browser.

Usage

```
## S4 method for signature 'SpatRaster'
plet(x, y=1, col, alpha=0.8, main=names(x),
tiles=c("Streets", "Esri.WorldImagery", "OpenTopoMap"),
wrap=TRUE, maxcell=500000, stretch=NULL, legend="bottomright",
shared=FALSE, panel=FALSE, collapse=TRUE, map=NULL)


## S4 method for signature 'SpatVector'
plet(x, y="", col,fill=0.2, main=y, cex=1, lwd=2,
border="black", alpha=1, popup=TRUE, label=FALSE, split=FALSE,
tiles=c("Streets", "Esri.WorldImagery", "OpenTopoMap"),
wrap=TRUE, legend="bottomright", collapse=FALSE, type=NULL, breaks=NULL,
breakby="eqint", sort=TRUE, decreasing=FALSE, map=NULL, ...)


## S4 method for signature 'SpatVectorCollection'
plet(x, col, fill=0, cex=1, lwd=2, border="black", alpha=1, popup=TRUE,
  label=FALSE, tiles=c("Streets", "Esri.WorldImagery", "OpenTopoMap"), wrap=TRUE,
   legend="bottomright", collapse=FALSE, map=NULL)


## S4 method for signature 'leaflet'
lines(x, y, col, lwd=2, alpha=1, ...)

## S4 method for signature 'leaflet'
points(x, y, col, cex=1, alpha=1, label=1:nrow(y), popup=FALSE, ...)

## S4 method for signature 'leaflet'
polys(x, y, col, fill=0.2, lwd=2, border="black", alpha=1, popup=TRUE, label=FALSE, ...)
```

Arguments

x	SpatRaster, SpatVector, or leaflet object
y	missing, or positive integer, or character (variable or layer name) indicating the layer(s) to be plotted. If x is a SpatRaster, you can select multiple layers
col	character. Vector of colors or color generating function
alpha	Number between 0 and 1 to set the transparency for lines (0 is transparent, 1 is opaque)
fill	Number between 0 and 1 to set the transparency for polygon areas (0 is transparent, 1 is opaque)
tiles	character or NULL. Names of background tile providers
wrap	logical. if TRUE, tiles wrap around
maxcell	positive integer. Maximum number of cells to use for the plot
stretch	NULL or character ("lin" or "hist") to stretch RGB rasters. See plotRGB

legend	character to indicate the legend position ("bottomleft", "bottomright", "topleft" or "topright") or NULL to suppress the legend
main	character. Title for the legend. The length should be 1 if x is a SpatVector and length nlyr(x) if x is a SpatVector
shared	logical. Should the legend be the same for all rasters (if multiple layers of SpatRaster x are mapped)
map	leaflet object
...	additional arguments for drawing points, lines, or polygons passed on the the relevant leaflet function
border	character. Color for the polygon borders
collapse	logical. Should the layers "control" panel be collapsed?
split	logical. If TRUE a check-box is created to toggle each value in y (If x is a SpatVector)
cex	numeric. point size magnifier. See par
lwd	numeric, line-width. See par
popup	logical. Should pop-ups be created?
label	logical. Should mouse-over labels be added?
panel	logical. Should SpatRaster layers be shown as a panel"
type	character. Type of map/legend. One of "classes", or "interval". If not specified, the type is chosen based on the data. Use "" to suppress the legend
breaks	numeric. Either a single number to indicate the number of breaks desired, or the actual breaks. When providing this argument, the default legend becomes "interval"
breakby	character or function. Either "eqint" for equal interval breaks, "cases" for equal quantile breaks. If a function is supplied it should take a single argument (a vector of values) and create groups
sort	logical. If TRUE legends with character values are sorted. You can also supply a vector of the unique values, in the order in which you want them to appear in the legend
decreasing	logical. If TRUE, legends are sorted in decreasing order

See Also

plot

Examples

```
## Not run:
if (require(leaflet) && (packageVersion("leaflet") > "2.1.1")) {

v <- vect(system.file("ex/lux.shp", package="terra"))
p <- spatSample(as.polygons(v, ext=T), 30)
values(p) = data.frame(id=11:40, name=letters[1:30])
```

```
m <- plet(v, "NAME_1", tiles="", border="blue")
m <- points(m, p, col="red", cex=2, popup=T)
lines(m, v, lwd=1, col="white")

plet(v, "NAME_1", split=TRUE, alpha=.2) |>
  points(p, col="gray", cex=2, popup=TRUE,
   clusterOptions = markerClusterOptions())

s <- svc(v, p)
names(s) <- c("the polys", "set of points")
plet(s, col=c("red", "blue"), lwd=1)


r <- rast(system.file("ex/elev.tif", package="terra"))
plet(r, main="Hi\nthere", tiles=NULL) |> lines(v, lwd=1)

plet(r, tiles="OpenTopoMap") |> lines(v, lwd=2, col="blue")

x <- c(r, 50*classify(r, 5))
names(x) <- c("first", "second")

# each their own legend
plet(x, 1:2, collapse=FALSE) |> lines(v, lwd=2, col="blue")

# shared legend
plet(x, 1:2, shared=TRUE, collapse=FALSE) |> lines(v, lwd=2, col="blue")

}
## End(Not run)
```

`plot` *Make a map*

Description

Plot the values of a SpatRaster or SpatVector to make a map.

See `points`, `lines` or `polys` to add a SpatVector to an existing map (or use argument `add=TRUE`).

There is a separate help file for plotting a `SpatGraticule` or `SpatExtent`.

Usage

```
## S4 method for signature 'SpatRaster,numeric'
plot(x, y=1, col, type=NULL, mar=NULL, legend=TRUE, axes=!add, plg=list(), pax=list(),
    maxcell=500000, smooth=FALSE, range=NULL, fill_range=FALSE,
levels=NULL, all_levels=FALSE, breaks=NULL, breakby="eqint", fun=NULL,
colNA=NULL, alpha=NULL, sort=FALSE,  decreasing=FALSE, grid=FALSE, ext=NULL,
reset=FALSE, add=FALSE, buffer=FALSE, background=NULL, box=axes, clip=TRUE, ...)

## S4 method for signature 'SpatRaster,missing'
```

```
plot(x, y, main, mar=NULL, nc, nr, maxnl=16, maxcell=500000, add=FALSE, ...)

## S4 method for signature 'SpatRaster,character'
plot(x, y, ...)

## S4 method for signature 'SpatVector,character'
plot(x, y, col=NULL, type=NULL, mar=NULL, add=FALSE, legend=TRUE, axes=!add,
    main="", buffer=TRUE, background=NULL, grid=FALSE, ext=NULL, sort=TRUE,
decreasing=FALSE, plg=list(), pax=list(), nr, nc, colNA=NA, alpha=NULL,
box=axes, clip=TRUE, ...)

## S4 method for signature 'SpatVector,numeric'
plot(x, y, ...)

## S4 method for signature 'SpatVector,missing'
plot(x, y, values=NULL, ...)

## S4 method for signature 'SpatVectorCollection,missing'
plot(x, y, main, mar=NULL, nc, nr, maxnl=16, ...)
```

Arguments

`x`	SpatRaster or SpatVector
`y`	missing or positive integer or name indicating the layer(s) to be plotted
`col`	character vector to specify the colors to use. The default is `map.pal("viridis", 100)`. The default can be changed with the `terra.pal` option. For example: `options(terra.pal=terrain.colors(10))`. If `x` is a `SpatRaster`, it can also be a `data.frame` with two columns (value, color) to get a "classes" type legend or with three columns (from, to, color) to get an "interval" type legend
`type`	character. Type of map/legend. One of "continuous", "classes", or "interval". If not specified, the type is chosen based on the data
`mar`	numeric vector of length 4 to set the margins of the plot (to make space for the legend). The default is (3.1, 3.1, 2.1, 7.1) for a single plot with a legend and (3.1, 3.1, 2.1, 2.1) otherwise. The default for a RGB raster is 0. Use `mar=NA` to not set the margins
`legend`	logical or character. If not `FALSE` a legend is drawn. The character value can be used to indicate where the legend is to be drawn. For example "topright" or "bottomleft". Use `plg` for more refined placement. Not supported for continuous legends (the default for raster data)
`axes`	logical. Draw axes?
`buffer`	logical. If `TRUE` the plotting area is made slightly larger than the extent of `x`
`background`	background color. Default is no color (white)
`box`	logical. Should a box be drawn around the map?
`clip`	logical. Should the axes be clipped to the extent of `x`?

`plg`	list with parameters for drawing the legend. For the classes and interval type legend see the arguments for `legend`. For example `x` and `y` can be used to place the legend. You can also use keywords such as "topleft" and "bottomright" to place the legend at these locations inside the map rectangle. Some of these do not apply to a continuous legend, or they behave a little differently. For example, only the placement keywords "left", "right", "top", and "bottom" are recognized; and when using these keywords, the legend is placed outside of the map rectangle. Additional parameters for continuous legends include: • `digits` to set the number of digits to print after the decimal point. `size` to change the height and/or width; the defaults are `c(1,1)`, negative values for size flip the order of the legend. • `at` to set the location of the tic-marks • `tic` One of these partially matched values: "through", "in", "out", or "none", to choose a tic-mark placement/length that is different from the default "through and out".
`pax`	list with parameters for drawing axes. See the arguments for `axis`. Arguments `side`, `tick` and `lab` can be used to indicate for which of the four axes to draw a line (side), tick-mark, and/or the tick-mark labels. The default is `c(1:4)` for side and `1:2` for the other two. If `side` is changed the other two default to that value. Logical argument `retro` can be used to use a sexagesimal notation for the labels (degrees/minutes/hemisphere) instead of the standard decimal notation
`maxcell`	positive integer. Maximum number of cells to use for the plot
`smooth`	logical. If `TRUE` the cell values are smoothed (only if a continuous legend is used)
`range`	numeric. minimum and maximum values to be used for the continuous legend
`fill_range`	logical. If `TRUE`, values outside of `range` get the colors of the extreme values; otherwise they get colored as `NA`
`levels`	character. labels for the legend when `type="classes"`
`all_levels`	logical. If `TRUE`, the legend shows all levels of a categorical raster, even if they are not present in the data
`breaks`	numeric. Either a single number to indicate the number of breaks desired, or the actual breaks. When providing this argument, the default legend becomes "interval"
`breakby`	character or function. Either "eqint" for equal interval breaks, "cases" for equal quantile breaks. If a function is supplied, it should take a single argument (a vector of values) and create groups
`fun`	function to be called after plotting each SpatRaster layer to add something to each map (such as text, legend, lines). For example, with SpatVector `v`, you could do `fun=function() lines(v)`. The function may have one argument, representing the layer that is plotted (1 to the number of layers)
`colNA`	character. color for the NA values
`alpha`	Either a single numeric between 0 and 1 to set the transparency for all colors (0 is transparent, 1 is opaque) or a SpatRaster with values between 0 and 1 to set

	the transparency by cell. To set the transparency for a given color, set it to the colors directly
sort	logical. If TRUE legends with categorical values are sorted. If x is a SpatVector you can also supply a vector of the unique values, in the order in which you want them to appear in the legend
decreasing	logical. If TRUE, legends are sorted in decreasing order
grid	logical. If TRUE grid lines are drawn. Their properties such as type and color can be set with the pax argument
nc	positive integer. Optional. The number of columns to divide the plotting device in (when plotting multiple layers)
nr	positive integer. Optional. The number of rows to divide the plotting device in (when plotting multiple layers)
main	character. Main plot titles (one for each layer to be plotted). You can use arguments cex.main, font.main, col.main to change the appearance; and loc.main to change the location of the main title (either two coordinates, or a character value such as "topleft")
maxnl	positive integer. Maximum number of layers to plot (for a multi-layer object)
add	logical. If TRUE add the object to the current plot
ext	SpatExtent. Can be use instead of xlim and ylim to set the extent of the plot
reset	logical. If TRUE add the margins (see argument mar) are reset to what they were before calling plot; doing so may affect the display of additional objects that are added to the map (e.g. with lines)
values	Either a vector with values to be used for plotting or a two-column data.frame, where the first column matches a variable in x and the second column has the values to be plotted
...	arguments passed to plot("SpatRaster", "numeric") and additional graphical arguments

See Also

points, lines, polys, image

Add map elements: text, sbar, north, add_legend, add_box

plot a SpatGraticule or SpatExtent,

multiple layers: plotRGB, panel

other plot types: scatterplot, hist, pairs, density, persp, contour, boxplot, barplot

Examples

```
## SpatRaster
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
plot(r)

plot(r, type="interval")
```

```
e <- c(6.37, 6.41, 49.9, 50.1)
plot(r, plg=list(ext=e, title="Legend\nTitle", title.cex=0.9),
pax=list(side=1:4, retro=FALSE))
north(cbind(5.8, 50.1))

d <- classify(r, c(100,200,300,400,500,600))
plot(d, type="classes")

plot(d, type="interval", breaks=1:5)
plot(d, type="interval", breaks=c(1,4,5), plg=list(legend=c("1-4", "4-5")))
plot(d, type="classes", xlim=c(5.6, 6.6),
plg=list(legend=c("Mr", "Xx", "As", "Zx", "Bb"), x="bottomleft"))


x <- trunc(r/200)
levels(x) <- data.frame(id=0:2, element=c("earth", "wind", "fire"))
plot(x, plg=list(x="topright"),mar=c(2,2,2,2))

oldpar <- par(no.readonly=TRUE)

# two plots with the same legend
dev.new(width=6, height=4, noRStudioGD = TRUE)
par(mfrow=c(1,2))
plot(r, range=c(50,600), mar=c(1,1,1,4))
plot(r/2, range=c(50,600), mar=c(1,1,1,4))

# as we only need one legend:
par(mfrow=c(1,2))
plot(r, range=c(50,600), mar=c(2, 2, 2, 2), plg=list(size=0.9, cex=.8),
pax=list(side=1:2, cex.axis=.6), box=FALSE)
#text(182500, 335000, "Two maps, one plot", xpd=NA)
plot(r/2, range=c(50,600), mar=c(2, 2, 2, 2), legend=FALSE,
pax=list(side=c(1,4), cex.axis=.6), box=FALSE)

par(oldpar)

# multi-layer with RGB
s <- rast(system.file("ex/logo.tif", package="terra"))
s
plot(s)
# remove RGB
plot(s*1)
# or use layers
plot(s, 1)
plot(s, 1:3)

# fix legend by linking values and colors

x = rast(nrows = 2, ncols = 2, vals=1)
y = rast(nrows = 2, ncols = 2, vals=c(1,2,2,1))
cols = data.frame(id=1:2, col=c("red", "blue"))
plot(c(x,y), col=cols)
```

```
r  = rast(nrows=10, ncols=10, vals=1:100)
dr = data.frame(from=c(5,33,66,150), to=c(33, 66, 95,200), col=rainbow(4))
plot(r, col=dr)

### SpatVector

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

plot(v)

plot(v, "NAME_2", col=rainbow(12), border=c("gray", "blue"), lwd=3)

plot(v, 2, pax=list(side=1:2), plg=list(x=6.16, y=50.17, cex=.8), xlim=c(5.7, 6.7))

plot(v, 4, pax=list(side=1:2), plg=list(x=6.2, y=50.2, ncol=2), main="", box=FALSE)

plot(v, 1, plg=list(x=5.8, y=49.37, horiz=TRUE, cex=1.1), main="", mar=c(5,2,0.5,0.5))

plot(v, density=1:12, angle=seq(18, 360, 20), col=rainbow(12))


plot(v, "AREA", type="interval", breaks=3, mar=c(3.1, 3.1, 2.1, 3.1),
  plg=list(x="topright"), main="")

plot(v, "AREA", type="interval", breaks=c(0,200,250,350),
mar=c(2,2,2,2), xlim=c(5.7, 6.75),
plg=list(legend=c("<200", "200-250", ">250"), cex=1, bty="o",
x=6.3, y=50.15, box.lwd=2, bg="light yellow", title="My legend"))
```

`plotRGB` *Red-Green-Blue plot of a multi-layered SpatRaster*

Description

Make a Red-Green-Blue plot based on three layers in a SpatRaster. The layers (sometimes referred to as "bands" because they may represent different bandwidths in the electromagnetic spectrum) are combined such that they represent the red, green and blue channel. This function can be used to make "true" (or "false") color images from Landsat and other multi-spectral satellite images.

Note that the margins of the plot are set to zero (no axes or titles are visible) but can be set with the `mar` argument.

An alternative way to plot RGB images is to first use `colorize` to create a single layer SpatRaster with a color-table and then use `plot`.

Usage

```
## S4 method for signature 'SpatRaster'
plotRGB(x, r=1, g=2, b=3, a=NULL, scale=NULL, mar=0,
stretch=NULL, smooth=TRUE, colNA="white", alpha=NULL, bgalpha=NULL,
zlim=NULL, zcol=FALSE, axes=FALSE ,...)
```

Arguments

`x`	SpatRaster
`r`	integer between 1 and `nlyr(x)`. Layer to use as the Red channel
`g`	integer between 1 and `nlyr(x)`. Layer to use as the Green channel
`b`	integer between 1 and `nlyr(x)`. Layer to use as the Blue channel
`a`	NULL or integer between 1 and `nlyr(x)`. Layer to use as the alpha (transparency) channel. If not NULL, argument `alpha` is ignored
`scale`	integer. Maximum (possible) value in the three channels. Defaults to 255 or to the maximum value of `x` if that is known and larger than 255
`mar`	numeric vector recycled to length 4 to set the margins of the plot. Use `mar=NULL` or `mar=NA` to not set the margins
`stretch`	character. Option to stretch the values to increase contrast: "lin" (linear) or "hist" (histogram). The linear stretch uses `stretch` with arguments `minq=0.02` and `maxq=0.98`
`smooth`	logical. If `TRUE`, smooth the image when drawing to get the appearance of a higher spatial resolution
`colNA`	color. The color used for cells that have NA values
`alpha`	transparency. Integer between 0 (transparent) and 255 (opaque)
`bgalpha`	Background transparency. Integer between 0 (transparent) and 255 (opaque)
`zlim`	numeric vector of length 2. Range of values to plot (optional). If this is set, and `stretch="lin"` is used, then the values are stretched within the range of `zlim`. This allows creating consistent coloring between SpatRasters with different cell-value ranges, even when stretching the colors for improved contrast
`zcol`	logical. If `TRUE` the values outside the range of zlim get the color of the extremes of the range. Otherwise, the values outside the zlim range get the color of `NA` values (see argument "colNA")
`axes`	logical. If `TRUE` axes are drawn (and arguments such as `main="title"` will be honored)
`...`	graphical parameters as in `plot`<SpatRaster-method>

See Also

`plot`, `colorize`, `RGB`

Examples

```
b <- rast(system.file("ex/logo.tif", package="terra"))
plotRGB(b)
plotRGB(b, mar=2)
plotRGB(b, 3, 2, 1)

b[1000:2000] <- NA
plotRGB(b, 3, 2, 1, stretch="hist")
```

---

plot_extent *Plot a SpatExtent*

Description

Plot a SpatExtent. Use `lines` to add a SpatExtent to an existing map.

See `plot` for plotting other object types.

Usage

```
## S4 method for signature 'SpatExtent,missing'
plot(x, y, ...)
```

Arguments

x	SpatExtent
y	missing
...	additional graphical arguments for lines

See Also

`plot`

Examples

```
r <- rast()
plot(ext(r))
```

plot_graticule *Plot a graticule*

Description

Plot a SpatGraticule. You can create a SpatGraticule with graticule.

Usage

```
## S4 method for signature 'SpatGraticule,missing'
plot(x, y, background=NULL, col="black", mar=NULL, labels=TRUE,
retro=FALSE, lab.loc=c(1,1), lab.lon=NULL, lab.lat=NULL, lab.cex=0.65,
lab.col="black", off.lat=0.25, off.lon=0.25, box=FALSE, box.col="black",
add=FALSE, ...)
```

Arguments

x	SpatRaster or SpatVector
y	missing or positive integer or name indicating the layer(s) to be plotted
background	background color. If NULL, no background is drawn
mar	numeric vector of length 4 to set the margins of the plot. To make space for the legend you may use something like `c(3.1, 3.1, 2.1, 7.1)`. To fill the plotting canvas, you can use `c(0,0,0,0`. Use `NA` to not set the margins
col	character. Color for the graticule lines
labels	logical. If `TRUE`, show graticule labels
retro	logical. If `TRUE`, show "retro" instead of decimal labels with the graticule
lab.loc	numeric. The first number indicates where the longitude graticule labels should be drawn (1=bottom, 2=top, NA=not drawn, any other number=top and bottom). The second number indicates where the latitude graticule labels should be drawn (1=left, 2=right, NA=not drawn, any other number=left and right)
lab.lon	positive integers between 1 and the number of labels, indicating which longitude graticule labels should be included
lab.lat	positive integers between 1 and the number of labels, indicating which latitude graticule labels should be included
lab.cex	double. size of the label font
lab.col	character. color of the labels
off.lon	numeric. longitude labels offset
off.lat	numeric. latitude labels offset
box	logical. If `TRUE`, the outer lines of the graticule are drawn on top with a sold line `lty=1`
box.col	character. color of the outer lines of the graticule if `box=TRUE`
add	logical. Add the graticule to the current plot?
...	additional graphical arguments passed to lines

See Also

graticule, plot, points, lines, polys, image, scatterplot, scale bar: sbar, north arrow: north

Examples

```
g <- graticule(60, 30, crs="+proj=robin")

plot(g, background="azure", col="red", lty=2, box=TRUE)
plot(g, background="azure", col="light gray", lab.loc=c(1,2),
lab.lon=c(2,4,6), lab.lat=3:5, lty=3, retro=TRUE)
```

---

prcomp *SpatRaster PCA with prcomp*

Description

Compute principal components for SpatRaster layers. This method may be preferred to princomp for its greater numerical accuracy. However, it is slower and for very large rasters it can only be done with a sample. This may be good enough but see princomp if you want to use all values. Unlike princomp, in this method the sample variances are used with n-1 as the denominator.

Usage

```
## S4 method for signature 'SpatRaster'
prcomp(x, retx=TRUE, center=TRUE, scale.=FALSE,
tol=NULL, rank.=NULL, maxcell=Inf)
```

Arguments

x	SpatRaster
retx	a logical value indicating whether the rotated variables should be returned
center	a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale
scale.	a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of x can be supplied. The value is passed to scale
tol	a value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to tol times the standard deviation of the first component.) With the default null setting, no components are omitted (unless rank. is specified less than min(dim(x))). Other settings for tol could be tol = 0 or tol = sqrt(.Machine$double.eps), which would omit essentially constant components

`rank.` optionally, a number specifying the maximal rank, i.e., maximal number of principal components to be used. Can be set as alternative or in addition to tol, useful notably when the desired rank is considerably smaller than the dimensions of the matrix

`maxcell` positive integer. The maximum number of cells to be used. If this is smaller than ncell(x), a regular sample of x is used

Value

prcomp object

Note

`prcomp` may change the layer names if they are not valid. See `make.names`. In that case, you will get a warning, and would need to also make the layer names of `x` valid before using `predict`. Even better would be to change them before calling `prcomp`.

See Also

`princomp`, `prcomp`

Examples

```
f <- system.file("ex/logo.tif", package = "terra")
r <- rast(f)
pca <- prcomp(r)
x <- predict(r, pca)

# use "index" to get a subset of the components
p <- predict(r, pca, index=1:2)
```

`predict` *Spatial model predictions*

Description

Make a SpatRaster with predictions from a fitted model object (for example, obtained with `glm` or `randomForest`). The first argument is a SpatRaster object with the predictor variables. The `names` in the SpatRaster should exactly match those expected by the model. Any regression like model for which a predict method has been implemented (or can be implemented) can be used.

The method should work if the model's predict function returns a vector, matrix or data.frame (or a list that can be coerced to a data.frame). In other cases it may be necessary to provide a custom "predict" function that wraps the model's predict function to return the values in the required form. See the examples.

This approach of using model predictions is commonly used in remote sensing (for the classification of satellite images) and in ecology, for species distribution modeling.

Usage

```
## S4 method for signature 'SpatRaster'
predict(object, model, fun=predict, ..., const=NULL, na.rm=FALSE,
      index=NULL, cores=1, cpkgs=NULL, filename="", overwrite=FALSE, wopt=list())
```

Arguments

object	SpatRaster
model	fitted model of any class that has a "predict" method (or for which you can supply a similar method as fun argument. E.g. glm, gam, or randomForest
fun	function. The predict function that takes model as first argument. The default value is predict, but can be replaced with e.g. predict.se (depending on the type of model), or your own custom function
...	additional arguments for fun
const	data.frame. Can be used to add a constant value as a predictor variable so that you do not need to make a SpatRaster layer for it
na.rm	logical. If TRUE, cells with NA values in the any of the layers of x are removed from the computation (even if the NA cell is in a layer that is not used as a variable in the model). This option prevents errors with models that cannot handle NA values when makeing predictions. In most other cases this will not affect the output. However, there are some models that return predicted values even if some (or all) variables are NA
index	integer or character. Can be used to to select a subset of the model output variables
cores	positive integer. If cores > 1, a 'parallel' package cluster with that many cores is created and used
cpkgs	character. The package(s) that need to be loaded on the nodes to be able to run the model.predict function (see examples)
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

interpolate for spatial model prediction

Examples

```
logo <- rast(system.file("ex/logo.tif", package="terra"))
names(logo) <- c("red", "green", "blue")
p <- matrix(c(48, 48, 48, 53, 50, 46, 54, 70, 84, 85, 74, 84, 95, 85,
   66, 42, 26, 4, 19, 17, 7, 14, 26, 29, 39, 45, 51, 56, 46, 38, 31,
```

```
   22, 34, 60, 70, 73, 63, 46, 43, 28), ncol=2)

a <- matrix(c(22, 33, 64, 85, 92, 94, 59, 27, 30, 64, 60, 33, 31, 9,
   99, 67, 15, 5, 4, 30, 8, 37, 42, 27, 19, 69, 60, 73, 3, 5, 21,
   37, 52, 70, 74, 9, 13, 4, 17, 47), ncol=2)

xy <- rbind(cbind(1, p), cbind(0, a))

# extract predictor values for points
e <- extract(logo, xy[,2:3])

# combine with response (excluding the ID column)
v <- data.frame(cbind(pa=xy[,1], e))

#build a model, here with glm
model <- glm(formula=pa~., data=v)

#predict to a raster
r1 <- predict(logo, model)

plot(r1)
points(p, bg='blue', pch=21)
points(a, bg='red', pch=21)

# logistic regression
model <- glm(formula=pa~., data=v, family="binomial")
r1log <- predict(logo, model, type="response")

# to get the probability and standard error
r1se <- predict(logo, model, se.fit=TRUE)

# or provide a custom predict function

predfun <- function(model, data) {
  v <- predict(model, data, se.fit=TRUE)
  cbind(p=as.vector(v$fit), se=as.vector(v$se.fit))
}

r2 <- predict(logo, model, fun=predfun)

### principal components of a SpatRaster
pca <- prcomp(logo)

# or use sampling if you have a large raster
# and cannot process all cell values
sr <- spatSample(logo, 100000, "regular")
pca <- prcomp(sr)

x <- predict(logo, pca)
plot(x)

## parallelization
## Not run:
```

```
## simple case with GLM
model <- glm(formula=pa~., data=v)
p <- predict(logo, model, cores=2)

## The above does not work with a model from a contributed
## package, as the package needs to be loaded in each core.
## Below are three approaches to deal with that

library(randomForest)
rfm <- randomForest(formula=pa~., data=v)

## approach 0 (not parallel)
rp0 <- predict(logo, rfm)

## approach 1, use the "cpkgs" argument
rp1 <- predict(logo, rfm, cores=2, cpkgs="randomForest")

## approach 2, write a custom predict function that loads the package
rfun <- function(mod, dat, ...) {
library(randomForest)
predict(mod, dat, ...)
}
rp2 <- predict(logo, rfm, fun=rfun, cores=2)

## approach 3, write a parallelized custom predict function
rfun <- function(mod, dat, ...) {
ncls <- length(cls)
nr <- nrow(dat)
s <- split(dat, rep(1:ncls, each=ceiling(nr/ncls), length.out=nr))
unlist(  parallel::clusterApply(cls, s, function(x, ...) predict(mod, x, ...))  )
}

library(parallel)
cls <- parallel::makeCluster(2)
parallel::clusterExport(cls, c("rfm", "rfun", "randomForest"))
rp3 <- predict(logo, rfm, fun=rfun)
parallel::stopCluster(cls)

plot(c(rp0, rp1, rp2, rp3))


### with two output variables (probabilities for each class)
v$pa <- as.factor(v$pa)
rfm2 <- randomForest(formula=pa~., data=v)
rfp <- predict(logo, rfm2, cores=2, type="prob", cpkgs="randomForest")

## End(Not run)
```

---

princomp *SpatRaster PCA with princomp*

Description

Compute principal components for SpatRaster layers. This method can use all values to compute the principal components, even for very large rasters. This is because it computes the covariance matrix by processing the data in chunks, if necessary, using layerCor. The population covariance is used (not the sample, with n-1 denominator, covariance).

Alternatively, you can specify maxcell or sample raster values to a data.frame to speed up calculations for very large rasters (see the examples below).

See prcomp for an alternative method that has higher numerical accuracy, but is slower, and for very large rasters can only be accomplished with a sample since all values must be read into memory.

Usage

```
## S4 method for signature 'SpatRaster'
princomp(x, cor=FALSE, fix_sign=TRUE, use="pairwise.complete.obs", maxcell=Inf)
```

Arguments

x	SpatRaster
cor	logical. If FALSE, the covariance matrix is used. Otherwise the correlation matrix is used
fix_sign	logical. If TRUE, the signs of the loadings and scores are chosen so that the first element of each loading is non-negative
use	character. To decide how to handle missing values. This must be (an abbreviation of) one of the strings "everything", "complete.obs", "pairwise.complete.obs", or "masked.complete". With "pairwise.complete.obs", the covariance between a pair of layers is computed for all cells that are not NA in that pair. Therefore, it may be that the (number of) cells used varies between pairs. The benefit of this approach is that all available data is used. Use "complete.obs", if you want to only use the values from cells that are not NA in any of the layers. By using "masked.complete" you indicate that all layers have NA values in the same cells
maxcell	positive integer. The maximum number of cells to be used. If this is smaller than ncell(x), a regular sample of x is used

Value

princomp object

Author(s)

Alex Ilich and Robert Hijmans, based on a similar method by Benjamin Leutner

See Also

prcomp princomp

Examples

```
f <- system.file("ex/logo.tif", package = "terra")
r <- rast(f)
pca <- princomp(r)
x <- predict(r, pca)

# use "index" to get a subset of the components
p <- predict(r, pca, index=1:2)

### use princomp directly
pca2 <- princomp(values(r),  fix_sign = TRUE)
p2 <- predict(r, pca2)

### may need to use sampling with a large raster
### here with prcomp instead of princomp
sr <- spatSample(r, 100000, "regular")
pca3 <- prcomp(sr)
p3 <- predict(r, pca3)
```

---

project *Change the coordinate reference system*

Description

Change the coordinate reference system ("project") of a SpatVector, SpatRaster or a matrix with coordinates.

Usage

```
## S4 method for signature 'SpatVector'
project(x, y, partial = FALSE)

## S4 method for signature 'SpatRaster'
project(x, y, method, mask=FALSE, align_only=FALSE, res=NULL,
origin=NULL, threads=FALSE, filename="", ..., use_gdal=TRUE, by_util = FALSE)

## S4 method for signature 'SpatExtent'
project(x, from, to)

## S4 method for signature 'matrix'
project(x, from, to)
```

Arguments

x	SpatRaster, SpatVector, SpatExtent or matrix (with x and y columns) whose coordinates to project

`y`	if `x` is a SpatRaster, the preferred approach is for `y` to be a SpatRaster as well, serving as a template for the geometry (extent and resolution) of the output SpatRaster. Alternatively, you can provide a coordinate reference system (CRS) description. You can use the following formats to define coordinate reference systems: WKT, PROJ.4 (e.g., `+proj=longlat +datum=WGS84`), or an EPSG code (e.g., `"epsg:4326"`). But note that the PROJ.4 notation has been deprecated, and you can only use it with the WGS84/NAD83 and NAD27 datums. Other datums are silently ignored. If `x` is a SpatVector, you can provide a crs definition as discussed above, or any other object from which such a crs can be extracted with `crs`
`partial`	logical. If `TRUE`, geometries that can only partially be represented in the output crs are included in the output
`method`	character. Method used for estimating the new cell values of a SpatRaster. One of: `near`: nearest neighbor. This method is fast, and it can be the preferred method if the cell values represent classes. It is not a good choice for continuous values. This is used by default if the first layer of `x` is categorical. `bilinear`: bilinear interpolation. This is the default if the first layer of `x` is numeric (not categorical). `cubic`: cubic interpolation. `cubicspline`: cubic spline interpolation. `lanczos`: Lanczos windowed sinc resampling. `sum`: the weighted sum of all non-NA contributing grid cells. `min, q1, med, q3, max, average, mode, rms`: the minimum, first quartile, median, third quartile, maximum, mean, mode, or root-mean-square value of all non-NA contributing grid cells.
`mask`	logical. If `TRUE`, mask out areas outside the input extent. For example, to avoid data wrapping around the date-line (see example with Robinson projection). To remove cells that are `NA` in `y` (if `y` is a SpatRaster) you can use the `mask` `method` after calling `project` (this function)
`align_only`	logical. If `TRUE`, and `y` is a SpatRaster, the template is used for the spatial resolution and origin, but the extent is set such that all of the extent of `x` is included
`res`	numeric. Can be used to set the resolution of the output raster if `y` is a CRS
`origin`	numeric. Can be used to set the origin of the output raster if `y` is a CRS
`threads`	logical. If `TRUE` multiple threads are used (faster for large files)
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`
`use_gdal`	logical. If `TRUE` the GDAL-warp algorithm is used. Otherwise, a slower internal algorithm is used that may be more accurate if there is much variation in the cell sizes of the output raster. Only the `near` and `bilinear` algorithms are available for the internal algorithm
`by_util`	logical. If `TRUE` and `gdal=TRUE`, the GDAL warp utility is used
`from`	character. Coordinate reference system of `x`
`to`	character. Output coordinate reference system

Value

SpatVector or SpatRaster

Note

The PROJ.4 notation of coordinate reference systems has been partly deprecated in the GDAL/PROJ library that is used by this function. You can still use this notation, but *only* with the WGS84 datum. Other datums are silently ignored.

Transforming (projecting) raster data is fundamentally different from transforming vector data. Vector data can be transformed and back-transformed without loss in precision and without changes in the values. This is not the case with raster data. In each transformation the values for the new cells are estimated in some fashion. Therefore, if you need to match raster and vector data for analysis, you should generally transform the vector data.

When using this method with a `SpatRaster`, the preferable approach is to provide a template `SpatRaster` as argument `y`. The template is then another raster dataset that you want your data to align with. If you do not have a template to begin with, you can do `project(rast(x), crs)` and then manipulate the output to get the template you want. For example, where possible use whole numbers for the extent and resolution so that you do not have to worry about small differences in the future. You can use commands like `dim(z) = c(180, 360)` or `res(z) <- 100000`.

The output resolution should generally be similar to the input resolution, but there is no "correct" resolution in raster transformation. It is not obvious what this resolution is if you are using lon/lat data that spans a large North-South extent.

See Also

`crs`, `resample`

Examples

```
## SpatRaster
a <- rast(ncols=40, nrows=40, xmin=-110, xmax=-90, ymin=40, ymax=60,
          crs="+proj=longlat +datum=WGS84")
values(a) <- 1:ncell(a)
newcrs="+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"
b <- rast(ncols=94, nrows=124, xmin=-944881, xmax=935118, ymin=4664377, ymax=7144377, crs=newcrs)
w <- project(a, b)


## SpatVector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
crs(v, proj=TRUE)
cat(crs(v), "\n")

project(v, "+proj=moll")


project(v, "EPSG:2169")
```

quantile *Quantiles of spatial data*

Description

Compute quantiles for each cell across the layers of a SpatRaster.

You can use use `global(x, fun=quantile)` to instead compute quantiles across cells for each layer.

You can also use this method to compute quantiles of the numeric variables of a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster'
quantile(x, probs=seq(0, 1, 0.25), na.rm=FALSE, filename="", ...)

## S4 method for signature 'SpatVector'
quantile(x, probs=seq(0, 1, 0.25), ...)
```

Arguments

`x`	SpatRaster or SpatVector
`probs`	numeric vector of probabilities with values in [0,1]
`na.rm`	logical. If `TRUE`, `NA`'s are removed from `x` before the quantiles are computed
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster with layers representing quantiles

See Also

`app`

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
rr <- c(r/2, r, r*2)
qr <- quantile(rr)
qr

## Not run:
# same but slower
qa <- app(rr, quantile)

## End(Not run)
```

```
#quantile by layer instead of by cell
qg <- global(r, quantile)
```

---

query *Query a SpatVectorProxy object*

Description

Query a SpatVectorProxy to extract a subset

Usage

```
## S4 method for signature 'SpatVectorProxy'
query(x, start=1, n=nrow(x), vars=NULL, where=NULL,
    extent=NULL, filter=NULL, sql=NULL, what="")
```

Arguments

`x`	SpatVectorProxy
`start`	positive integer. The record to start reading at
`n`	positive integer. The number of records requested
`vars`	character. Variable names. Must be a subset of `names(x)`
`where`	character. expression like "NAME_1='California' AND ID > 3" , to subset records. Note that start and n are applied after executing the where statement
`extent`	Spat* object. The extent of the object is used as a spatial filter to select the geometries to read. Ignored if `filter` is not `NULL`
`filter`	SpatVector. Used as a spatial filter to select geometries to read (the convex hull is used for lines or points)
`sql`	character. Arbitrary SQL statement. If used, arguments "start", "n", "vars" and "where" are ignored
`what`	character indicating what to read. Either `""` for geometries and attributes, or `"geoms"` to only read the geometries, `"attributes"` to only read the attributes (that are returned as a data.frame)

Value

SpatVector

See Also

`vect`

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f, proxy=TRUE)
v

x <- query(v, vars=c("ID_2", "NAME_2"), start=5, n=2)
x

query(v, vars=c("ID_2", "NAME_1", "NAME_2"), where="NAME_1='Grevenmacher' AND ID_2 > 6")

## with an extent
e <- ext(5.9, 6.3, 49.9, 50)
x <- query(v, extent=e)

## with polygons
p <- as.polygons(e)
x <- query(v, filter=p)
x
```

`rangeFill` *Fill layers with a range*

Description

Fill layers with cell-varying ranges defined by a start and end SpatRaster. The range must start at 1 and end at a user-defined maximum. Output values are either zero (not in the range) or one (in the range).

For example, for a cell with `start=3`, `end=5` and with `limit=8`, the output for that cell would be `0,0,1,1,1,0,0,0`

Usage

```
## S4 method for signature 'SpatRaster'
rangeFill(x, limit, circular=FALSE, filename="", ...)
```

Arguments

`x`	SpatRaster with at two layers. The cell values of the first layer indicate the start of the range (1 based); the cell values are indicate the end of the range
`limit`	numeric > 1. The range size
`circular`	logical. If `TRUE` the values are considered circular, such as the days of the year. In that case, if first > last the layers used are c(first:limit, 1:last). Otherwise, if `circular=FALSE`, such a range would be considered invalid and `NA` would be used
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

rapp

Examples

```
x <- y <- rast(ncol=2, nrow=2)
values(x) <- c(NA, 1:3)
values(y) <- c(NA, 4:6)

r <- rangeFill(c(x, y), 8)
```

---

rapp *Range-apply*

Description

Apply a function to a range of the layers of a SpatRaster that varies by cell. The range is specified for each cell with one or two SpatRasters (arguments `first` and `last`). For either `first` or `last` you can use a single number instead.

You cannot use single numbers for both `first` and `last` because in that case you could use `app` or `Summary-methods`, perhaps `subset`ting the layers of a SpatRaster.

See `selectRange` to create a new SpatRaster by extracting one or more values starting at a cell-varying layer.

Usage

```
## S4 method for signature 'SpatRaster'
rapp(x, first, last, fun, ..., allyrs=FALSE, fill=NA,
        clamp=FALSE, circular=FALSE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`first`	SpatRaster or positive integer between 1 and nlyr(x), indicating the first layer in the range of layers to be considered
`last`	SpatRaster or positive integer between 1 and nlyr(x), indicating the last layer in the range to be considered
`fun`	function to be applied
`...`	additional arguments passed to `fun`
`allyrs`	logical. If `TRUE`, values for all layers are passed to `fun` but the values outside of the range are set to `fill`

fill	numeric. The fill value for the values outside of the range, for when allyrs=TRUE
clamp	logical. If FALSE and the specified range is outside 1:nlyr(x) all cells are considered NA. Otherwise, the invalid part of the range is ignored
circular	logical. If TRUE the values are considered circular, such as the days of the year. In that case, if first > last the layers used are c(first:nlyr(x), 1:last). Otherwise, the range would be considered invalid and NA would be returned
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

selectRange, app, Summary-methods, lapp, tapp

Examples

```
r <- rast(ncols=9, nrows=9)
values(r) <- 1:ncell(r)
s <- c(r, r, r, r, r, r)
s <- s * 1:6
s[1:2] <- NA
start <- end <- rast(r)
start[] <- 1:3
end[]   <- 4:6
a <- rapp(s, start, end, fun="mean")
b <- rapp(s, start, 2, fun="mean")

# cumsum from start to nlyr(x). return all layers
r <- rapp(s, start, nlyr(s), cumsum, allyrs=TRUE, fill=0)
# return only the final value
rr <- rapp(s, start, nlyr(s), function(i) max(cumsum(i)))
```

---

rast   *Create a SpatRaster*

Description

Methods to create a SpatRaster. These objects can be created from scratch, from a filename, or from another object.

A SpatRaster represents a spatially referenced surface divided into three dimensional cells (rows, columns, and layers).

When a SpatRaster is created from one or more files, it does not load the cell (pixel) values into memory (RAM). It only reads the parameters that describe the geometry of the SpatRaster, such as the number of rows and columns and the coordinate reference system. The actual values will be read when needed.

Note that there are operating system level limitations to the number of files that can be opened simulatenously. Using a SpatRaster of very many files (e.g. 10,000) may cause R to crash when you use it in a computation. In situations like that you may need to split up the task or combine data into fewer (multi-layer) files. Also note that the GTiff format used for temporary files cannot store more than 65535 layers in a single file.

Usage

```
## S4 method for signature 'character'
rast(x, subds=0, lyrs=NULL, drivers=NULL, opts=NULL,
  win=NULL, snap="near", vsi=FALSE, raw=FALSE)

## S4 method for signature 'missing'
rast(x, nrows=180, ncols=360, nlyrs=1, xmin=-180, xmax=180,
          ymin=-90, ymax=90, crs, extent, resolution, vals, names, time, units)

## S4 method for signature 'SpatRaster'
rast(x, nlyrs=nlyr(x), names, vals, keeptime=TRUE,
  keepunits=FALSE, props=FALSE, tags=FALSE)

## S4 method for signature 'matrix'
rast(x, type="", crs="", digits=6, extent=NULL)

## S4 method for signature 'data.frame'
rast(x, type="xyz", crs="", digits=6, extent=NULL)

## S4 method for signature 'array'
rast(x, crs="", extent=NULL)

## S4 method for signature 'list'
rast(x, warn=TRUE)

## S4 method for signature 'SpatRasterDataset'
rast(x)

## S4 method for signature 'SpatVector'
rast(x, ...)

## S4 method for signature 'SpatExtent'
rast(x, ...)
```

Arguments

x	filename (character), missing, SpatRaster, SpatRasterDataset, SpatExtent, SpatVector, matrix, array, list of SpatRasters. For other types it will be attempted to create a SpatRaster via ('as(x, "SpatRaster")'

`subds`	positive integer or character to select a sub-dataset. If zero or "", all sub-datasets are returned (if possible)
`lyrs`	positive integer or character to select a subset of layers (a.k.a. "bands")
`drivers`	character. GDAL drivers to consider
`opts`	character. GDAL dataset open options
`win`	SpatExtent to set a `window` (area of interest)
`snap`	character. One of "near", "in", or "out", to indicate how the extent of `window` should be "snapped" to `x`
`vsi`	logical. If `TRUE`, "\vsicurl\" is prepended to filenames that start with "http"
`raw`	logical. If `TRUE`, scale and offset values are ignored. See `scoff` to get these parameters
`nrows`	positive integer. Number of rows
`ncols`	positive integer. Number of columns
`nlyrs`	positive integer. Number of layers
`xmin`	minimum x coordinate (left border)
`xmax`	maximum x coordinate (right border)
`ymin`	minimum y coordinate (bottom border)
`ymax`	maximum y coordinate (top border)
`crs`	character. Description of the Coordinate Reference System (map projection) in `PROJ.4`, `WKT` or `authority:code` notation. See `crs`. If this argument is missing, and the x coordinates are within -360 .. 360 and the y coordinates are within -90 .. 90, longitude/latitude is assigned
`keeptime`	logical. If `FALSE` the time stamps are discarded
`keepunits`	logical. If `FALSE` the layer units are discarded
`props`	logical. If `TRUE` the properties (categories and color-table) are kept
`tags`	logical. If `TRUE` the user specified metadata tags are kept (see `metags`).
`extent`	object of class SpatExtent. If present, the arguments xmin, xmax, ymin and ymax are ignored
`resolution`	numeric vector of length 1 or 2 to set the spatial resolution (see `res`). If this argument is used, arguments `ncols` and `nrows` are ignored
`vals`	numeric. An optional vector with cell values (if fewer values are provided, these are recycled to reach the number of cells)
`names`	character. An optional vector with layer names (must match the number of layers)
`time`	time or date stamps for each layer
`units`	character. units for each layer
`type`	character. If the value is `"xyz"`, the matrix or data.frame `x` must have at least two columns, the first with `x` (or longitude) and the second with `y` (or latitude) coordinates that represent the centers of raster cells. The additional columns are the values associated with the raster cells. If the value is `"xylz"`, `x` must have four columns with the third representing the layer and the fourth the cell values. If the value is `""`, the resulting SpatRaster will have the same number of rows and columns as `x`.

`digits`	integer to set the precision for detecting whether points are on a regular grid (a low number of digits is a low precision). Only used when `type="xyz"`
`warn`	logical. If `TRUE`, a warnings about empty rasters may be emitted
`...`	additional arguments passed on to the `rast,missing-method`

Details

Files are read with the GDAL library. GDAL guesses the file format from the name, and/or tries reading it with different "drivers" (see `gdal`) until it succeeds. In very few cases this may cause a file to be opened with the wrong driver, and some information may be lost. For example, when a netCDF file is opened with the HDF5 driver. You can avoid that by using argument `rast("filename.ncdf", drivers="NETCDF")`

These classes hold a C++ pointer to the data "reference class" and that creates some limitations. They cannot be recovered from a saved R session either or directly passed to nodes on a computer cluster. Generally, you should use `writeRaster` to save SpatRaster objects to disk (and pass a filename or cell values of cluster nodes). Also see `wrap`.

Value

SpatRaster

See Also

`sds` to create a SpatRasterDataset (4 dimensions) and `vect` for vector (points, lines, polygons) data

Examples

```
# Create a SpatRaster from scratch
x <- rast(nrows=108, ncols=21, xmin=0, xmax=10)

# Create a SpatRaster from a file
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)

# A file with multiple layers. This one is special as the layers are RGB color channels
s <- rast(system.file("ex/logo.tif", package="terra"))

# remove the color channels
#plot(s)
#RGB(s) <- NULL
#plot(s)

# Create a skeleton with no associated cell values
rast(s)

# from a matrix
m <- matrix(1:25, nrow=5, ncol=5)
rm <- rast(m)

# from a "xyz" data.frame
```

```
d <- as.data.frame(rm, xy=TRUE)
head(d)
rast(d, type="xyz")
```

---

rasterize *Rasterize vector data*

Description

Transfer values associated with the geometries of vector data to a raster

Usage

```
## S4 method for signature 'SpatVector,SpatRaster'
rasterize(x, y, field="", fun, ..., background=NA, touches=FALSE, update=FALSE,
cover=FALSE, by=NULL, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'matrix,SpatRaster'
rasterize(x, y, values=1, fun, ..., background=NA, update=FALSE,
by=NULL, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatVector or a two-column matrix (point coordinates)
`y`	SpatRaster
`field`	character or numeric. If `field` is a character, it should a variable name in `x`. If `field` is numeric it typically is a single number or a vector of length `nrow(x)`. The values are recycled to `nrow(x)`
`values`	typically a numeric vector of length `1` or `nrow(x)`. If the length is below `nrow(x)` the values will be recycled to `nrow(x)`. Only used when `x` is a matrix. Can also be a matrix or data.frame
`fun`	summarizing function for when there are multiple geometries in one cell. For lines and polygons you can only use `"min"`, `"max"`, `"mean"`, `"count"` and `"sum"` For points you can use any function that returns a single number; for example `mean`, `length` (to get a count), `min` or `max`
`...`	additional arguments passed to `fun`
`background`	numeric. Value to put in the cells that are not covered by any of the features of `x`. Default is `NA`
`touches`	logical. If `TRUE`, all cells touched by lines or polygons are affected, not just those on the line render path, or whose center point is within the polygon. If `touches=TRUE`, add cannot be `TRUE`
`update`	logical. If `TRUE`, the values of the input SpatRaster are updated

cover logical. If TRUE and the geometry of x is polygons, the fraction of a cell that is covered by the polygons is returned. This is estimated by determining presence/absence of the polygon in at least 100 sub-cells (more of there are very few cells)

by character or numeric value(s) to split x into multiple groups. There will be a separate layer for each group returned. If x is a SpatVector, by can be a column number or name. If x is a matrix, by should be a vector that identifies group membership for each row in x

filename character. Output filename

overwrite logical. If TRUE, filename is overwritten

wopt list with additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

rasterizeGeom, rasterizeWin, mask

Examples

```
r <- rast(xmin=0, ncols=18, nrows=18)

# generate points
set.seed(1)
p <- spatSample(r, 1000, xy=TRUE, replace=TRUE)

# rasterize points as a matrix
x <- rasterize(p, r, fun=sum)
y <- rasterize(p, r, value=1:nrow(p), fun=max)

# rasterize points as a SpatVector
pv <- vect(p)
xv <- rasterize(pv, r, fun=sum)

# Polygons
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
r <- rast(v, ncols=75, nrows=100)
z <- rasterize(v, r, "NAME_2")
plot(z)
lines(v)
```

`rasterizeGeom` *Rasterize geometric properties of vector data*

Description

Rasterization of geometric properties of vector data. You can get the count of the number of geometries in each cell; the area covered by polygons; the length of the lines; or the number of lines that cross the boundary of each cell. See `rasterize` for standard rasterization (of attribute values associated with geometries).

The area of polygons is intended for summing the area of polygons that are relatively small relative to the raster cells, and for when there may be multiple polygons per cell. See `rasterize(fun="sum")` for counting large polygons and `rasterize(cover=TRUE)` to get the fraction that is covered by larger polygons.

Usage

```
## S4 method for signature 'SpatVector,SpatRaster'
rasterizeGeom(x, y, fun="count", unit="m", filename="", ...)
```

Arguments

`x`	SpatVector
`y`	SpatRaster
`fun`	character. "count", "area", "length", or "crosses"
`unit`	character. "m" or "km"
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`rasterize`

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
r <- rast(v, res=.1)

# length of lines
lns <- as.lines(v)
x <- rasterizeGeom(lns, r, fun="length", "km")
```

```
# count of points
set.seed(44)
pts <- spatSample(v, 100)
y <- rasterizeGeom(pts, r)

# area of polygons
pols <- buffer(pts, 1000)
z <- rasterizeGeom(pols, r, fun="area")
```

---

rasterizeWin *Rasterize points with a moving window*

Description

Rasterize points using a circle (or ellipse) as moving window. For each raster cell, the points (`x, y`) that fall within the window centered on that cell are considered. A function is used to compute a summary value (e.g. "mean") for the values (`z`) associated with these points.

This can result in much smoother results compared to the standard `rasterize` method.

Usage

```
## S4 method for signature 'SpatVector,SpatRaster'
rasterizeWin(x, y, field, win="circle", pars, fun, ..., cvars=FALSE,
  minPoints=1, fill=NA, filename="", wopt=list())

## S4 method for signature 'data.frame,SpatRaster'
rasterizeWin(x, y, win="circle", pars, fun, ..., cvars=FALSE,
      minPoints=1, fill=NA, filename="", wopt=list())
```

Arguments

`x`	SpatVector or matrix with at least three columns ((x, y) coordinates and a variable to be rasterized)
`y`	SpatRaster
`field`	character. field name in SpatVector `x` with the values to rasterize
`win`	character to choose the window type. Can be "circle", "ellipse", "rectangle", or "buffer"
`pars`	parameters to define the window. If `win="circle"` or `win="buffer"`, a single number to set the radius of the circle or the width of the buffer. If `win="ellipse"`, either two numbers (the x and y-axis) or three numbers the axes and a rotation (in degrees). If `win="rectangle"`, either two (width, height) or three (width, height) and the rotation in degrees. The unit of the radius/width/height/axis parameters is that of the coordinate reference system (it is not expressed as cells). That is, if you have a lon/lat crs, there is no conversion of degrees to meters or vice-versa.

`fun`	function to summarize the values for each cell. If `cvars=FALSE`, functions must take a numeric vector and return (in all cases) one or more numbers. If `cvars=TRUE`, and multiple variables are used, the function must take a single argument (a data.frame with the names variables). For `win="circle"` and `win="ellipse"` there are two additional character values that can be used: `"distto"` (average distance to the points from the center of the cell) and `"distbetween"` (average distance between the points inside the window)
`...`	additional named arguments passed to `fun`
`minPoints`	numeric. The minimum number of points to use. If fewer points are found in a search ellipse it is considered empty and the fill value is returned
`fill`	numeric. value to use to fill cells with empty search areas
`cvars`	logical. When using multiple fields, should `fun` operate on all of them at once? If not, `fun` is applied to each variable separately
`filename`	character. Output filename
`wopt`	list with additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

`rasterize`, `rasterizeGeom`, `interpNear`, `interpIDW`

Examples

```
r <- rast(ncol=100, nrow=100, crs="local", xmin=0, xmax=50, ymin=0, ymax=50)
set.seed(100)
x <- runif(50, 5, 45)
y <- runif(50, 5, 45)
z <- sample(50)
xyz <- data.frame(x,y,z)

r <- rasterizeWin(xyz, r, fun="count", pars=5)

rfuns <- c("count", "min", "max", "mean")
x <- lapply(rfuns, function(f) rasterizeWin(xyz, r, fun=f, pars=5))
names(x) <- rfuns
x <- rast(x)
#plot(x)
```

rcl *Combine row, column, and layer numbers*

Description

Get a matrix with the combination of row, column, and layer numbers

Usage

```
## S4 method for signature 'SpatRaster'
rcl(x, row=NULL, col=NULL, lyr=NULL)
```

Arguments

x	SpatRaster
row	positive integer that are row number(s), a list thereof, or NULL for all rows
col	as above for columns
lyr	as above for layers

Details

If a list is used for at least one of `row`, `col` or `lyr`, these are evaluated in parallel. That is combinations are made for each list element, not across list elements. If, in this case another argument is not a list it has to have either length 1 (used for all cases) or have the same length as the (longest) list, in which case the value is coerced into a list with `as.list`

If multiple arguments are a list but they have different lengths, theyare recycled to the longest list.

Value

matrix

See Also

`rowColCombine`, `cellFromRowCol`

Examples

```
x <- rast(ncol=5, nrow=5, nlyr=2)
values(x) <- 1:size(x)

rcl(x, 1, 2:3, 1:2)

i <- rcl(x, 1, list(1:2, 3:4), 1:2)
i

# get the values for these cells
x[i]
```

readwrite *Read from, or write to, file*

Description

Methods to read from or write chunks of values to or from a file. These are low level methods for programmers. Use writeRaster if you want to save an entire SpatRaster to file in one step. It is much easier to use.

To write chunks, begin by opening a file with `writeStart`, then write values to it in chunks using the list that is returned by `writeStart`. When writing is done, close the file with `writeStop`.

`blocks` only returns chunk size information. This can be useful when reading, but not writing, raster data.

Usage

```
## S4 method for signature 'SpatRaster'
readStart(x)

## S4 method for signature 'SpatRaster'
readStop(x)

## S4 method for signature 'SpatRaster'
readValues(x, row=1, nrows=nrow(x), col=1, ncols=ncol(x), mat=FALSE, dataframe=FALSE, ...)

## S4 method for signature 'SpatRaster,character'
writeStart(x, filename="", overwrite=FALSE, n=4, sources="", ...)

## S4 method for signature 'SpatRaster'
writeStop(x)

## S4 method for signature 'SpatRaster,vector'
writeValues(x, v, start, nrows)

## S4 method for signature 'SpatRaster'
blocks(x, n=4)

fileBlocksize(x)
```

Arguments

`x`	SpatRaster
`filename`	character. Output filename
`v`	vector with cell values to be written
`start`	integer. Row number (counting starts at 1) from where to start writing `v`
`row`	positive integer. Row number to start from, should be between 1 and nrow(x)

nrows	positive integer. How many rows?
col	positive integer. Column number to start from, should be between 1 and ncol(x)
ncols	positive integer. How many columns? Default is the number of columns left after the start column
mat	logical. If `TRUE`, values are returned as a numeric matrix instead of as a vector, except when `dataframe=TRUE`. If any of the layers of `x` is a factor, the level index is returned, not the label. Use `dataframe=TRUE` to get the labels
dataframe	logical. If `TRUE`, values are returned as a `data.frame` instead of as a vector (also if matrix is `TRUE`)
overwrite	logical. If `TRUE`, `filename` is overwritten
n	positive integer indicating how many copies the data may be in memory at any point in time. This is used to determine how many blocks (large) datasets need to be read
sources	character. Filenames that may not be overwritten because they are used as input to the function. Can be obtained with `sources(x)`
...	For `writeStart`: additional arguments for writing files as in `writeRaster` For `readValues`: additional arguments for `data.frame` (and thus only relevant when `dataframe=TRUE`)

Value

`readValues` returns a vector, matrix, or data.frame

`writeStart` returns a list that can be used for processing the file in chunks.

The other methods invisibly return a logical value indicating whether they were successful or not. Their purpose is the side-effect of opening or closing files.

---

rectify *Rectify a SpatRaster*

Description

Rectify a rotated SpatRaster into a non-rotated object

Usage

```
## S4 method for signature 'SpatRaster'
rectify(x, method="bilinear", aoi=NULL, snap=TRUE,
         filename="", ...)
```

Arguments

x	SpatRaster to be rectified
method	character. Method used to for resampling. See resample
aoi	SpatExtent or SpatRaster to crop x to a smaller area of interest; Using a SpatRaster allowing to set the exact output extent and output resolution
snap	logical. If TRUE, the origin and resolution of the output are the same as would the case when aoi = NULL. Only relevant if aoi is a SpatExtent
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

is.rotated

---

regress *Cell level regression*

Description

Run a regression model for each cell of a SpatRaster. The independent variable can either be defined by a vector, or another SpatRaster to make it spatially variable.

Usage

```
## S4 method for signature 'SpatRaster,numeric'
regress(y, x, formula=y~x, na.rm=FALSE, cores=1, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatRaster,SpatRaster'
regress(y, x, formula=y~x, na.rm=FALSE, cores=1, filename="", overwrite=FALSE, ...)
```

Arguments

y	SpatRaster
x	SpatRaster or numeric (of the same length as nlyr(x)
formula	regression formula in the general form of y ~ x. You can add additional terms such as I(x^2)
na.rm	logical. Remove NA values?
cores	positive integer. If cores > 1, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object.
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
...	list with named options for writing files as in writeRaster

Value

SpatRaster

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
x <- regress(s, 1:nlyr(s))
```

---

relate *Spatial relationships between geometries*

Description

`relate` returns a logical matrix indicating the presence or absence of a specific spatial relationships between the geometries in `x` and `y`.

`is.related` returns a logical vector indicating the presence or absence of a specific spatial relationships between `x` and any of the geometries in `y`

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
relate(x, y, relation, pairs=FALSE, na.rm=TRUE)

## S4 method for signature 'SpatVector,missing'
relate(x, y, relation, pairs=FALSE, na.rm=TRUE)

## S4 method for signature 'SpatVector,SpatVector'
is.related(x, y, relation)
```

Arguments

`x`	SpatVector or SpatExtent
`y`	missing or as for `x`
`relation`	character. One of "intersects", "touches", "crosses", "overlaps", "within", "contains", "covers", "coveredby", "disjoint". Or a "DE-9IM" string such as "FF*FF****". See wikipedia or geotools doc
`pairs`	logical. If `TRUE` a two-column matrix is returned with the indices of the cases where the requested relation is `TRUE`. This is especially helpful when dealing with many geometries as the returned value is generally much smaller
`na.rm`	logical. If `TRUE` and `pairs=TRUE`, geometries in `x` for which there is no related geometry in `y` are omitted

Value

matrix (relate) or vector (is.related)

See Also

adjacent, nearby, intersect, crop

Examples

```
# polygons
p1 <- vect("POLYGON ((0 0, 8 0, 8 9, 0 9, 0 0))")
p2 <- vect("POLYGON ((5 6, 15 6, 15 15, 5 15, 5 6))")
p3 <- vect("POLYGON ((8 2, 9 2, 9 3, 8 3, 8 2))")
p4 <- vect("POLYGON ((2 6, 3 6, 3 8, 2 8, 2 6))")
p5 <- vect("POLYGON ((2 12, 3 12, 3 13, 2 13, 2 12))")
p6 <- vect("POLYGON ((10 4, 12 4, 12 7, 11 7, 11 6, 10 6, 10 4))")

p <- rbind(p1, p2, p3, p4, p5, p6)
plot(p, col=rainbow(6, alpha=.5))
lines(p, lwd=2)
text(p)

## relate SpatVectors
relate(p1, p2, "intersects")
relate(p1, p3, "touches")
relate(p1, p5, "disjoint")
relate(rbind(p1, p2), p4, "disjoint")

## relate geometries within SpatVectors
# which are completely separated?
relate(p, relation="disjoint")

# which touch (not overlap or within)?
relate(p, relation="touches")
# which overlap (not merely touch, and not within)?
relate(p, relation="overlaps")
# which are within (not merely overlap)?
relate(p, relation="within")

# do they touch or overlap or are within?
relate(p, relation="intersects")

all(relate(p, relation="intersects") ==
  (relate(p, relation="overlaps") |
   relate(p, relation="touches") |
   relate(p, relation="within")))

#for polygons, "coveredby" is "within"
relate(p, relation="coveredby")

# polygons, lines, and points

pp <- rbind(p1, p2)
L1 <- vect("LINESTRING(1 11, 4 6, 10 6)")
L2 <- vect("LINESTRING(8 14, 12 10)")
L3 <- vect("LINESTRING(1 8, 12 14)")
```

```
lns <- rbind(L1, L2, L3)
pts <- vect(cbind(c(7,10,10), c(3,5,6)))

plot(pp, col=rainbow(2, alpha=.5))
text(pp, paste0("POL", 1:2), halo=TRUE)
lines(pp, lwd=2)
lines(lns, col=rainbow(3), lwd=4)
text(lns, paste0("L", 1:3), halo=TRUE)
points(pts, cex=1.5)
text(pts, paste0("PT", 1:3), halo=TRUE, pos=4)


relate(lns, relation="crosses")
relate(lns, pp, relation="crosses")
relate(lns, pp, relation="touches")
relate(lns, pp, relation="intersects")

relate(lns, pp, relation="within")
# polygons can contain lines or points, not the other way around
relate(lns, pp, relation="contains")
relate(pp, lns, relation="contains")
# points and lines can be covered by polygons
relate(lns, pp, relation="coveredby")

relate(pts, pp, "within")
relate(pts, pp, "touches")
relate(pts, lns, "touches")
```

---

rep *Replicate layers*

Description

Replicate layers in a SpatRaster

Usage

```
## S4 method for signature 'SpatRaster'
rep(x, ...)
```

Arguments

x	SpatRaster
...	arguments as in rep

Value

SpatRaster

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
x <- rep(s, 2)
nlyr(x)
names(x)
x
```

`replace_dollar` *Replace with $<-*

Description

Replace a layer of a SpatRaster, or an attribute variable of a SpatVector

Usage

```
## S4 replacement method for signature 'SpatRaster'
x$name <- value

## S4 replacement method for signature 'SpatVector'
x$name<-value

## S4 replacement method for signature 'SpatExtent'
x$name <- value
```

Arguments

`x`	SpatRaster, SpatVector or SpatExtent
`name`	character. If `x` is a SpatRaster: layer name. If `x` is a SpatVector: variable name. If `x` is a SpatExtent: "xmin", "xmax". "ymin" or "ymax"
`value`	if `x` is a SpatRaster, a SpatRaster for which this `TRUE`: `nlyr(value) == length(i)`; if `x` is a SpatVector, a vector of new values; if `x` is a SpatExtent a single number

Value

Same as `x`

See Also

`[[<-`, `[<-`, `$`

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$ID_1 <- LETTERS[1:12]
v$new <- sample(12)
values(v)
```

replace_layers *Replace layers or variables*

Description

Replace the layers of SpatRaster with (layers from) another SpatRaster or replace variables of a SpatVector. You can also create new layers/variables with these methods.

Usage

```
## S4 replacement method for signature 'SpatRaster,numeric'
x[[i]] <- value

## S4 replacement method for signature 'SpatRaster,character'
x[[i]] <- value

## S4 replacement method for signature 'SpatVector,numeric'
x[[i]] <- value

## S4 replacement method for signature 'SpatVector,character'
x[[i]] <- value
```

Arguments

`x`	SpatRaster or SpatVector
`i`	if `x` is a SpatRaster: layer number(s) of name(s). If `x` is a SpatVector: variable number(s) or name(s) (column of the attributes)
`value`	if `x` is a SpatRaster: SpatRaster for which this `TRUE`: `nlyr(value) == length(i)`. if `x` is a SpatVector: vector or data.frame

Value

SpatRaster

See Also

`$<-`, `[<-`

Examples

```
# raster
s <- rast(system.file("ex/logo.tif", package="terra"))
s[["red"]] <- mean(s)
s[[2]] <- sqrt(s[[1]])

# vector
v <- vect(system.file("ex/lux.shp", package="terra"))
v[["ID_1"]] <- 12:1
```

replace_values *Replace values of a SpatRaster*

Description

Replace values of a SpatRaster. These are convenience functions for smaller objects only. For larger rasters see link{classify} or subst

Usage

```
## S4 replacement method for signature 'SpatRaster,ANY,ANY,ANY'
x[i, j, k] <- value

## S4 replacement method for signature 'SpatVector,ANY,ANY'
x[i, j] <- value

## S4 replacement method for signature 'SpatExtent,numeric,missing'
x[i, j] <- value
```

Arguments

x	SpatRaster
i	row numbers. numeric, logical, or missing for all rows. Can also be a SpatRaster or SpatVector
j	column numbers. numeric, logical or missing for all columns
k	layer number. numeric, logical or missing for all layers
value	numeric, matrix, or data.frame

Value

SpatRaster

See Also

classify, subst, set.values, values, [[<-

Examples

```
## SpatRaster
r <- rast(ncols=5, nrows=5, xmin=0, xmax=5, ymin=0, ymax=5)
r[] <- 1:25
r[1,] <- 5
r[,2] <- 10
r[r>10] <- NA

## SpatVector
f <- system.file("ex/lux.shp", package="terra")
```

```
v <- vect(f)
v[2,2] <- "hello"
v[1,] <- v[10,]
v[,3] <- v[,1]
v[2, "NAME_2"] <- "terra"
head(v, 3)
```

---

resample *Transfer values of a SpatRaster to another one with a different geometry*

Description

resample transfers values between SpatRaster objects that do not align (have a different origin and/or resolution). See `project` to change the coordinate reference system (crs).

If the origin and extent of the input and output are the same, you should consider using these other functions instead: `aggregate`, `disagg`, `extend` or `crop`.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
resample(x, y, method, threads=FALSE, filename="", ...)
```

Arguments

`x`	SpatRaster to be resampled
`y`	SpatRaster with the geometry that `x` should be resampled to
`method`	character. Method used for estimating the new cell values. One of: `near`: nearest neighbor. This method is fast, and it can be the preferred method if the cell values represent classes. It is not a good choice for continuous values. This is used by default if the first layer of `x` is categorical. `bilinear`: bilinear interpolation. This is the default if the first layer of `x` is numeric (not categorical). (3x3 cell window). `cubic`: cubic interpolation. (5x5 cell window). `cubicspline`: cubic B-spline interpolation. (5x5 cell window). `lanczos`: Lanczos windowed sinc resampling. (7x7 cell window). `sum`: the weighted sum of all non-NA contributing grid cells. `min, q1, med, q3, max, average, mode, rms`: the minimum, first quartile, median, third quartile, maximum, mean, mode, or root-mean-square value of all non-NA contributing grid cells.
`threads`	logical. If `TRUE` multiple threads are used (faster for large files)
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

aggregate, disagg, crop, project

Examples

```
r <- rast(nrows=3, ncols=3, xmin=0, xmax=10, ymin=0, ymax=10)
values(r) <- 1:ncell(r)
s <- rast(nrows=25, ncols=30, xmin=1, xmax=11, ymin=-1, ymax=11)
x <- resample(r, s, method="bilinear")

opar <- par(no.readonly =TRUE)
par(mfrow=c(1,2))
plot(r)
plot(x)
par(opar)
```

---

rescale *rescale*

Description

Rescale a SpatVector or SpatRaster. This may be useful to make small inset maps or for georeferencing.

Usage

```
## S4 method for signature 'SpatRaster'
rescale(x, fx=0.5, fy=fx, x0, y0)

## S4 method for signature 'SpatVector'
rescale(x, fx=0.5, fy=fx, x0, y0)
```

Arguments

x	SpatVector or SpatRaster
fx	numeric > 0. The horizontal scaling factor
fy	numeric > 0. The vertical scaling factor
x0	numeric. x-coordinate of the center of rescaling. If missing, the center of the extent of x is used
y0	numeric. y-coordinate of the center of rescaling. If missing, the center of the extent of x is used

Value

Same as x

See Also

t, shift, flip, rotate, inset

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- rescale(v, 0.2)
plot(v)
lines(w, col="red")
```

RGB *Layers representing colors*

Description

With RGB you can get or set the layers to be used as Red, Green and Blue when plotting a SpatRaster. Currently, a benefit of this is that plot will send the object to plotRGB. You can also associated the layers with another color space (HSV, HSI or HSL)

With colorize you can convert a three-layer RGB SpatRaster into other color spaces. You can also convert it into a single-layer SpatRaster with a color-table.

Usage

```
## S4 method for signature 'SpatRaster'
RGB(x, value=NULL, type="rgb")

## S4 replacement method for signature 'SpatRaster'
RGB(x, ..., type="rgb")<-value

## S4 method for signature 'SpatRaster'
colorize(x, to="hsv", alpha=FALSE, stretch=NULL,
grays=FALSE, NAzero=FALSE, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatRaster'
has.RGB(x, strict=TRUE)
```

Arguments

x	SpatRaster
value	three (or four) positive integers indicating the layers that are red, green and blue (and optionally a fourth transparency layer). Or NULL to remove the RGB settings

`type`	character. The color space. One of "rgb" "hsv", "hsi" and "hsl"
`to`	character. The color space to transform the values to. If `x` has RGB set, you can transform these to "hsv", "hsi" and "hsl", or use "col" to create a single layer with a color table. You can also use "rgb" to back transform to RGB
`alpha`	logical. Should an alpha (transparency) channel be included? Only used if `x` has a color-table and `to="rgb"`
`stretch`	character. Option to stretch the values to increase contrast: "lin" (linear) or "hist" (histogram). Only used for transforming RGB to col
`grays`	logical. If `TRUE`, a gray-scale color-table is created. Only used for transforming RGB to col
`NAzero`	logical. If `TRUE`, NAs are treated as zeros such that a color can be returned if at least one of the three channels has a value. Only used for transforming RGB to (`"col"`)
`strict`	logical. If `TRUE`, the function returns `FALSE` if a color space such as "hsv", "hsi" and "hsl" is used
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional arguments for writing files as in `writeRaster`

See Also

`set.RGB`

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
RGB(r)
plot(r)
has.RGB(r)
RGB(r) <- NULL
has.RGB(r)
plot(r)
RGB(r) <- c(3,1,2)
# same as
# r <- RGB(r, c(3,1,2))

plot(r)

RGB(r) <- 1:3
x <- colorize(r, "col")
y <- colorize(r, "hsv")
z <- colorize(y, "rgb")
```

roll — *Rolling (moving) functions*

Description

Compute "rolling" or "moving" values, such as the "rolling average" for each cell in a SpatRaster.

See `focal` for spatially moving averages and similar computations. And see `cumsum` and other cum* functions to compute cumulate values.

Usage

```
## S4 method for signature 'SpatRaster'
roll(x, n, fun=mean, type="around", circular=FALSE,
na.rm=FALSE, filename="", ..., wopt=list())

## S4 method for signature 'numeric'
roll(x, n, fun=mean, type="around", circular=FALSE, na.rm=FALSE, ...)
```

Arguments

`x`	SpatRaster or numeric
`n`	integer > 1. The size of the "window", that is, the number of sequential cells to use in `fun`
`fun`	a function like mean, min, max, sum
`type`	character. One of "around", "to", or "from". The choice indicates which values should be used in the computation. The focal cell is always used. If type is "around", `(n-1)/2` before and after the focal cell are also included. If type = "from", `n-1` cells are after the focal cell are included. If type = "to", `n-1` cells before the focal cell are included. For example, when using n=3 for element 5 of a vector; "around" used elements 4,5,6; "to" used elements 3,4,5, and "from" uses elements 5,6,7
`circular`	logical. If `TRUE`, the data are considered to have a circular nature (e.g. days or months of the year), such that there are no missing values before first or after the last value.
`na.rm`	logical. If `TRUE`, `NA` values should be ignored (by `fun`)
`filename`	character. Output filename
`...`	additional arguments for `fun`
`wopt`	list with named options for writing files as in `writeRaster`

Value

Same as `x`

See Also

cumsum, focal

Examples

```
## numeric
roll(1:12, 3, mean)
roll(1:12, 3, mean, "to")
roll(1:12, 3, mean, circular=TRUE)

## SpatRaster
r <- rast(ncol=2, nrow=2, nlyr=10, vals=1)
r[1,2] = 2
r[2,2] = 4

roll(r, n=3, "sum", "from", na.rm=FALSE) |> values()
roll(r, n=3, "sum", "from", na.rm=TRUE) |> values()
roll(r, n=3, "sum", "from", circular=TRUE) |> values()

roll(r, n=3, "sum", "to", na.rm=TRUE) |> values()

roll(r, n=3, "sum", "around", circular=TRUE) |> values()
```

---

rotate *Rotate data along longitude*

Description

Rotate a SpatRaster that has longitude coordinates from 0 to 360, to standard coordinates between -180 and 180 degrees (or vice-versa). Longitude between 0 and 360 is frequently used in global climate models.

Rotate a SpatVector as for a SpatRaster `split=TRUE`, or to correct for coordinates that are connected across the date line (and end up at the "other side" of the longitude scale) are reconnected.

Usage

```
## S4 method for signature 'SpatRaster'
rotate(x, left=TRUE, filename="", ...)

## S4 method for signature 'SpatVector'
rotate(x, longitude=0, split=FALSE, left=TRUE, normalize=FALSE)
```

Arguments

`x`	SpatRaster or SpatVector
`left`	logical. If `TRUE`, rotate to the left, else to the right
`filename`	character. Output filename

...	additional arguments for writing files as in writeRaster
longitude	numeric. The longitude around which to rotate
split	logical. Should geometries be split at longitude?
normalize	logical. Should the output be normalized to longitudes between -180 and 180? See normalize.longitude

Value

SpatRaster

See Also

shift and spin

Examples

```
x <- rast(nrows=9, ncols=18, nl=3, xmin=0, xmax=360)
v <- rep(as.vector(t(matrix(1:ncell(x), nrow=9, ncol=18))), 3)
values(x) <- v
z <- rotate(x)

## Not run:
#SpatVector
p <- rbind(c(3847903, 1983584 ), c(3847903, 5801864), c(8301883, 5801864), c(8301883, 1983584 ))
p <- vect(p, "polygons", crs="+init=EPSG:3347")
d <- densify(p, 100000)
g <- project(d, "+proj=longlat")

x <- rotate(g, 50)
plot(g)
lines(x, col="red")

## End(Not run)

## rotate countries to 0-360 longitude
#w <- geodata::world(path=".")
#x <- rotate(w, long=0, split=TRUE, left=FALSE)
```

---

rowSums *row/col sums and means for SpatRaster*

Description

Sum or average values of SpatRaster layers by row or column.

Usage

```
## S4 method for signature 'SpatRaster'
rowSums(x, na.rm=FALSE, dims=1L, ...)
## S4 method for signature 'SpatRaster'
colSums(x, na.rm=FALSE, dims=1L, ...)
## S4 method for signature 'SpatRaster'
rowMeans(x, na.rm=FALSE, dims=1L, ...)
## S4 method for signature 'SpatRaster'
colMeans(x, na.rm=FALSE, dims=1L, ...)
```

Arguments

`x`	SpatRaster
`na.rm`	logical. If `TRUE`, `NA` values are ignored
`dims`	this argument is ignored
`...`	additional arguments (none implemented)

Value

matrix

See Also

See `global` for summing all cells values

Examples

```
r <- rast(ncols=2, nrows=5, nl=2, vals=1:20)
rowSums(r)
colSums(r)
colMeans(r)
```

---

same.crs — *Compare coordinate reference systems*

Description

The function takes two coordinate reference system descriptions and compares them for equality.

Usage

```
same.crs(x, y)
```

Arguments

`x`	character, SpatRaster, SpatVector, CRS, or other object that returns something intelligible with`crs(x)`
`y`	same types as for `x`

Value

logical

Examples

```
r <- rast()
same.crs(r, "+proj=longlat")

same.crs(r, "+proj=utm +zone=1")
```

`sapp` *Apply a terra function that takes only a single layer and returns a SpatRaster to all layers of a SpatRaster*

Description

Apply to all layers of a SpatRaster a function that only takes a single layer SpatRaster and returns a SpatRaster (these are rare). In most cases you can also use `lapply` or `sapply` for this.

Or apply the same method to each sub-dataset (SpatRaster) in a SpatRasterDataset

Usage

```
## S4 method for signature 'SpatRaster'
sapp(x, fun, ..., filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterDataset'
sapp(x, fun, ..., filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster or SpatRasterDataset
`fun`	if `x` is a `SpatRaster`: a function that takes a SpatRaster argument and can be applied to each layer of `x` (e.g. `terrain`. if `x` is a `SpatRasterDataset`: a function that is applied to all layers of the SpatRasters in `x` (e.g. `mean`
`...`	additional arguments to be passed to `fun`
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Value

SpatRaster

See Also

`lapp`, `app`, `tapp`, `lapply`

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra")) + 1

#SpatRasterDataset
sd <- sds(s*2, s/2)
y <- sapp(sd, mean)
z <- sapp(sd, \(i) 2 * mean(i))
```

---

sbar *scale bar*

Description

Add a scale bar to a map

Usage

```
sbar(d, xy=NULL, type="line", divs=2, below="", lonlat=NULL, labels,
adj=c(0.5, -1), lwd=2, xpd=TRUE, ticks=FALSE, scaleby=1, halo=TRUE, ...)
```

Arguments

d	numeric. Distance covered by the scale bar. For the scale bar, it should be in the units of the coordinates of the plot (map), and in km for angular (longitude/latitude) data; see argument `lonlat`. It can also be missing
xy	numeric. x and y coordinate to place the scale bar. It can also be one of following character values: "bottomleft", "bottom", "bottomright", topleft", "top", "topright", "left", "right", or NULL
type	for `sbar`: "line" or "bar"
divs	number of divisions for a bar: 2 or 4
below	character. Text to go below the scale bar (e.g., "kilometers")
lonlat	logical or NULL. If logical, `TRUE` indicates if the plot is using longitude/latitude coordinates. If `NULL` this is guessed from the plot's coordinates
labels	vector of three numbers to label the scale bar (beginning, midpoint, end)
adj	adjustment for text placement
lwd	line width for the "line" type of the scale bar
xpd	logical. If `TRUE`, the scale bar can be outside the plotting area
ticks	logical or numeric. If not `FALSE`, tick marks are added to a "line" scale bar. The length of the tick marks can be specified
scaleby	numeric. If `labels` is not provided. The labels are divided by this number. For example, use 1000 to go from m to km
halo	logical. If `TRUE` the "line" type scale bar gets a white background
...	graphical arguments to be passed to other methods

Value

none

See Also

north, plot, inset

Examples

```
f <- system.file("ex/meuse.tif", package="terra")
r <- rast(f)
plot(r)
sbar()
sbar(1000, xy=c(178500, 333500), type="bar", divs=4, cex=.8)
sbar(1000, xy="bottomright", divs=3, cex=.8, ticks=TRUE)
north(d=250, c(178550, 332500))


f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
plot(r, type="interval")
sbar(20, c(6.2, 50.1), type="bar", cex=.8, divs=4)
sbar(15, c(6.3, 50), type="bar", below="km", label=c(0,7.5,15), cex=.8)
sbar(15, c(6.65, 49.8), cex=.8, label=c(0,"km",15))
north(type=2)
sbar(15, c(6.65, 49.7), cex=.8, label="15 kilometer", lwd=5)
sbar(15, c(6.65, 49.6), divs=4, cex=.8, below="km")
```

---

scale *Scale values*

Description

Center and/or scale raster data. For details see scale

Usage

```
## S4 method for signature 'SpatRaster'
scale(x, center=TRUE, scale=TRUE)
```

Arguments

x	SpatRaster
center	logical or numeric. If TRUE, centering is done by subtracting the layer means (omitting NAs), and if FALSE, no centering is done. If center is a numeric vector (recycled to nlyr(x)), then each layer of x has the corresponding value from center subtracted from it.

scale — logical or numeric. If TRUE, scaling is done by dividing the (centered) layers of x by their standard deviations if center is TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done. If scale is a numeric vector (recycled to nlyr(x)), each layer of x is divided by the corresponding value. Scaling is done after centering.

Value

SpatRaster

See Also

scale

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
s <- scale(r)

## the equivalent, computed in steps
m <- global(r, "mean")
rr <- r - m[,1]
rms <- global(rr, "rms")
ss <- rr / rms[,1]
```

---

scatterplot — *Scatterplot of two SpatRaster layers*

Description

Scatterplot of the values of two SpatRaster layers

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
plot(x, y, maxcell=100000, warn=TRUE, nc, nr,
   maxnl=16, smooth=FALSE, gridded=FALSE, ncol=25, nrow=25, ...)
```

Arguments

x	SpatRaster
y	SpatRaster
maxcell	positive integer. Maximum number of cells to use for the plot
nc	positive integer. Optional. The number of columns to divide the plotting device in (when plotting multiple layers)

nr	positive integer. Optional. The number of rows to divide the plotting device in (when plotting multiple layers)
maxnl	positive integer. Maximum number of layers to plot (for multi-layer objects)
smooth	logical. If TRUE show a smooth scatterplot
gridded	logical. If TRUE the scatterplot is gridded (counts by cells)
warn	boolean. Show a warning if a sample of the pixels is used (for scatterplot only)
ncol	positive integer. Number of columns for gridding
nrow	positive integer. Number of rows for gridding
...	additional graphical arguments

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
plot(s[[1]], s[[2]])
plot(s, sqrt(s[[3:1]]))
```

---

scoff *Scale (gain) and offset*

Description

These functions can be used to get or set the scale (gain) and offset parameters used to transform values when reading raster data from a file. The parameters are applied to the raw values using the formula below:

`value <- value * scale + offset`

The default value for scale is 1 and for offset is 0. 'scale' is sometimes referred to as 'gain'.

Note that setting the scale and/or offset are intended to be used with values that are stored in a file. When values are memory, assigning scale or offset values will lead to the immediate computation of new values; in such cases it would be clearer to use `Arith-methods`.

Usage

```
## S4 method for signature 'SpatRaster'
scoff(x)

## S4 replacement method for signature 'SpatRaster'
scoff(x)<-value
```

Arguments

x	SpatRaster
value	two-column matrix with scale (first column) and offset (second column) for each layer. Or NULL to remove all scale and offset values

Value

matrix or changed SpatRaster

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
minmax(r)
scoff(r)
r[4603]

scoff(r) <- cbind(10, 5)
minmax(r)
scoff(r)
r[4603]
```

---

sds *Create a SpatRasterDataset*

Description

Methods to create a SpatRasterDataset. This is an object to hold "sub-datasets", each a SpatRaster that in most cases will have multiple layers.

See `describe` for getting information about the sub-datasets present in a file.

Usage

```
## S4 method for signature 'missing'
sds(x)

## S4 method for signature 'character'
sds(x, ids=0, opts=NULL, raw=FALSE)

## S4 method for signature 'SpatRaster'
sds(x, ...)

## S4 method for signature 'list'
sds(x)

## S4 method for signature 'array'
sds(x, crs="", extent=NULL)
```

Arguments

x	character (filename), or SpatRaster, or list of SpatRasters, or missing. If multiple filenames are provided, it is attempted to make SpatRasters from these, and combine them into a SpatRasterDataset

ids	optional. vector of integer subdataset ids. Ignored if the first value is not a positive integer
opts	character. GDAL dataset open options
raw	logical. If TRUE, scale and offset values are ignored
crs	character. Description of the Coordinate Reference System (map projection) in PROJ.4, WKT or authority:code notation. If this argument is missing, and the x coordinates are within -360 .. 360 and the y coordinates are within -90 .. 90, longitude/latitude is assigned
extent	SpatExtent
...	additional SpatRaster objects

Value

SpatRasterDataset

See Also

describe

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
x <- sds(s, s/2)
names(x) <- c("first", "second")
x
length(x)

# extract the second SpatRaster
x[2]

a <- array(1:9, c(3,3,3,3))
sds(a)
```

---

segregate *segregate*

Description

Create a SpatRaster with a layer for each class (value, or subset of the values) in the input SpatRaster. For example, if the input has vegetation types, this function will create a layer (presence/absence; dummy variable) for each of these classes.

This is called "one-hot encoding" or "dummy encoding" (for a dummy encoding scheme you can remove (any) one of the output layers as it is redundant).

Usage

```
## S4 method for signature 'SpatRaster'
segregate(x, classes=NULL, keep=FALSE, other=0, round=FALSE, digits=0, filename="", ...)
```

Arguments

x	SpatRaster
classes	numeric. The values (classes) for which layers should be made. If NULL all classes are used
keep	logical. If TRUE, cells that are of the class represented by a layer get that value, rather than a value of 1
other	numeric. Value to assign to cells that are not of the class represented by a layer
round	logical. Should the values be rounded first?
digits	integer. Number of digits to round the values to
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

Examples

```
r <- rast(nrows=5, ncols=5)
values(r) <- rep(c(1:4, NA), each=5)
b <- segregate(r)
bb <- segregate(r, keep=TRUE, other=NA)
```

---

sel *Spatial selection*

Description

Geometrically subset SpatRaster or SpatVector (to be done) by drawing on a plot (map).

Usage

```
## S4 method for signature 'SpatRaster'
sel(x, ...)

## S4 method for signature 'SpatVector'
sel(x, use="rec", show=TRUE, col="cyan", draw=TRUE, ...)
```

Arguments

x	SpatRaster or SpatVector
use	character indicating what to draw. One of "rec" (rectangle) or "pol" (polygon)
show	logical. If TRUE the selected geometries are shown on the map
col	color to be used for drawing if draw=TRUE
draw	logical. If TRUE the area drawn to select geometries is shown on the map
...	additional graphics arguments for drawing the selected geometries

Value

SpatRaster or SpatVector

See Also

crop and intersect to make an intersection and click and text to see cell values or geometry attributes.

Use draw to draw a SpatExtent of SpatVector that you want to keep.

Examples

```
## Not run:
# select a subset of a SpatRaster
r <- rast(nrows=10, ncols=10)
values(r) <- 1:ncell(r)
plot(r)
s <- sel(r) # now click on the map twice

# plot the selection on a new canvas:
x11()
plot(s)

# vector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
plot(v)
x <- sel(v) # now click on the map twice
x

## End(Not run)
```

---

selectHighest *select cells with high or low values*

Description

Identify n cells that have the highest or lowest values in the first layer of a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
selectHighest(x, n, low=FALSE)
```

Arguments

x	SpatRaster. Only the first layer is processed
n	The number of cells to select
low	logical. If TRUE, the lowest values are selected instead of the highest values

Value

SpatRaster

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- selectHighest(r, 1000)
y <- selectHighest(r, 1000, TRUE)

m <- merge(y-1, x)
levels(m) <- data.frame(id=0:1, elevation=c("low", "high"))
plot(m)
```

`selectRange` *Select the values of a range of layers, as specified by cell values in another SpatRaster*

Description

Use a single layer SpatRaster to select cell values from different layers in a multi-layer SpatRaster. The values of the SpatRaster to select layers (y) should be whole numbers between 1 and `nlyr(x)` (values outside this range are ignored).

See `rapp` for applying a function to a range of variable size.

See `extract` for extraction of values by cell, point, or otherwise.

Usage

```
## S4 method for signature 'SpatRaster'
selectRange(x, y, z=1, repint=0, filename="", ...)
```

Arguments

`x`	SpatRaster
`y`	SpatRaster. Cell values must be positive integers. They indicate the first layer to select for each cell
`z`	positive integer. The number of layers to select
`repint`	integer > 1 and < nlyr(x) allowing for repeated selection at a fixed interval. For example, if `x` has 36 layers, and the value of a cell in `y`=2 and `repint` = 12, the values for layers 2, 14 and 26 are returned
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

rapp, tapp, extract

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1
s <- c(r, r+2, r+5)
s <- c(s, s)
set.seed(1)
values(r) <- sample(3, ncell(r), replace=TRUE)
x <- selectRange(s, r)

x <- selectRange(s, r, 3)
```

---

serialize — *saveRDS and serialize for SpatVector and SpatRaster**

Description

serialize and saveRDS for SpatVector, SpatRaster, SpatRasterDataset and SpatRasterCollection. Note that these objects will first be "packed" with wrap, and after unserialize/readRDS they need to be unpacked with rast or vect.

Extensive use of these functions is not recommended. Especially for SpatRaster it is generally much more efficient to use writeRaster and write, e.g., a GTiff file.

Usage

```
## S4 method for signature 'SpatRaster'
saveRDS(object, file="", ascii = FALSE, version = NULL, compress=TRUE, refhook = NULL)

## S4 method for signature 'SpatRasterDataset'
saveRDS(object, file="", ascii = FALSE, version = NULL, compress=TRUE, refhook = NULL)

## S4 method for signature 'SpatRasterCollection'
saveRDS(object, file="", ascii = FALSE, version = NULL, compress=TRUE, refhook = NULL)

## S4 method for signature 'SpatVector'
saveRDS(object, file="", ascii = FALSE, version = NULL, compress=TRUE, refhook = NULL)

## S4 method for signature 'SpatRaster'
serialize(object, connection, ascii = FALSE, xdr = TRUE, version = NULL, refhook = NULL)

## S4 method for signature 'SpatVector'
serialize(object, connection, ascii = FALSE, xdr = TRUE, version = NULL, refhook = NULL)
```

Arguments

`object`	SpatVector, SpatRaster, SpatRasterDataset or SpatRasterCollection
`file`	file name to save object to
`connection`	see `serialize`
`ascii`	see `serialize` or `saveRDS`
`version`	see `serialize` or `saveRDS`
`compress`	see `serialize` or `saveRDS`
`refhook`	see `serialize` or `saveRDS`
`xdr`	see `serialize` or `saveRDS`

Value

Packed* object

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
p <- serialize(v, NULL)
head(p)
x <- unserialize(p)
x
```

---

setValues *Set the values of raster cells or of geometry attributes*

Description

Set cell values of a SpatRaster or the attributes of a SpatVector. For large SpatRasters use `init` instead to set values.

Usage

```
## S4 replacement method for signature 'SpatRaster,ANY'
values(x)<-value

## S4 method for signature 'SpatRaster,ANY'
setValues(x, values, keeptime=TRUE, keepunits=TRUE, keepnames=FALSE, props=FALSE)

## S4 replacement method for signature 'SpatVector,ANY'
values(x)<-value
```

Arguments

x	SpatRaster or SpatVector
value	For SpatRaster: numeric, matrix or data.frame. The length of the numeric values must match the total number of cells (ncell(x) * nlyr(x)), or be a single value. The number of columns of the matrix or data.frame must match the number of layers of x, and the number of rows must match the number of cells of x. It is also possible to use a matrix with the same number of rows as x and the number of columns that matches ncol(x) * nlyr(x). For SpatVector: data.frame, matrix, vector, or NULL
values	Same as for value
keeptime	logical. If TRUE the time stamps are kept
keepunits	logical. If FALSE the units are discarded
keepnames	logical. If FALSE the layer names are replaced by the column names in y (if present)
props	logical. If TRUE the properties (categories and color-table) are kept

Value

The same object type as x

See Also

values, init

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- setValues(r, 1:ncell(r))
x
values(x) <- runif(ncell(x))
x
head(x)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
values(v) <- data.frame(ID=1:12, name=letters[1:12])
head(v)
```

---

shade *Hill shading*

Description

Compute hill shade from slope and aspect layers (both in radians). Slope and aspect can be computed with function terrain.

A hill shade layer is often used as a backdrop on top of which another, semi-transparent, layer is drawn.

Usage

```
shade(slope, aspect, angle=45, direction=0, normalize=FALSE,
   filename="", overwrite=FALSE, ...)
```

Arguments

`slope`	SpatRasterwith slope values (in radians)
`aspect`	SpatRaster with aspect values (in radians)
`angle`	The elevation angle(s) of the light source (sun), in degrees
`direction`	The direction (azimuth) angle(s) of the light source (sun), in degrees
`normalize`	Logical. If `TRUE`, values below zero are set to zero and the results are multiplied with 255
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional arguments for writing files as in `writeRaster`

References

Horn, B.K.P., 1981. Hill shading and the reflectance map. Proceedings of the IEEE 69(1):14-47

See Also

`terrain`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
alt <- disagg(r, 10, method="bilinear")
slope <- terrain(alt, "slope", unit="radians")
aspect <- terrain(alt, "aspect", unit="radians")
hill <- shade(slope, aspect, 40, 270)
plot(hill, col=grey(0:100/100), legend=FALSE, mar=c(2,2,1,4))
plot(alt, col=rainbow(25, alpha=0.35), add=TRUE)

# A better hill shade may be achieved by combining
# different angles and directions. For example

h <- shade(slope, aspect, angle = c(45, 45, 45, 80), direction = c(225, 270, 315, 135))
h <- Reduce(mean, h)
```

`sharedPaths` *Shared paths*

Description

Get shared paths of line or polygon geometries. This can for geometries in a single SpatVector, or between two SpatVectors

Usage

```
## S4 method for signature 'SpatVector'
sharedPaths(x, y=NULL)
```

Arguments

x	SpatVector of lines or polygons
y	missing or SpatVector of lines or polygons

Value

SpatVector

See Also

gaps, topology

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
plot(v, col="light gray")
text(v, halo=TRUE)

x <- sharedPaths(v)
lines(x, col="red", lwd=2)
text(x, col="blue", halo=TRUE, cex=0.8)
head(x)

z <- sharedPaths(v[3,], v[12,])
```

shift — *Shift*

Description

Shift a SpatRaster, SpatVector or SpatExtent to another location.

Usage

```
## S4 method for signature 'SpatRaster'
shift(x, dx=0, dy=0, filename="", ...)

## S4 method for signature 'SpatVector'
shift(x, dx=0, dy=0)

## S4 method for signature 'SpatExtent'
shift(x, dx=0, dy=0)
```

Arguments

`x`	SpatRaster, SpatVector or SpatExtent
`dx`	numeric. The shift in horizontal direction
`dy`	numeric. The shift in vertical direction
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

Same as `x`

See Also

`flip`, `rotate`

Examples

```
r <- rast(xmin=0, xmax=1, ymin=0, ymax=1)
r <- shift(r, dx=1, dy=-1)

e <- ext(r)
shift(e, 5, 5)
```

sieve *Sieve filter*

Description

Apply a sieve filter. That is, remove "noise", by changing small clumps of cells with a value that is different from the surrounding cells, to the value of the largest neighboring clump.

Note that the numerical input values are truncated to integers.

Usage

```
## S4 method for signature 'SpatRaster'
sieve(x, threshold, directions=8, filename="", ...)
```

Arguments

`x`	SpatRaster, single layer with integer or categorical values
`threshold`	positive integer. Only clumps smaller than this threshold will be removed
`directions`	numeric to indicate which cells are connected. Either 4 to only consider the horizontal and vertical neighbors ("rook"), or 8 to consider the vertical, horizontal and diagonal neighbors
`filename`	character. Output filename
`...`	Options for writing files as in `writeRaster`

See Also

`focal`

Examples

```
r <- rast(nrows=18, ncols=18, xmin=0, vals=0, crs="local")
r[2, 5] <- 1
r[5:8, 2:3] <- 2
r[7:12, 10:15] <- 3
r[15:16, 15:18] <- 4
freq(r, bylayer=FALSE)

x <- sieve(r, 8)
y <- sieve(r, 9)
```

simplifyGeom *simplifyGeom geometries*

Description

Reduce the number of nodes used to represent geometries.

Usage

```
## S4 method for signature 'SpatVector'
simplifyGeom(x, tolerance=0.1, preserveTopology=TRUE, makeValid=TRUE)
```

Arguments

`x`	SpatVector of lines or polygons
`tolerance`	numeric. The minimum distance between nodes in units of the crs (i.e. degrees for long/lat)
`preserveTopology`	logical. If `TRUE` the topology of output geometries is preserved
`makeValid`	logical. If `TRUE`, `makeValid` is run after simplification to assure that the output polygons are valid

Value

SpatVector

See Also

`sharedPaths`, `gaps`, `is.valid`

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- simplifyGeom(v, .02, makeValid=FALSE)
e <- erase(w)
g <- gaps(e)
plot(e, lwd=5, border="light gray")
polys(g, col="red", border="red")
```

sort *Sort a SpatRaster or SpatVector*

Description

Sort the cell values of a SpatRaster across layers. You can also compute the sorting order.

Or sort the records of SpatVector (or data.frame) by specifying the column number(s) or names(s) to sort on.

Usage

```
## S4 method for signature 'SpatRaster'
sort(x, decreasing=FALSE, order=FALSE, filename="", ...)

## S4 method for signature 'SpatVector'
sort(x, v, decreasing=FALSE)
```

Arguments

`x`	SpatRaster
`decreasing`	logical. If `TRUE`, sorting is in decreasing order
`order`	logical. If `TRUE` the sorting order is returned instead of the sorted values
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`
`v`	character or numeric indicating the column(s) to sort on

Value

SpatRaster

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
r <- c(r, r/2, r*2)
sort(r)

ord <- sort(r, order=TRUE)
# these two are the same
ord[[1]]
which.min(r)
```

sources — *Data sources of a SpatRaster*

Description

Get the data sources of a SpatRaster or SpatVector or related object. Sources are either files (or similar resources) or "", meaning that they are in memory. You can use `hasValues` to check if in-memory layers actually have cell values.

Usage

```
## S4 method for signature 'SpatRaster'
sources(x, nlyr=FALSE, bands=FALSE)

## S4 method for signature 'SpatVector'
sources(x)

## S4 method for signature 'SpatRaster'
hasValues(x)

## S4 method for signature 'SpatRaster'
inMemory(x, bylayer=FALSE)
```

Arguments

`x`	SpatRaster, SpatRasterCollection, SpatVector or SpatVectorProxy
`nlyr`	logical. If `TRUE` for each source, the number of layers is returned
`bands`	logical. If `TRUE` for each source, the "bands" used, that is, the layer number in the source file, are returned
`bylayer`	logical. If `TRUE` a value is returned for each layer instead of for each source

Value

A vector of filenames, or `""` when there is no filename, if `nlyr` and `bands` are both `FALSE`. Otherwise a `data.frame`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
s <- rast(r)
values(s) <- 1:ncell(s)
rs <- c(r,r,s,r)
sources(rs)
hasValues(r)
x <- rast()
hasValues(x)
```

SpatExtent-class *Class "SpatExtent"*

Description

Objects of class SpatExtent are used to define the spatial extent (extremes) of objects of the SpatRaster class.

Objects from the Class

You can use the `ext` function to create SpatExtent objects, or to extract them from a SpatRaster, SpatVector or related objects.

Methods

show display values of a SpatExtent object

Examples

```
e <- ext(-180, 180, -90, 90)
e
```

SpatRaster-class *SpatRaster class*

Description

A `SpatRaster` represents a rectangular part of the world that is sub-divided into rectangular cells of equal area (in terms of the units of the coordinate reference system). For each cell can have multiple values ("layers").

An object of the `SpatRaster` class can point to one or more files on disk that hold the cell values, and/or it can hold these values in memory. These objects can be created with the `rast` method.

A `SpatRasterDataset` is a collection of sub-datasets, where each is a `SpatRaster` for the same area (extent) and coordinate reference system, but possibly with a different resolution. Sub-datasets are often used to capture variables (e.g. temperature and precipitation), or a fourth dimension (e.g. height, depth or time) if the sub-datasets already have three dimensions (multiple layers).

A `SpatRasterCollection` is a collection of SpatRasters with no restriction in the extent or other geometric parameters.

Examples

```
rast()
```

spatSample *Take a regular sample*

Description

Take a spatial sample from a SpatRaster, SpatVector or SpatExtent. Sampling a SpatVector or SpatExtent always returns a SpatVector of points.

With a SpatRaster, you can get cell values, cell numbers (`cells=TRUE`), coordinates (`xy=TRUE`) or (when `method="regular"` and `as.raster=TRUE`) get a new SpatRaster with the same extent, but fewer cells.

In order to assure regularity when requesting a regular sample, the number of cells or points returned may not be exactly the same as the `size` requested.

Usage

```
## S4 method for signature 'SpatRaster'
spatSample(x, size, method="random", replace=FALSE, na.rm=FALSE,
    as.raster=FALSE, as.df=TRUE, as.points=FALSE, values=TRUE, cells=FALSE,
    xy=FALSE, ext=NULL, warn=TRUE, weights=NULL, exp=5, exhaustive=FALSE)

## S4 method for signature 'SpatVector'
spatSample(x, size, method="random", strata=NULL, chess="")

## S4 method for signature 'SpatExtent'
spatSample(x, size, method="random", lonlat, as.points=FALSE)
```

Arguments

`x`	SpatRaster, SpatVector or SpatExtent
`size`	numeric. The sample size. If `x` is a SpatVector, you can also provide a vector of the same length as `x` in which case sampling is done separately for each geometry. If `x` is a SpatRaster, and you are using `method="regular"` you can specify the size as two numbers (number of rows and columns)
`method`	character. Should be "regular" or "random", If `x` is a `SpatRaster`, it can also be "stratified" (each value in `x` is a stratum) or "weights" (each value in `x` is a probability weight)
`replace`	logical. If `TRUE`, sampling is with replacement (if `method="random"`)
`na.rm`	logical. If `TRUE`, `NA`s are removed. Only used with random sampling of cell values. That is with `method="random"`, `as.raster=FALSE`, `cells=FALSE`
`as.raster`	logical. If `TRUE`, a SpatRaster is returned
`as.df`	logical. If `TRUE`, a data.frame is returned instead of a matrix
`as.points`	logical. If `TRUE`, a SpatVector of points is returned
`values`	logical. If `TRUE` raster cell values are returned

cells	logical. If TRUE, cell numbers are returned. If method="stratified" this is always set to TRUE if xy=FALSE
xy	logical. If TRUE, cell coordinates are returned
ext	SpatExtent or NULL to restrict sampling to a subset of the area of x
warn	logical. Give a warning if the sample size returned is smaller than requested
weights	SpatRaster. Used to provide weights when method="stratified"
strata	if not NULL, stratified random sampling is done, taking size samples from each stratum. If x has polygon geometry, strata must be a field name (or index) in x. If x has point geometry, strata can be a SpatVector of polygons or a SpatRaster
chess	character. One of "", "white", or "black". For stratified sampling if strata is a SpatRaster. If not "", samples are only taken from alternate cells, organized like the "white" or "black" fields on a chessboard
lonlat	logical. If TRUE, sampling of a SpatExtent is weighted by cos(latitude). For SpatRaster and SpatVector this done based on the crs, but it is ignored if as.raster=TRUE
exp	numeric >= 1. "Expansion factor" that is multiplied with size to get an initial sample used for stratified samples and random samples with na.rm=TRUE to try to get at least size samples
exhaustive	logical. If TRUE and na.rm=TRUE first all cells that are not NA are determined and a sample is taked from these cells. This is useful when you are dealing with a very large raster that is sparse (most cells are NA). Otherwise, the default approach may not find enough samples. This should not be used in other cases, especially not with large rasters that mostly have values

Value

numeric matrix, data.frame, SpatRaster or SpatVector

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
s <- spatSample(r, 10, as.raster=TRUE)
spatSample(r, 5)
spatSample(r, 5, na.rm=TRUE)
spatSample(r, 5, "regular")

## if you require cell numbers and/or coordinates
size <- 6
spatSample(r, 6, "random", cells=TRUE, xy=TRUE, values=FALSE)

# regular, with values
spatSample(r, 6, "regular", cells=TRUE, xy=TRUE)

# stratified
rr <- rast(ncol=10, nrow=10, names="stratum")
set.seed(1)
values(rr) <- round(runif(ncell(rr), 1, 3))
```

```
spatSample(rr, 2, "stratified", xy=TRUE)

s <- spatSample(rr, 5, "stratified", as.points=TRUE)
plot(rr, plg=list(title="raster"))
plot(s, 1, add=TRUE, plg=list(x=185, y=1, title="points"))

## SpatExtent
e <- ext(r)
spatSample(e, 10, "random", lonlat=TRUE)

## SpatVector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

# sample the geometries
i <- sample(v, 3)

# sample points in geometries
p <- spatSample(v, 3)
```

`SpatVector-class` *Class "SpatVector"*

Description

`SpatVector` can represent points, lines or polygons.

`SpatVectorCollection` can hold a collection of SpatVectors

`SpatVectorProxy` is a SpatVector for which the data are on-disk in-stead of in memory.

`spin` *spin a SpatVector*

Description

Spin (rotate) the geometry of a SpatVector.

Usage

```
## S4 method for signature 'SpatVector'
spin(x, angle, x0, y0)
```

Arguments

`x`	SpatVector
`angle`	numeric. Angle of rotation in degrees
`x0`	numeric. x-coordinate of the center of rotation. If missing, the center of the extent of `x` is used
`y0`	numeric. y-coordinate of the center of rotation. If missing, the center of the extent of `x` is used

Value

SpatVector

See Also

`rescale`, `t`, `shift`

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- spin(v, 180)
plot(v)
lines(w, col="red")

# lower-right corner as center
e <- as.vector(ext(v))
x <- spin(v, 45, e[1], e[3])
```

---

split *Split a SpatRaster or SpatVector*

Description

Split a SpatRaster by layer, or a SpatVector by attributes. You can also split the geometry of a polygon SpatVector with another SpatVector.

Usage

```
## S4 method for signature 'SpatRaster,ANY'
split(x, f)

## S4 method for signature 'SpatVector,ANY'
split(x, f)

## S4 method for signature 'SpatVector,SpatVector'
split(x, f)
```

Arguments

x	SpatRaster or SpatVector
f	If x is a SpatRaster: a vector of the length nlyr(x). If x is a SpatVector: a field (variable) name or a vector of the same length as x; or, if x is a SpatVector of polygons, a SpatVector of lines or polygons to split the polygon geometries

Value

list or SpatVector

Examples

```
## split layers
s <- rast(system.file("ex/logo.tif", package="terra"))
y <- split(s, c(1,2,1))
sds(y)

## split attributes
v <- vect(system.file("ex/lux.shp", package="terra"))
x <- split(v, "NAME_1")

## split geometries
v <- v[1:5,]
line <- vect(matrix(c(5.79, 6.22, 5.75, 6.1, 5.8,
50.14, 50.05, 49.88, 49.85, 49.71), ncol=2), "line")
s <- split(v, line)
```

---

sprc *Create a SpatRasterCollection*

Description

Methods to create a SpatRasterCollection. This is an object to hold a collection (list) of SpatRasters. There are no restrictions on the similarity of the SpatRaster geometry.

They can be used to combine several SpatRasters to be used with merge or mosaic

You can create a SpatRasterCollection from a file with subdatasets.

Usage

```
## S4 method for signature 'character'
sprc(x, ids=0, opts=NULL, raw=FALSE)

## S4 method for signature 'SpatRaster'
sprc(x, ...)

## S4 method for signature 'list'
sprc(x)
```

```
## S4 method for signature 'missing'
sprc(x)
```

Arguments

x	SpatRaster, list with SpatRasters, missing, or filename
ids	optional. vector of integer subdataset ids. Ignored if the first value is not a positive integer
opts	character. GDAL dataset open options
raw	logical. If TRUE, scale and offset values are ignored
...	additional SpatRasters

Value

SpatRasterCollection

See Also

sds

Examples

```
x <- rast(xmin=-110, xmax=-50, ymin=40, ymax=70, ncols=60, nrows=30)
y <- rast(xmin=-80, xmax=-20, ymax=60, ymin=30)
res(y) <- res(x)
values(x) <- 1:ncell(x)
values(y) <- 1:ncell(y)

z <- sprc(x, y)
z
```

stretch *Stretch*

Description

Linear or histogram equalization stretch of values in a SpatRaster.

For linear stretch, provide the desired output range (minv and maxv) and the lower and upper bounds in the original data, either as quantiles (minq and maxq, or as cell values (smin and smax). If smin and smax are both not NA, minq and maxq are ignored.

For histogram equalization, these arguments are ignored, but you can provide the desired scale of the output and the maximum number of cells that is used to compute the histogram (empirical cumulative distribution function).

Usage

```
## S4 method for signature 'SpatRaster'
stretch(x, minv=0, maxv=255, minq=0, maxq=1, smin=NA, smax=NA,
histeq=FALSE, scale=1, maxcell=500000, filename="", ...)
```

Arguments

`x`	SpatRaster
`minv`	numeric >= 0 and smaller than maxv. lower bound of stretched value
`maxv`	numeric <= 255 and larger than maxv. upper bound of stretched value
`minq`	numeric >= 0 and smaller than maxq. lower quantile bound of original value. Ignored if smin is supplied
`maxq`	numeric <= 1 and larger than minq. upper quantile bound of original value. Ignored if smax is supplied
`smin`	numeric < smax. user supplied lower value for the layers, to be used instead of a quantile computed by the function itself
`smax`	numeric > smin. user supplied upper value for the layers, to be used instead of a quantile computed by the function itself
`histeq`	logical. If `TRUE` histogram equalization is used instead of linear stretch
`scale`	numeric. The scale (maximum value) of the output if `histeq=TRUE`
`maxcell`	positive integer. The size of the regular sample used to compute the histogram
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

Examples

```
r <- rast(nc=10, nr=10)
values(r) <- rep(1:25, 4)
rs <- stretch(r)
s <- c(r, r*2)
sr <- stretch(s)
```

subset *Subset a SpatRaster or a SpatVector*

Description

Select a subset of layers from a SpatRaster or select a subset of records (row) and/or variables (columns) from a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster'
subset(x, subset, negate=FALSE, NSE=FALSE, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatVector'
subset(x, subset, select, drop=FALSE, NSE=FALSE)
```

Arguments

`x`	SpatRaster or SpatVector
`subset`	if `x` is a `SpatRaster`: integer or character to select layers if `x` is a `SpatVector`: logical expression indicating the rows to keep (missing values are taken as FALSE)
`select`	expression, indicating columns to select
`negate`	logical. If `TRUE` all layers that are **not** in the subset are selected
`NSE`	logical. If `TRUE`, non-standard evaluation (the use of unquoted variable names) is allowed. Set this to `FALSE` when calling `subset` from a function
`drop`	logical. If `TRUE`, the geometries will be dropped, and a data.frame is returned
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional arguments for writing files as in `writeRaster`

Value

if `x` is a `SpatRaster`: SpatRaster

if `x` is a `SpatVector`: SpatVector or, if `drop=TRUE`, a `data.frame`.

Examples

```
### SpatRaster
s <- rast(system.file("ex/logo.tif", package="terra"))
subset(s, 2:3)
subset(s, c(3,2,3,1))

#equivalent to
s[[ c(3,2,3,1) ]]
```

```
s[[c("red", "green")]]
s$red

# expression based (partial) matching of names with single brackets
s["re"]
s["^re"]

# not with double brackets
# s[["re"]]


### SpatVector

v <- vect(system.file("ex/lux.shp", package="terra"))

subset(v, v$NAME_1 == "Diekirch", c("NAME_1", "NAME_2"))

subset(v, NAME_1 == "Diekirch", c(NAME_1, NAME_2), NSE=TRUE)

# or like this
v[2:3,]
v[1:2, 2:3]
v[1:2, c("NAME_1", "NAME_2")]
```

---

subset_dollar *Subset a SpatRaster or a SpatVector*

Description

Select a subset of layers from a SpatRaster or select a subset of records (row) and/or variables (columns) from a SpatVector.

Usage

```
## S4 method for signature 'SpatExtent'
x$name
```

Arguments

x	SpatRaster, SpatVector or SpatExtent
name	character. If x is a SpatRaster: layer name. If x is a SpatVector: variable name. If x is a SpatExtent: xmin, xmax, ymin or ymax

Value

if x is a SpatRaster: SpatRaster

if x is a SpatVector: SpatVector or, if drop=TRUE, a data.frame.

See Also

subset, [, [[, extract

Examples

```
### SpatRaster
s <- rast(system.file("ex/logo.tif", package="terra"))
subset(s, 2:3)
subset(s, c(3,2,3,1))
#equivalent to
s[[ c(3,2,3,1) ]]

s[[c("red", "green")]]
s$red

# expression based (partial) matching of names with single brackets
s["re"]
s["^re"]

# not with double brackets
# s[["re"]]


### SpatVector

v <- vect(system.file("ex/lux.shp", package="terra"))
v[2:3,]
v[1:2, 2:3]

subset(v, v$NAME_1 == "Diekirch", c("NAME_1", "NAME_2"))

subset(v, NAME_1 == "Diekirch", c(NAME_1, NAME_2), NSE=TRUE)
```

---

subset_double *Subset a SpatRaster or a SpatVector*

Description

Select a subset of layers from a SpatRaster or select a subset of records (row) and/or variables (columns) from a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster,numeric,missing'
x[[i, j]]

## S4 method for signature 'SpatRasterDataset,ANY,ANY'
x[[i, j, drop=TRUE]]
```

```
## S4 method for signature 'SpatVector,numeric,missing'
x[[i, j, drop=FALSE]]
```

Arguments

x	SpatRaster or SpatVector
i	if x is a SpatRaster: integer, logical, or character to select layers if x is a SpatVector: integer, logical, or character to select variables
j	missing, or, for SpatRasterDataset only, numeric
drop	logical. If TRUE, the geometries will be dropped, and a data.frame is returned

Value

if x is a SpatRaster or SpatRasterDataset: SpatRaster

if x is a SpatVector: a data.frame.

See Also

subset, $, [, extract

Examples

```
### SpatRaster
s <- rast(system.file("ex/logo.tif", package="terra"))
s[[ 1:2 ]]

s[[c("red", "green")]]

# expression based (partial) matching of names with single brackets
s["re"]
s["^re"]

# does not with double brackets
# s[["re"]]


### SpatVector

v <- vect(system.file("ex/lux.shp", package="terra"))
v[[2:3]]

# to keep the geometry use
v[,2:3]
```

subset_single *Extract values from a SpatRaster, SpatVector or SpatExtent*

Description

Extract values from a SpatRaster; a subset of records (row) and/or variables (columns) from a SpatVector; or a number from a SpatExtent.

You can use indices (row, column, layer or cell numbers) to extract. You can also use other Spat* objects.

Usage

```
## S4 method for signature 'SpatRaster,ANY,ANY,ANY'
x[i, j, k]

## S4 method for signature 'SpatVector,numeric,numeric'
x[i, j, drop=FALSE]

## S4 method for signature 'SpatVector,SpatVector,missing'
x[i, j]

## S4 method for signature 'SpatExtent,numeric,missing'
x[i, j]
```

Arguments

x	SpatRaster, SpatVector or SpatExtent
i	if x is a SpatRaster: numeric, logical or missing to select rows or, if j is missing, to select cells numbers. if x is a SpatVector: numeric or missing to select rows. if i is another SpatVector: get a new SpatVector with the geometries that intersect. if x is a SpatExtent: integer between 1 and 4.
j	numeric, logical, or missing to select columns
k	numeric, character, or missing to select layers
drop	logical. If FALSE an object of the same class as x is returned

Value

numeric if x is a SpatExtent. Same as x if drop=FALSE. Otherwise a data.frame

See Also

extract, subset, $, [[

Examples

```
### SpatRaster
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
r[3638]
rowColFromCell(r, 2638)
r[39, 28]
x <- r[39:40, 28:29, drop=FALSE]
as.matrix(x, wide=TRUE)

### SpatVector

v <- vect(system.file("ex/lux.shp", package="terra"))
v[2:3,]
v[1:2, 2:3]
v[1:2, 2:3, drop=TRUE]
```

`subst` *replace cell values*

Description

Substitute(replace) cell values of a SpatRaster with a new value. See `classify` for more complex/flexible replacement.

Usage

```
## S4 method for signature 'SpatRaster'
subst(x, from, to, others=NULL, raw=FALSE, filename="", ...)
```

Arguments

`x`	SpatRaster
`from`	numeric value(s). Normally a vector of the same length as 'to'. If `x` has multiple layers, it can also be a matrix of numeric value(s) where `nrow(x) == length(to)`. In that case the output has a single layer, with values based on the combination of the values of the input layers
`to`	numeric value(s). Normally a vector of the same length as 'from'. If `x` has a single layer, it can also be a matrix of numeric value(s) where `nrow(x) == length(from)`. In that case the output has multiple layers, one for each column in `to`
`others`	numeric. If not `NULL` all values that are not matched are set to this value. Otherwise they retain their original value.
`raw`	logical. If `TRUE`, the values in from and to are the raw cell values, not the categorical labels. Only relevant if `is.factor(x)`
`filename`	character. Output filename
`...`	Additional arguments for writing files as in `writeRaster`

Value

SpatRaster

See Also

classify, clamp

Examples

```
r <- rast(ncols=5, nrows=5, xmin=0, xmax=1, ymin=0, ymax=1, crs="")
r <- init(r, 1:6)
x <- subst(r, 3, 7)
x <- subst(r, 2:3, NA)
x <- subst(x, NA, 10)

# multiple output layers
z <- subst(r, 2:3, cbind(20,30))

# multiple input layers
rr <- c(r, r+1, r+2)
m <- rbind(c(1:3), c(3:5))
zz <- subst(rr, m, c(100, 200))
```

---

summarize *Summarize*

Description

Compute summary statistics for cells, either across layers or between layers (parallel summary).

The following summary methods are available for SpatRaster: `any`, `anyNA`, `all`, `allNA`, `max`, `min`, `mean`, `median`, `prod`, `range`, `stdev`, `sum`, `which.min`, `which.max`. See `modal` to compute the mode and `app` to compute summary statistics that are not included here.

Because generic functions are used, the method applied is chosen based on the first argument: "`x`". This means that if `r` is a SpatRaster, `mean(r, 5)` will work, but `mean(5, r)` will not work.

The `mean` method has an argument "trim" that is ignored.

If pop=`TRUE` `stdev` computes the population standard deviation, computed as:

`f <- function(x) sqrt(sum((x-mean(x))^2) / length(x))`

This is different than the sample standard deviation returned by `sd` (which uses `n-1` as denominator).

Usage

```
## S4 method for signature 'SpatRaster'
min(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
max(x, ..., na.rm=FALSE)
```

```
## S4 method for signature 'SpatRaster'
range(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
prod(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
sum(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
any(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
all(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
range(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
which.min(x)

## S4 method for signature 'SpatRaster'
which.max(x)

## S4 method for signature 'SpatRaster'
stdev(x, ..., pop=TRUE, na.rm=FALSE)

## S4 method for signature 'SpatRaster'
mean(x, ..., trim=NA, na.rm=FALSE)

## S4 method for signature 'SpatRaster'
median(x, na.rm=FALSE, ...)

## S4 method for signature 'SpatRaster'
anyNA(x)

## S4 method for signature 'SpatRaster'
countNA(x, n=0)

## S4 method for signature 'SpatRaster'
noNA(x, falseNA=FALSE)

## S4 method for signature 'SpatRaster'
allNA(x, falseNA=FALSE)
```

Arguments

x	SpatRaster

`...`	additional SpatRasters or numeric values; and arguments `par` for parallel summarization (see Details), and `filename`, `overwrite` and `wopt` as for `writeRaster`
`na.rm`	logical. If `TRUE`, `NA` values are ignored. If `FALSE`, `NA` is returned if `x` has any `NA` values
`trim`	ignored
`pop`	logical. If `TRUE`, the population standard deviation is computed. Otherwise the sample standard deviation is computed
`falseNA`	logical. If `TRUE`, cells that would otherwise be `FALSE` are set to `NA`
`n`	integer. If `n > 0`, cell values are `TRUE` if at least `n` of its layers are `NA`

Details

Additional argument `par` can be used for "parallel" summarizing a SpatRaster and a numeric or logical value. If a SpatRaster `x` has three layers, `max(x, 5)` will return a single layer (the number five is treated as a layer in which all cells have value five). In contrast `max(x, 5, par=TRUE)` returns three layers (the number five is treated as another SpatRaster with a single layer with all cells having the value five.

Value

SpatRaster

See Also

`app`, `Math-methods`, `modal`, `which.lyr`

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10, nlyrs=3)
values(r) <- runif(ncell(r) * nlyr(r))

x <- mean(r)
# note how this returns one layer
x <- sum(c(r, r[[2]]), 5)

# and this returns three layers
y <- sum(r, r[[2]], 5)

max(r)

## when adding a number, do you want 1 layer or all layers?
# 1 layer
max(r, 0.5)

# all layers
max(r, 0.5, par=TRUE)

y <- stdev(r)
# not the same as
```

```
yy <- app(r, sd)

z <- stdev(r, r*2)

x <- mean(r, filename=paste0(tempfile(), ".tif"))


v <- values(r)
set.seed(3)
v[sample(length(v), 50)] <- NA
values(r) <- v
is.na(r)
anyNA(r)
allNA(r)
countNA(r)
countNA(r, 2)
```

---

summary *summary*

Description

Compute summary statistics (min, max, mean, and quartiles) for SpatRaster using base summary method. A sample is used for very large files.

For single or other statistics see Summary-methods, global, and quantile

Usage

```
## S4 method for signature 'SpatRaster'
summary(object, size=100000, warn=TRUE, ...)

## S4 method for signature 'SpatVector'
summary(object, ...)
```

Arguments

object	SpatRaster or SpatVector
size	positive integer. Size of a regular sample used for large datasets (see spatSample)
warn	logical. If TRUE a warning is given if a sample is used
...	additional arguments passed on to the base summary method

Value

matrix with (an estimate of) the median, minimum and maximum values, the first and third quartiles, and the number of cells with NA values

See Also

Summary-methods, global, quantile

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10, nlyrs=3)
values(r) <- runif(nlyr(r)*ncell(r))
summary(r)
```

---

surfArea *Compute surface area from elevation data*

Description

It is often said that if Wales was flattened out it would have an area bigger than England. This function computes the surface area for a raster with elevation values, taking into account the sloping nature of the surface.

Usage

```
## S4 method for signature 'SpatRaster'
surfArea(x, filename="", ...)
```

Arguments

x	SpatRaster with elevation values. Currently the raster CRS must be planar and have the same distance units (e.g. m) as the elevation values
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

Author(s)

Barry Rowlingson <b.rowlingson@lancaster.ac.uk>

References

Jenness, Jeff S., 2004. Calculating Landscape Surface Area from Digital Elevation Models. Wildlife Society Bulletin 32(3): 829-839

Examples

```
v <- rast(volcano, crs="local")
x <- terra::surfArea(v)
```

svc *Create a SpatVectorCollection*

Description

Methods to create a SpatVectorCollection. This is an object to hold "sub-datasets", each a SpatVector, perhaps of different geometry type.

Usage

```
## S4 method for signature 'missing'
svc(x)

## S4 method for signature 'SpatVector'
svc(x, ...)

## S4 method for signature 'list'
svc(x)

## S4 method for signature 'character'
svc(x, layer="", query="", extent=NULL, filter=NULL)
```

Arguments

`x`	SpatVector, character (filename), list with SpatVectors, or missing
`...`	Additional `SpatVectors`
`layer`	character. layer name to select a layer from a file (database) with multiple layers
`query`	character. A query to subset the dataset in the OGR-SQL dialect
`extent`	Spat* object. The extent of the object is used as a spatial filter to select the geometries to read. Ignored if `filter` is not `NULL`
`filter`	SpatVector. Used as a spatial filter to select geometries to read (the convex hull is used for lines or points). It is guaranteed that all features that overlap with the extent of filter will be returned. It can happen that additional geometries are returned

Value

SpatVectorCollection

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
x <- svc()
x <- svc(v, v[1:3,], as.lines(v[3:5,]), as.points(v))
length(x)
```

```
x

# extract
x[3]

# replace
x[2] <- as.lines(v[1,])
```

---

symdif *Symmetrical difference*

Description

Symmetrical difference of polygons

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
symdif(x, y)
```

Arguments

x	SpatVector
y	SpatVector

Value

SpatVector

See Also

erase

Examples

```
p <- vect(system.file("ex/lux.shp", package="terra"))
b <- as.polygons(ext(6, 6.4, 49.75, 50))
#sd <- symdif(p, b)
#plot(sd, col=rainbow(12))
```

tapp — *Apply a function to subsets of layers of a SpatRaster*

Description

Apply a function to subsets of layers of a SpatRaster (similar to `tapply` and `aggregate`). The layers are combined based on the `index`.

The number of layers in the output SpatRaster equals the number of unique values in `index` times the number of values that the supplied function returns for a single vector of numbers.

For example, if you have a SpatRaster with 6 layers, you can use `index=c(1,1,1,2,2,2)` and `fun=sum`. This will return a SpatRaster with two layers. The first layer is the sum of the first three layers in the input SpatRaster, and the second layer is the sum of the last three layers in the input SpatRaster. Indices are recycled such that `index=c(1,2)` would also return a SpatRaster with two layers (one based on the odd layers (1,3,5), the other based on the even layers (2,4,6)).

The index can also be one of the following values to group by time period (if `x` has the appropriate `time` values): "years", "months", "yearmonths", "dekads", "yeardekads", "weeks" (the ISO 8601 week number, see Details), "yearweeks", "days", "doy" (day of the year), "7days" (seven-day periods starting at Jan 1 of each year), "10days", or "15days". It can also be a function that makes groups from time values.

See `app` or `Summary-methods` if you want to use a more efficient function that returns multiple layers based on **all** layers in the SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
tapp(x, index, fun, ..., cores=1, filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`index`	factor or numeric (integer). Vector of length `nlyr(x)` (shorter vectors are recycled) grouping the input layers. It can also be one of the following values: "years", "months", "yearmonths", "days", "week" (ISO 8601 week number), or "doy" (day of the year)
`fun`	function to be applied. The following functions have been re-implemented in C++ for speed: "sum", "mean", "median", "modal", "which", "which.min", "which.max", "min", "max", "prod", "any", "all", "sd", "std", "first". To use the base-R function for say, "min", you could use something like `fun = \(i) min(i)`
`...`	additional arguments passed to `fun`
`cores`	positive integer. If `cores > 1`, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under fun)
`filename`	character. Output filename

`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Details

"week" follows the ISO 8601 definition. Weeks start on Monday. If the week containing 1 January has four or more days in the new year, then it is considered week "01". Otherwise, it is the last week of the previous year (week "52" or "53", and the next week is week 1.

Value

SpatRaster

See Also

`app`, `Summary-methods`

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
s <- c(r, r, r, r, r, r)
s <- s * 1:6
b1 <- tapp(s, index=c(1,1,1,2,2,2), fun=sum)
b1
b2 <- tapp(s, c(1,2,3,1,2,3), fun=sum)
b2
```

`terrain` *terrain characteristics*

Description

Compute terrain characteristics from elevation data. The elevation values should be in the same units as the map units (typically meter) for projected (planar) raster data. They should be in meter when the coordinate reference system is longitude/latitude.

For accuracy, always compute these values on the original data (do not first change the projection). Distances (needed for slope and aspect) for longitude/latitude data are computed on the WGS84 ellipsoid with Karney's algorithm.

Usage

```
## S4 method for signature 'SpatRaster'
terrain(x, v="slope", neighbors=8, unit="degrees", filename="", ...)
```

Arguments

`x`	SpatRaster, single layer with elevation values. Values should have the same unit as the map units, or in meters when the crs is longitude/latitude
`v`	character. One or more of these options: slope, aspect, TPI, TRI, TRIriley, TRIrmsd, roughness, flowdir (see Details)
`unit`	character. "degrees" or "radians" for the output of "slope" and "aspect"
`neighbors`	integer. Indicating how many neighboring cells to use to compute slope or aspect with. Either 8 (queen case) or 4 (rook case)
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Details

When `neighbors=4`, slope and aspect are computed according to Fleming and Hoffer (1979) and Ritter (1987). When `neighbors=8`, slope and aspect are computed according to Horn (1981). The Horn algorithm may be best for rough surfaces, and the Fleming and Hoffer algorithm may be better for smoother surfaces (Jones, 1997; Burrough and McDonnell, 1998).

If slope = 0, aspect is set to 0.5*pi radians (or 90 degrees if unit="degrees"). When computing slope or aspect, the coordinate reference system of `x` must be known for the algorithm to differentiate between planar and longitude/latitude data.

`terrain` is not vectorized over "neighbors" or "unit" – only the first value is used.

flowdir returns the "flow direction" (of water), that is the direction of the greatest drop in elevation (or the smallest rise if all neighbors are higher). They are encoded as powers of 2 (0 to 7). The cell to the right of the focal cell is 1, the one below that is 2, and so on:

32	64	128
16	x	1
8	4	2

Cells without lower neighboring cells are encoded as zero.

If two cells have the same drop in elevation, a random cell is picked. That is not ideal as it may prevent the creation of connected flow networks. ArcGIS implements the approach of Greenlee (1987) and I might adopt that in the future.

Most terrain indices are according to Wilson et al. (2007), as in gdaldem. TRI (Terrain Ruggedness Index) is the mean of the absolute differences between the value of a cell and its 8 surrounding cells. TPI (Topographic Position Index) is the difference between the value of a cell and the mean value of its 8 surrounding cells. Roughness is the difference between the maximum and the minimum value of a cell and its 8 surrounding cells.

TRIriley (TRI according to Riley et al., 2007) returns the square root of summed squared differences between the value of a cell and its 8 surrounding cells. TRIrmsd computes the square root of the mean of the squared differences between these cells.

These measures can also be computed with `focal` functions:

TRI <- focal(x, w=3, fun=\(x) sum(abs(x[-5]-x[5]))/8)

TPI <- focal(x, w=3, fun=\(x) x[5] - mean(x[-5]))

rough <- focal(x, w=3, fun=\(x) max(x) - min(x))

References

Burrough, P., and R.A. McDonnell, 1998. Principles of Geographical Information Systems. Oxford University Press.

Fleming, M.D. and Hoffer, R.M., 1979. Machine processing of Landsat MSS data and DMA topographic data for forest cover type mapping. LARS Technical Report 062879. Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana.

Horn, B.K.P., 1981. Hill shading and the reflectance map. Proceedings of the IEEE 69:14-47

Jones, K.H., 1998. A comparison of algorithms used to compute hill slope as a property of the DEM. Computers & Geosciences 24: 315-323

Karney, C.F.F., 2013. Algorithms for geodesics, J. Geodesy 87: 43-55. doi:10.1007/s00190-012-0578-z.

Riley, S.J., De Gloria, S.D., Elliot, R. (1999): A Terrain Ruggedness that Quantifies Topographic Heterogeneity. Intermountain Journal of Science 5: 23-27.

Ritter, P., 1987. A vector-based terrain and aspect generation algorithm. Photogrammetric Engineering and Remote Sensing 53: 1109-1111

Wilson et al 2007, Multiscale Terrain Analysis of Multibeam Bathymetry Data for Habitat Mapping on the Continental Slope. Marine Geodesy 30:3-35

See Also

viewshed

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- terrain(r, "slope")
```

---

text *Add labels to a map*

Description

Plots labels, that is a textual (rather than color) representation of values, on top an existing plot (map).

Usage

```
## S4 method for signature 'SpatRaster'
text(x, labels, digits=0, halo=FALSE, ...)

## S4 method for signature 'SpatVector'
text(x, labels, halo=FALSE, inside=FALSE, ...)
```

Arguments

x	SpatRaster or SpatVector
labels	character. Optional. Vector of labels with `length(x)` or a variable name from `names(x)`
digits	integer. How many digits should be used?
halo	logical. If `TRUE` a "halo" is printed around the text. If `TRUE`, additional arguments `hc="white"` and `hw=0.1` can be modified to set the color and width of the halo
inside	logical. Should the text always be placed inside one the sub-geometries?
...	additional arguments to pass to graphics function `text`

See Also

`text`, `plot`, `halo`

Examples

```
r <- rast(nrows=4, ncols=4)
values(r) <- 1:ncell(r)
plot(r)
text(r)

plot(r)
text(r, halo=TRUE, hc="blue", col="white", hw=0.2)

plot(r, col=rainbow(16))
text(r, col=c("black", "white"), vfont=c("sans serif", "bold"), cex=2)
```

---

tighten — *tighten SpatRaster or SpatRasterDataset objects*

Description

Combines data sources within a SpatRaster (that are in memory, or from the same file) to allow for faster processing.

Or combine sub-datasets into a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
tighten(x)

## S4 method for signature 'SpatRasterDataset'
tighten(x)
```

Arguments

x	SpatRaster or SpatRasterDataset

Value

SpatRaster

Examples

```
r <- rast(nrow=5, ncol=9, vals=1:45)
x <- c(r, r*2, r*3)
x
tighten(x)
```

---

time — *time of SpatRaster layers*

Description

Get or set the time of the layers of a SpatRaster. Time can be stored as `POSIXlt` (date and time, with a resolution of seconds, and a time zone), `Date`, "months", "years", or "yearmonths".

`timeInfo` and `has.time` are helper functions to understand what a time data a SpatRaster has.

Usage

```
## S4 method for signature 'SpatRaster'
has.time(x)

## S4 method for signature 'SpatRaster'
time(x, format="")

## S4 replacement method for signature 'SpatRaster'
time(x, tstep="")<-value

## S4 method for signature 'SpatRaster'
timeInfo(x)
```

Arguments

`x`	SpatRaster or SpatRasterDataset
`format`	One of "", "seconds" (POSIXlt), "days" (Date), "yearmonths" (decimal years), "years", "months". If "", the returned format is (based on) the format that was used to set the time
`value`	`Date`, `POSIXt`, `yearmon` (defined in package zoo), or numeric
`tstep`	One of "years", "months", "yearmonths". Used when `value` is numeric. Ignored when `value` is of type `Date`, `POSIXt`, or `yearmon`

Value

`time`: POSIXlt, Date, or numeric `timeInfo`: data.frame with time step and time zone information (if available) `has.time`: logical

See Also

depth

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))

# Date"
d <- as.Date("2001-05-04") + 0:2
time(s) <- d
time(s)

# POSIX (date/time with a resolution of seconds)
time(s) <- as.POSIXlt(d)
time(s)

# with time zone
time(s) <- as.POSIXlt(Sys.time(), "America/New_York") + 0:2
time(s)
timeInfo(s)

# years
time(s, tstep="years") <- 2000 + 0:2
s

time(s, tstep="months") <- 1:3
s
```

---

tmpFiles *Temporary files*

Description

List and optionally remove temporary files created by the terra package. These files are created when an output SpatRaster may be too large to store in memory (RAM). This can happen when no filename is provided to a function and when using functions where you cannot provide a filename.

Temporary files are automatically removed at the end of each R session that ends normally. You can use `tmpFiles` to see the files in the current sessions, including those that are orphaned (not connect to a SpatRaster object any more) and from other (perhaps old) sessions, and remove all the temporary files.

Usage

```
tmpFiles(current=TRUE, orphan=FALSE, old=FALSE, remove=FALSE)
```

Arguments

`current`	logical. If `TRUE`, temporary files from the current R session are included
`orphan`	logical. If `TRUE`, temporary files from the current R session that are no longer associated with a SpatRaster (if `current` is `TRUE` these are also included)
`old`	logical. If `TRUE`, temporary files from other "R" sessions. Unless you are running multiple instances of R at the same time, these are from old (possibly crashed) R sessions and should be removed
`remove`	logical. If `TRUE`, temporary files are removed

Value

character

See Also

`terraOptions`

Examples

```
tmpFiles()
```

---

toMemory — *Read all cell values into memory*

Description

Reads all cell values of a SpatRaster or SpatRasterDataset into memory.

Using this method is discouraged as it is not necessary for processing the data and may lead to excessive memory use that will slow down your computer or worse. It cannot be used for SpatRasters that are based on very large files.

The method may be useful if a relatively small dataset is used repeatedly, such that efficiency gains are made because the values only need to be read from disk once.

Usage

```
## S4 method for signature 'SpatRaster'
toMemory(x)

## S4 method for signature 'SpatRasterDataset'
toMemory(x)
```

Arguments

`x`	SpatRaster or SpatRasterDataset

Value

Same as x

See Also

values, as.data.frame, readValues

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
sources(r)
inMemory(r)
x <- toMemory(r)
inMemory(x)
```

---

topology *Vector topology methods*

Description

makeNodes create nodes on lines

mergeLines connect lines to form polygons

removeDupNodes removes duplicate nodes in geometries and optionally rounds the coordinates

emptyGeoms returns the indices of empty (null) geometries. is.na also checks if any of the coordinates is NA.

snap makes boundaries of geometries identical if they are very close to each other.

Usage

```
## S4 method for signature 'SpatVector'
mergeLines(x)
## S4 method for signature 'SpatVector'
snap(x, y=NULL, tolerance)
## S4 method for signature 'SpatVector'
removeDupNodes(x, digits = -1)
## S4 method for signature 'SpatVector'
makeNodes(x)
```

Arguments

x	SpatVector of lines or polygons
y	SpatVector of lines or polygons to snap to. If NULL snapping is to the other geometries in x
tolerance	numeric. Snapping tolerance (distance between geometries)
digits	numeric. Number of digits used in rounding. Ignored if < 0

Value

SpatVector

See Also

sharedPaths, gaps, simplifyGeom, forceCCW

Examples

```
p1 <- as.polygons(ext(0,1,0,1))
p2 <- as.polygons(ext(1.1,2,0,1))

p <- rbind(p1, p2)

y <- snap(p, tol=.15)
plot(p, lwd=3, col="light gray")
lines(y, col="red", lwd=2)
```

---

transpose *Transpose*

Description

Transpose a SpatRaster or SpatVector

Usage

```
## S4 method for signature 'SpatRaster'
t(x)

## S4 method for signature 'SpatVector'
t(x)

## S4 method for signature 'SpatRaster'
trans(x, filename="", ...)
```

Arguments

x	SpatRaster or SpatVector
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

flip, rotate

Examples

```
r <- rast(nrows=18, ncols=36)
values(r) <- 1:ncell(r)
tr1 <- t(r)
tr2 <- trans(r)
ttr <- trans(tr2)
```

---

trim *Trim a SpatRaster*

Description

Trim (shrink) a SpatRaster by removing outer rows and columns that are NA or another value.

Usage

```
## S4 method for signature 'SpatRaster'
trim(x, padding=0, value=NA, filename="", ...)
```

Arguments

x	SpatRaster
padding	integer. Number of outer rows/columns to keep
value	numeric. The value of outer rows or columns that are to be removed
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

Examples

```
r <- rast(ncols=10, nrows=10, xmin=0,xmax=10,ymin=0,ymax=10)
v <- rep(NA, ncell(r))
v[c(12,34,69)] <- 1:3
values(r) <- v
s <- trim(r)
```

union *Union SpatVector or SpatExtent objects*

Description

If you want to append polygon SpatVectors use `rbind` instead of `union`. `union` will also intersect overlapping polygons between, not within, objects. Union for lines and points simply combines the two data sets; without any geometric intersections. This is equivalent to `rbind`. Attributes are joined.

If `x` and `y` have a different geometry type, a SpatVectorCollection is returned.

If a single SpatVector is supplied, overlapping polygons are intersected. Original attributes are lost. New attributes allow for determining how many, and which, polygons overlapped.

SpatExtent: Objects are combined into their union; this is equivalent to `+`.

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
union(x, y)

## S4 method for signature 'SpatVector,missing'
union(x, y)

## S4 method for signature 'SpatExtent,SpatExtent'
union(x, y)
```

Arguments

`x`	SpatVector or SpatExtent
`y`	Same as `x` or missing

Value

SpatVector or SpatExtent

See Also

`rbind`

`intersect`

`merge` and `mosaic` to union SpatRasters.

`crop` and `extend` for the union of SpatRaster and SpatExtent.

`merge` for merging a data.frame with attributes of a SpatVector.

`aggregate` to dissolve SpatVector objects.

Examples

```
e1 <- ext(-10, 10, -20, 20)
e2 <- ext(0, 20, -40, 5)
union(e1, e2)

#SpatVector
v <- vect(system.file("ex/lux.shp", package="terra"))
v <- v[,3:4]
p <- vect(c("POLYGON ((5.8 49.8, 6 49.9, 6.15 49.8, 6 49.65, 5.8 49.8))",
"POLYGON ((6.3 49.9, 6.2 49.7, 6.3 49.6, 6.5 49.8, 6.3 49.9))"), crs=crs(v))
values(p) <- data.frame(pid=1:2, value=expanse(p))
u <- union(v, p)
plot(u, "pid")

b <- buffer(v, 1000)

u <- union(b)
u$sum <- rowSums(as.data.frame(u))
plot(u, "sum")
```

`unique` *Unique values*

Description

This method returns the unique values in a SpatRaster, or removes duplicates records (geometry and attributes) in a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster'
unique(x, incomparables=FALSE, digits=NA, na.rm=TRUE, as.raster=FALSE)

## S4 method for signature 'SpatVector'
unique(x, incomparables=FALSE, ...)
```

Arguments

`x`	SpatRaster or SpatVector
`incomparables`	logical. If `FALSE` and `x` is a SpatRaster: the unique values are determined for all layers together, and the result is a matrix. If `TRUE`, each layer is evaluated separately, and a list is returned. If `x` is a SpatVector this argument is as for a `data.frame`
`digits`	integer. The number of digits for rounding the values before finding the unique values. Use `NA` means to not do any rounding
`na.rm`	logical. If `TRUE`, `NaN` is included if there are any missing values
`as.raster`	logical. If `TRUE`, a single-layer categorical SpatRaster with the unique values is returned
`...`	additional arguments passed on to `unique`

Value

If x is a SpatRaster: data.frame or list (if incomparables=FALSE)

If x is a SpatVector: SpatVector

Examples

```
r <- rast(ncols=5, nrows=5)
values(r) <- rep(1:5, each=5)
unique(r)
s <- c(r, round(r/3))
unique(s)
unique(s,TRUE)

unique(s, as.raster=TRUE)


v <- vect(cbind(x=c(1:5,1:5), y=c(5:1,5:1)),
crs="+proj=utm +zone=1 +datum=WGS84")
nrow(v)
u <- unique(v)
nrow(u)

values(v) <- c(1:5, 1:3, 5:4)
unique(v)
```

---

units *units of SpatRaster or SpatRasterDataSet*

Description

Get or set the units of the layers of a SpatRaster or the datasets in a SpatRasterDataSet.

Usage

```
## S4 method for signature 'SpatRaster'
units(x)

## S4 replacement method for signature 'SpatRaster'
units(x)<-value

## S4 method for signature 'SpatRasterDataset'
units(x)

## S4 replacement method for signature 'SpatRasterDataset'
units(x)<-value
```

Arguments

`x`	SpatRaster
`value`	character

Value

character

See Also

`time`, `names`

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))

units(s) <- c("m/s", "kg", "ha")
units(s)
s

units(s) <- "kg"
units(s)
```

`update` *Change values in a file*

Description

Change the contents of a file that is the data source of a SpatRaster. BE CAREFUL as you are overwriting values in an existing file.

Usage

```
## S4 method for signature 'SpatRaster'
update(object, names=FALSE, crs=FALSE, extent=FALSE)
```

Arguments

`object`	SpatRaster
`names`	logical. Should the names be updated?
`crs`	logical. Should the coordinate reference system be updated?
`extent`	logical. Should the extent be updated?

Value

SpatRaster (invisibly)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
fname <- paste0(tempfile(), ".tif")
x <- writeRaster(s, fname)
names(x) <- c("A", "B", "C")
ext(x) <- ext(x) + 1
crs(x) <- "+proj=utm +zone=1"

update(x, names=TRUE, crs=TRUE, extent=TRUE)

rast(fname)
```

---

values *Cell values and geometry attributes*

Description

Get the cell values of a SpatRaster or the attributes of a SpatVector.

By default all values returned are numeric. This is because a vector or matrix can only store one data type, and a SpatRaster may consist of multiple data types. However, if all layers have integer or logical values, the returned values also have that datatype.

Note that with `values(x, dataframe=TRUE)` and `as.data.frame(x)` the values returned match the type of each layer, and can be a mix of numeric, logical, integer, and factor.

Usage

```
## S4 method for signature 'SpatRaster'
values(x, mat=TRUE, dataframe=FALSE, row=1,
    nrows=nrow(x), col=1, ncols=ncol(x), na.rm=FALSE, ...)

## S4 method for signature 'SpatVector'
values(x, ...)
```

Arguments

`x`	SpatRaster or SpatVector
`mat`	logical. If `TRUE`, values are returned as a matrix instead of as a vector, except when dataframe is `TRUE`
`dataframe`	logical. If `TRUE`, values are returned as a `data.frame` instead of as a vector (also if matrix is `TRUE`)
`row`	positive integer. Row number to start from, should be between 1 and nrow(x)
`nrows`	positive integer. How many rows?
`col`	positive integer. Column number to start from, should be between 1 and ncol(x)
`ncols`	positive integer. How many columns? Default is the number of columns left after the start column
`na.rm`	logical. Remove `NA`s?
`...`	additional arguments passed to `data.frame`

Details

If x is a SpatRaster, and mat=FALSE, the values are returned as a vector. In cell-order by layer. If mat=TRUE, a matrix is returned in which the values of each layer are represented by a column (with ncell(x) rows). The values per layer are in cell-order, that is, from top-left, to top-right and then down by row. Use as.matrix(x, wide=TRUE) for an alternative matrix representation where the number of rows and columns matches that of x.

Value

matrix or data.frame

Note

raster values that are NA (missing) are represented by NaN (not-a-number) unless argument dataframe is TRUE.

See Also

values<-, focalValues, as.data.frame

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
r
x <- values(r)
x[3650:3655, ]
r[3650:3655]


ff <- system.file("ex/lux.shp", package="terra")
v <- vect(ff)
y <- values(v)
head(y)
```

---

varnames *variable and long variable names*

Description

Set or get names for each dataset (variable) in a SpatRasterDataset.

Each SpatRaster _data source_ can also have a variable name and a long variable name. They are set when reading a file with possibly multiple sub-datasets (e.g. netcdf or hdf5 format) into a single SpatRaster. Each sub-datset is a seperate "data-source" in the SpatRaster. Note that newly created or derived SpatRasters always have a single variable (data source), and therefore the variable names are lost when processing a multi-variable SpatRaster. Thus the variable names are mostly useful to understand a SpatRaster created from some files and for managing SpatRasterDatasets.

See link{names} for the more commonly used _layer_ names.

Usage

```
## S4 method for signature 'SpatRaster'
varnames(x)

## S4 replacement method for signature 'SpatRaster'
varnames(x)<-value

## S4 method for signature 'SpatRaster'
longnames(x)

## S4 replacement method for signature 'SpatRaster'
longnames(x)<-value

## S4 method for signature 'SpatRasterDataset'
varnames(x)

## S4 replacement method for signature 'SpatRasterDataset'
varnames(x)<-value

## S4 method for signature 'SpatRasterDataset'
longnames(x)

## S4 replacement method for signature 'SpatRasterDataset'
longnames(x)<-value
```

Arguments

`x`	SpatRaster, SpatRasterDataset
`value`	character (vector)

Value

character

Note

terra enforces neither unique nor valid names. See `make.unique` to create unique names and `{make.names}` to make syntactically valid names.

Examples

```
s <- rast(ncols=5, nrows=5, nlyrs=3)
names(s) <- c("a", "b", "c")
x <- sds(s, s)
varnames(x) <- c("one", "two")
x
```

vect *Create SpatVector objects*

Description

Methods to create a SpatVector from a filename or other R object.

A filename can be for a shapefile or any spatial file format.

You can use a data.frame to make a SpatVector of points; or a "geom" matrix to make a SpatVector of any supported geometry (see examples and `geom`).

You can supply a list of SpatVectors to append them into a single SpatVector.

SpatVectors can also be created from "Well Known Text", and from spatial vector data objects defined in the `sf` or `sp` packages.

Usage

```
## S4 method for signature 'character'
vect(x, layer="", query="", extent=NULL, filter=NULL,
crs="", proxy=FALSE, what="", opts=NULL)

## S4 method for signature 'matrix'
vect(x, type="points", atts=NULL, crs="")

## S4 method for signature 'data.frame'
vect(x, geom=c("lon", "lat"), crs="", keepgeom=FALSE)

## S4 method for signature 'list'
vect(x, type="points", crs="")

## S4 method for signature 'SpatExtent'
vect(x, crs="")

## S4 method for signature 'SpatVectorCollection'
vect(x)

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

Arguments

`x`	character. A filename; or a "Well Known Text" string; SpatExtent, data.frame (to make a SpatVector of points); a "geom" matrix to make a SpatVector of any supported geometry (see examples and `geom`); a spatial vector data object defined in the `sf` or `sp` packages; or a list with matrices with coordinates
`layer`	character. layer name to select a layer from a file (database) with multiple layers
`query`	character. A query to subset the dataset in the OGR-SQL dialect

extent	Spat* object. The extent of the object is used as a spatial filter to select the geometries to read. Ignored if `filter` is not `NULL`
filter	SpatVector. Used as a spatial filter to select geometries to read (the convex hull is used for lines or points). It is guaranteed that all features that overlap with the extent of filter will be returned. It can happen that additional geometries are returned
type	character. Geometry type. Must be "points", "lines", or "polygons"
atts	data.frame with the attributes. The number of rows must match the number of geometrical elements
crs	character. The coordinate reference system in one of the following formats: WKT/WKT2, <authority>:<code>, or PROJ-string notation (see `crs`)
proxy	logical. If `TRUE` a SpatVectorProxy is returned
what	character indicating what to read. Either `""` for geometries and attributes, or `"geoms"` to only read the geometries, `"attributes"` to only read the attributes (that are returned as a data.frame)
opts	character. GDAL dataset open options
geom	character. The field name(s) with the geometry data. Either two names for x and y coordinates of points, or a single name for a single column with WKT geometries
keepgeom	logical. If `TRUE` the geom variable(s) is (are) also included in the attributes

Value

SpatVector

See Also

`geom`

Examples

```
### SpatVector from file
f <- system.file("ex/lux.shp", package="terra")
f
v <- vect(f)
v

## subsetting (large) files
## with attribute query
v <- vect(f, query="SELECT NAME_1, NAME_2, ID_2 FROM lux WHERE ID_2 < 4")

## with an extent
e <- ext(5.9, 6.3, 49.9, 50)
v <- vect(f, extent=e)

## with polygons
p <- as.polygons(e)
v <- vect(f, filter=p)
```

```
### SpatVector from a geom matrix
x1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60))
x2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
x3 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))
hole <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))
z <- rbind(cbind(object=1, part=1, x1, hole=0), cbind(object=2, part=1, x3, hole=0),
cbind(object=3, part=1, x2, hole=0), cbind(object=3, part=1, hole, hole=1))
colnames(z)[3:4] <- c('x', 'y')

p <- vect(z, "polygons")
p

z[z[, "hole"]==1, "object"] <- 4
lns <- vect(z[,1:4], "lines")
plot(p)
lines(lns, col="red", lwd=2)

### from wkt
v <- vect("POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")

wkt <- c("MULTIPOLYGON ( ((40 40, 20 45, 45 30, 40 40)),
((20 35, 10 30, 10 10, 30 5, 45 20, 20 35),(30 20, 20 15, 20 25, 30 20)))",
"POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")
w <- vect(wkt)

# combine two SpatVectors
vw <- rbind(w, v)

# add a data.frame
d <- data.frame(id=1:2, name=c("a", "b"))
values(w) <- d

# add data.frame on creation, here from a geom matrix
g <- geom(w)
d <- data.frame(id=1:2, name=c("a", "b"))
m <- vect(g, "polygons", atts=d, crs="+proj=longlat +datum=WGS84")

### SpatVector from a data.frame
d$wkt <- wkt
x <- vect(d, geom="wkt")

d$wkt <- NULL
d$lon <- c(0,10)
d$lat <- c(0,10)
x <- vect(d, geom=c("lon", "lat"))

# SpatVector to sf
#sf::st_as_sf(x)
```

vector_layers *List or remove layers from a vector file*

Description

List or remove layers from a vector file that supports layers such as GPGK

Usage

```
vector_layers(filename, delete="", return_error=FALSE)
```

Arguments

`filename`	character. filename
`delete`	character. layers to be deleted (ignored if the value is `""`
`return_error`	logical. If `TRUE`, an error occurs if some layers cannot be deleted. Otherwise a warning is given

viewshed *Compute a viewshed*

Description

Use elevation data to compute the locations that can be seen, or how much higher they would have to be to be seen, from a certain position. The raster data coordinate reference system must planar (not lon/lat), with the elevation values in the same unit as the distance unit of the coordinate reference system.

Usage

```
## S4 method for signature 'SpatRaster'
viewshed(x, loc, observer=1.80, target=0, curvcoef=6/7, output="yes/no", filename="", ...)
```

Arguments

`x`	SpatRaster, single layer with elevation values. Values should have the same unit as the map units
`loc`	location (x and y coordinates) or a cell number
`observer`	numeric. The height above the elevation data of the observer
`target`	numeric. The height above the elevation data of the targets
`curvcoef`	numeric. Coefficient to consider the effect of the curvature of the earth and refraction of the atmosphere. The elevation values are corrected with: `elevation = elevation - curvcoeff * (distance)^2 / (earth_diameter)`. This means that with the default value of 0.85714, you lose sight of about 1 meter of elevation for each 385 m of planar distance

output	character. Can be "yes/no" to get a binary (logical) output showing what areas are visible; "land" to get the height above the current elevation that would be visible; or "sea" the elevation above sea level that would be visible
filename	character. Output filename
...	Options for writing files as in writeRaster

References

The algorithm used is by Wang et al.: https://www.asprs.org/wp-content/uploads/pers/2000journal/january/2000_jan_87-90.pdf.

See Also

terrain

Examples

```
if (gdal() >= 3.1) {
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- project(r, "EPSG:2169")
p <- cbind(70300, 96982)
v <- viewshed(x, p, 0, 0, 0.85714)
}
```

---

voronoi *Voronoi diagram and Delaunay triangles*

Description

Get a Voronoi diagram or Delaunay triangles for points, or the nodes of lines or polygons

Usage

```
## S4 method for signature 'SpatVector'
voronoi(x, bnd=NULL, tolerance=0, as.lines=FALSE, deldir=FALSE)

## S4 method for signature 'SpatVector'
delaunay(x, tolerance=0, as.lines=FALSE)
```

Arguments

x	SpatVector
bnd	SpatVector to set the outer boundary of the voronoi diagram
tolerance	numeric >= 0, snapping tolerance (0 is no snapping)
as.lines	logical. If TRUE, lines are returned without the outer boundary
deldir	logical. If TRUE, the deldir is used instead of the GEOS C++ library method. It has been reported that deldir does not choke on very large data sets

Value

SpatVector

Examples

```
wkt <- c("MULTIPOLYGON ( ((40 40, 20 45, 45 30, 40 40)),
  ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35),(30 20, 20 15, 20 25, 30 20)))",
  "POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")
x <- vect(wkt)
v <- voronoi(x)
v

d <- delaunay(x)
d

plot(v, lwd=2, col=rainbow(15))
lines(x, col="gray", lwd=2)
points(x)
```

---

vrt *Virtual Raster Dataset*

Description

Create a Virtual Raster Dataset (VRT) from a collection of file-based raster datasets (tiles). See gdalbuildvrt for details.

Usage

```
## S4 method for signature 'character'
vrt(x, filename="", options=NULL, overwrite=FALSE, set_names=FALSE, return_filename=FALSE)
```

Arguments

`x`	character. Filenames of raster "tiles". That is, files that have data for, typically non-overlapping, sub-regions of an raster. See `makeTiles`
`filename`	character. output VRT filename
`options`	character. All arguments as separate vector elements. Options as for gdalbuildvrt
`overwrite`	logical. Should `filename` be overwritten if it exists?
`set_names`	logical. Add the layer names of the first tile to the vrt?
`return_filename`	logical. If `TRUE` the filename is returned, otherwise a SpatRaster is returned

Value

SpatRaster

Note

A VRT can reference very many datasets. These are not all opened at the same time. The default is to open not more than 100 files. To increase performance, this maximum limit can be increased by setting the GDAL_MAX_DATASET_POOL_SIZE configuration option to a bigger value with `setGDALconfig`. Note that a typical user process on Linux is limited to 1024 simultaneously opened files.

See Also

`makeTiles` to create tiles; `makeVRT` to create a .vrt file for a binary raster file that does not have a header file. `vrt_tiles` to get the filenames of the tiles in a VRT.

Examples

```
r <- rast(ncols=100, nrows=100)
values(r) <- 1:ncell(r)
x <- rast(ncols=2, nrows=2)
filename <- paste0(tempfile(), "_.tif")
ff <- makeTiles(r, x, filename)
ff

#vrtfile <- paste0(tempfile(), ".vrt")
#v <- vrt(ff, vrtfile)


## output in lower resolution
#vrtfile <- paste0(tempfile(), ".vrt")
#v <- vrt(ff, vrtfile, options = c("-tr", 5, 5))
#head(readLines(vrtfile))
#v
```

`vrt_tiles` *filenames of VRT tiles*

Description

Get the filenames of the tiles in a Virtual Raster Dataset (VRT)

Usage

```
vrt_tiles(x)
```

Arguments

x	character (filename) or SpatRaster

Value

character

See Also

vrt

`watershed` *Catchment delineation*

Description

delineate the area covered by a catchment from a SpatRaster with flow direction and a pour-point (catchment outlet).

Usage

```
## S4 method for signature 'SpatRaster'
watershed(x, pourpoint, filename="",...)
```

Arguments

`x`	SpatRaster with flow direction. See `terrain`.
`pourpoint`	matrix or SpatVector with the pour point location
`filename`	character. Output filename
`...`	additional arguments for writing files as in `writeRaster`

Value

SpatRaster

Author(s)

Ezio Crestaz, Emanuele Cordano, Roman Seliger

Examples

```
elev <- rast(system.file('ex/elev_vinschgau.tif', package="terra"))
flowdir <- terrain(elev, "flowdir")
## pour point at Naturns
pp <- cbind(653358.3, 5168222)
w <- watershed(flowdir, pp)
```

weighted.mean *Weighted mean of layers*

Description

Compute the weighted mean for each cell of the layers of a SpatRaster. The weights can be spatially variable or not.

Usage

```
## S4 method for signature 'SpatRaster,numeric'
weighted.mean(x, w, na.rm=FALSE, filename="", ...)

## S4 method for signature 'SpatRaster,SpatRaster'
weighted.mean(x, w, na.rm=FALSE, filename="", ...)
```

Arguments

`x`	SpatRaster
`w`	A vector of weights (one number for each layer), or for spatially variable weights, a SpatRaster with weights (should have the same extent, resolution and number of layers as x)
`na.rm`	Logical. Should missing values be removed?
`filename`	character. Output filename
`...`	options for writing files as in `writeRaster`

Value

SpatRaster

See Also

`Summary-methods`, `weighted.mean`

Examples

```
b <- rast(system.file("ex/logo.tif", package="terra"))

# give least weight to first layer, most to last layer
wm1 <- weighted.mean(b, w=1:3)

# spatially varying weights
# weigh by column number
w1 <- init(b, "col")

# weigh by row number
w2 <- init(b, "row")
```

```
w <- c(w1, w2, w2)

wm2 <- weighted.mean(b, w=w)
```

---

where *Where are the cells with the min or max values?*

Description

This method returns the cell numbers for the cells with the min or max values of each layer in a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
where.min(x, values=TRUE, list=FALSE)

## S4 method for signature 'SpatRaster'
where.max(x, values=TRUE, list=FALSE)
```

Arguments

`x`	SpatRaster
`values`	logical. If `TRUE` the min or max values are also returned
`list`	logical. If `TRUE` a list is returned instead of a matrix

Value

matrix or list

See Also

`which` and `Summary-methods` for `which.min` and `which.max`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
where.min(r)
```

`which.lyr` *Which cells are TRUE?*

Description

This method returns a single layer SpatRaster with cell values indicating the first layer in the input that is `TRUE`. All numbers that are not zero (or `FALSE`), are considered to be `TRUE`.

Usage

```
## S4 method for signature 'SpatRaster'
which.lyr(x)
```

Arguments

`x`	SpatRaster

Value

SpatRaster

See Also

`isTRUE`, `which`, See `Summary-methods` for `which.min` and `which.max`

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
x <- which.lyr(s > 100)
```

`width` *SpatVector geometric properties*

Description

`width` returns the minimum diameter of the geometry, defined as the smallest band that contains the geometry, where a band is a strip of the plane defined by two parallel lines. This can be thought of as the smallest hole that the geometry can be moved through, with a single rotation.

`clearance` returns the minimum clearance of a geometry. The minimum clearance is the smallest amount by which a vertex could be moved to produce an invalid polygon, a non-simple linestring, or a multipoint with repeated points. If a geometry has a minimum clearance of 'mc', it can be said that:

No two distinct vertices in the geometry are separated by less than 'mc' No vertex is closer than 'mc' to a line segment of which it is not an endpoint. If the minimum clearance cannot be defined for a geometry (such as with a single point, or a multipoint whose points are identical, NA is returned.

Usage

```
## S4 method for signature 'SpatVector'
width(x, as.lines=FALSE)
## S4 method for signature 'SpatVector'
clearance(x, as.lines=FALSE)
```

Arguments

x	SpatVector of lines or polygons
as.lines	logical. If TRUE lines are returned that define the width or clearance

Value

numeric or SpatVector

See Also

minRect

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
width(v)
clearance(v)
```

---

window *Set a window*

Description

Assign a window (area of interest) to a SpatRaster with a `SpatExtent`, or set it to `NULL` to remove the window. This is similar to `crop` without actually creating a new dataset.

The window is intersect with the extent of the SpatRaster. It is envisioned that in a future version, the window may also go outside these boundaries.

Usage

```
## S4 replacement method for signature 'SpatRaster'
window(x)<-value

## S4 method for signature 'SpatRaster'
window(x)
```

Arguments

x	SpatRaster
value	SpatExtent

Value

none for `window<-` and logical for `window`

See Also

crop, extend

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
global(r, "mean", na.rm=TRUE)
e <- ext(c(5.9, 6,49.95, 50))

window(r) <- e
global(r, "mean", na.rm=TRUE)
r

x <- rast(f)
xe <- crop(x, e)
global(xe, "mean", na.rm=TRUE)

b <- c(xe, r)
window(b)
b

window(r) <- NULL
r
```

`wrap` *wrap and unwrap*

Description

Use `wrap` to pack a SpatVector or SpatRaster* to create a Packed* object. Packed objects can be passed over a connection that serializes (e.g. to nodes on a computer cluster). At the receiving end they need to be unpacked with `unwrap`.

Usage

```
## S4 method for signature 'SpatRaster'
wrap(x, proxy=FALSE)

## S4 method for signature 'SpatRasterDataset'
wrap(x, proxy=FALSE)

## S4 method for signature 'SpatRasterCollection'
wrap(x, proxy=FALSE)
```

```
## S4 method for signature 'SpatVector'
wrap(x)

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

Arguments

x	SpatVector, SpatRaster, SpatRasterDataset or SpatRasterCollection
proxy	logical. If FALSE raster cell values are forced to memory if possible. If TRUE, a reference to source filenames is stored for data sources that are not in memory

Value

wrap: Packed* object

unwrap: SpatVector, SpatRaster, SpatRasterCollection, SpatRasterDataset

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
p <- wrap(v)
p
vv <- vect(p)
vv
```

---

wrapCache *SpatRaster wrap with caching options*

Description

Use wrap to pack a SpatRaster with caching options. See wrap for the general approach that is easier and better to use in most cases.

This method allows for specifying a folder, or filenames, to cache all sources of a SpatRaster in a specific location (on disk).

Usage

```
## S4 method for signature 'SpatRaster'
wrapCache(x, filename=NULL, path=NULL, overwrite=FALSE, ...)
```

Arguments

`x`	SpatRaster
`filename`	character. A single filename, or one filename per SpatRaster data source. If not `NULL`, the raster sources are saved in these files
`path`	character. If not `NULL`, the path where raster sources will be saved. Ignored if filenames is not `NULL`
`overwrite`	Should existing files be overwritten when `files` or `path` is not `NULL`? If this value is not `TRUE` or `FALSE`, only files that do not exist are created
`...`	Additional arguments for `writeRaster`. Only used for raster sources that are in memory, as other sources are cached by copying the files

Value

PackedSpatRaster

See Also

`wrap`, `unwrap`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)

x <- wrapCache(r, path=tempdir())
x
```

`writeCDF` *Write raster data to a NetCDF file*

Description

Write a SpatRaster or SpatRasterDataset to a NetCDF file.

When using a SpatRasterDataset, the varname, longname, and unit should be set in the object (see examples).

Always use the `".nc"` or `".cdf"` file extension to assure that the file can be properly read again by GDAL

Usage

```
## S4 method for signature 'SpatRaster'
writeCDF(x, filename, varname, longname="", unit="", split=FALSE, ...)

## S4 method for signature 'SpatRasterDataset'
writeCDF(x, filename, overwrite=FALSE, zname="time", atts="",
    gridmap="", prec="float", compression=NA, missval, ...)
```

Arguments

`x`	SpatRaster or SpatRasterDataset
`filename`	character. Output filename
`varname`	character. Name of the dataset
`longname`	character. Long name of the dataset
`unit`	character. Unit of the data
`split`	logical. If `TRUE` each layer of `x` is treated as a sub-dataset
`atts`	character. A vector of additional global attributes to write. The must be formatted like c("x=a value", "y=abc")
`gridmap`	character. The crs is always writting to the file in standard formats. With this argument you can also write the format commonly used in netcdf files. Something like `c("grid_mapping_name=lambert_azimuthal_equal_area", "longitude_of_projection_origin=10", "latitude_of_projection_origin=52", "false_easting=4321000", "false_northing=3210000")`
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`zname`	character. The name of the "time" dimension
`prec`	character. One of "double", "float", "integer", "short", "byte" or "char"
`compression`	Can be set to an integer between 1 (least compression) and 9 (most compression)
`missval`	numeric, the number used to indicate missing values
`...`	additional arguments passed on to the SpatRasterDataset method, and from there possibly to `ncvar_def`

Value

SpatRaster or SpatDataSet

See Also

see `writeRaster` for writing other file formats

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
fname <- paste0(tempfile(), ".nc")
rr <- writeCDF(r, fname, overwrite=TRUE, varname="alt",
      longname="elevation in m above sea level", unit="m")

a <- rast(ncols=5, nrows=5, nl=50)
values(a) <- 1:prod(dim(a))
time(a) <- as.Date("2020-12-31") + 1:nlyr(a)
aa <- writeCDF(a, fname, overwrite=TRUE, varname="power",
      longname="my nice data", unit="U/Pa")

b <- sqrt(a)
s <- sds(a, b)
```

```
names(s) <- c("temp", "prec")
longnames(s) <- c("temperature (C)", "precipitation (mm)")
units(s) <- c("°C", "mm")
ss <- writeCDF(s, fname, overwrite=TRUE)

# for CRAN
file.remove(fname)
```

`writeRaster` *Write raster data to a file*

Description

Write a SpatRaster to a file.

Usage

```
## S4 method for signature 'SpatRaster,character'
writeRaster(x, filename, overwrite=FALSE, ...)
```

Arguments

`x`	SpatRaster
`filename`	character. Output filename. Can be a single filename, or as many filenames as `nlyr(x)` to write a file for each layer
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional arguments for for writing files. See Details

Details

In writeRaster, and in other methods that generate SpatRasters, options for writing raster files to disk can be provided as additional arguments or, in a few cases, as the `wopt` argument (a named list) if the additional arguments are already used for a different purpose. See `terraOptions` to get or set default values. The following options are available:

name	**description**
`datatype`	values accepted are "INT1U", "INT2U", "INT2S", "INT4U", "INT4S", "FLT4S", "FLT8S". With GDAL >= 3.5 y
`filetype`	file format expresses as GDAL driver names. If this argument is not supplied, the driver is derived from the filena
`gdal`	GDAL driver specific datasource creation options. See the GDAL documentation. For example, with the GeoTiff
`tempdir`	the path where temporary files are to be written to.
`progress`	positive integer. If the number of chunks is larger, a progress bar is shown.
`memfrac`	numeric between 0 and 0.9 (higher values give a warning). The fraction of available RAM that terra is allowed to
`memmax`	memmax - the maximum amount of RAM (in GB) that terra can use when processing a raster dataset. Should be
`names`	output layer names.
`NAflag`	numeric. value to represent missing (`NA` or `NaN`) values. See note
`scale`	numeric. Cell values written to disk are divided by this value (default is 1). See `scoff`
`offset`	numeric. Value that is subtracted from the cell values written to disk (default is 0). See `scoff`

verbose	logical. If TRUE debugging information is printed
steps	positive integers. In how many steps (chunks) do you want to process the data (for debugging)
todisk	logical. If TRUE processing operates as if the dataset is very large and needs to be written to a temporary file (for d

Value

SpatRaster. This function is used for the side-effect of writing values to a file.

Note

GeoTiff files are, by default, written with LZW compression. If you do not want compression, use gdal="COMPRESS=NONE".

When writing integer values the lowest available value (given the datatype) is used to represent NA for signed types, and the highest value is used for unsigned values. This can be a problem with byte data (between 0 and 255) as the value 255 is reserved for NA. To keep the value 255, you need to set another value as NAflag, or do not set a NAflag (with NAflag=NA)

See Also

see writeCDF for writing NetCDF files.

Examples

```
r <- rast(nrows=5, ncols=5, vals=1:25)

# create a temporary filename for the example
f <- file.path(tempdir(), "test.tif")

writeRaster(r, f, overwrite=TRUE)

writeRaster(r, f, overwrite=TRUE, gdal=c("COMPRESS=NONE", "TFW=YES"), datatype='INT1U')

## Or with a wopt argument:

writeRaster(r, f, overwrite=TRUE, wopt= list(gdal=c("COMPRESS=NONE"), datatype='INT1U'))

## remove the file
unlink(f)
```

---

writeVector *Write SpatVector data to a file*

Description

Write a SpatVector to a file. You can choose one of many file formats.

Usage

```
## S4 method for signature 'SpatVector,character'
writeVector(x, filename, filetype=NULL, layer=NULL, insert=FALSE,
    overwrite=FALSE, options="ENCODING=UTF-8")
```

Arguments

x	SpatVector
filename	character. Output filename
filetype	character. A file format associated with a GDAL "driver" such as "ESRI Shapefile". See gdal(drivers=TRUE) or the GDAL docs. If NULL it is attempted to guess the filetype from the filename extension
layer	character. Output layer name. If NULL the filename is used
insert	logical. If TRUE, a new layer is inserted into the file, or an existing layer overwritten (if overwrite=TRUE), if the format supports it (e.g. GPKG allows that). See vector_layers to remove a layer
overwrite	logical. If TRUE and insert=FALSE, filename is overwritten if the file format and layer structure permits it. If TRUE and insert=TRUE, only the target layer is overwritten if the format supports it (e.g. GPKG).
options	character. Format specific GDAL options such as "ENCODING=UTF-8". Use NULL or "" to not use any options

Examples

```
v <- vect(cbind(1:5,1:5))
crs(v) <- "+proj=longlat +datum=WGS84"
v$id <- 1:length(v)
v$name <- letters[1:length(v)]
tmpf1 <- paste0(tempfile(), ".gpkg")
writeVector(v, tmpf1, overwrite=TRUE)
x <- vect(tmpf1)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
tmpf2 <- paste0(tempfile(), ".gpkg")
writeVector(v, tmpf2, overwrite=TRUE)
y <- vect(tmpf2)
```

---

xapp *Apply a function to the cells of a two SpatRasters*

Description

Apply a function to the values of each cell of two (multilayer) SpatRasters.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
xapp(x, y, fun, ..., filename="", overwrite=FALSE, wopt=list())
```

Arguments

`x`	SpatRaster
`y`	SpatRaster with the same geometry as `x`
`fun`	a function that operates on two vectors
`...`	additional arguments for `fun`. These are typically numerical constants. They should *never* be another SpatRaster
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`wopt`	list with named options for writing files as in `writeRaster`

Value

SpatRaster

See Also

`app`, `lapp`, `tapp`, `Math-methods`, `roll`

Examples

```
r <- rast(ncols=10, nrows=10, nlyr=5)
set.seed(1)
r <- init(r, runif)
s <- init(r, runif)
x <- xapp(r, s, fun=cor)
```

`xmin` *Get or set single values of an extent*

Description

Get or set single values of an extent. Values can be set for a SpatExtent or SpatRaster, but not for a SpatVector)

Usage

```
## S4 method for signature 'SpatExtent'
xmin(x)

## S4 method for signature 'SpatExtent'
xmax(x)

## S4 method for signature 'SpatExtent'
ymin(x)

## S4 method for signature 'SpatExtent'
ymax(x)

## S4 method for signature 'SpatRaster'
xmin(x)

## S4 method for signature 'SpatRaster'
xmax(x)

## S4 method for signature 'SpatRaster'
ymin(x)

## S4 method for signature 'SpatRaster'
ymax(x)

## S4 method for signature 'SpatVector'
xmin(x)

## S4 method for signature 'SpatVector'
xmax(x)

## S4 method for signature 'SpatVector'
ymin(x)

## S4 method for signature 'SpatVector'
ymax(x)

## S4 replacement method for signature 'SpatRaster,numeric'
xmin(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
xmax(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
ymin(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
ymax(x)<-value
```

Arguments

x	SpatRaster, SpatExtent, or SpatVector
value	numeric

Value

SpatExtent or numeric coordinate

Examples

```
r <- rast()
ext(r)
ext(c(0, 20, 0, 20))

xmin(r)
xmin(r) <- 0
xmin(r)
```

---

xyRowColCell *Coordinates from a row, column or cell number and vice versa*

Description

Get coordinates of the center of raster cells for a row, column, or cell number of a SpatRaster. Or get row, column, or cell numbers from coordinates or from each other.

Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the SpatRaster (see `ncell`). Row numbers start at 1 at the top, column numbers start at 1 at the left.

When computing row, column, or cell numbers from coordinates, and coordinates fall on the edge of two or four cells, they are assigned to the right-most and/or lowest cell. That is, in these cases of ambiguity, the highest row, column, or cell number is returned.

Usage

```
## S4 method for signature 'SpatRaster,numeric'
xFromCol(object, col)

## S4 method for signature 'SpatRaster,numeric'
yFromRow(object, row)

## S4 method for signature 'SpatRaster,numeric'
xyFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
xFromCell(object, cell)
```

```
## S4 method for signature 'SpatRaster,numeric'
yFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
colFromX(object, x)

## S4 method for signature 'SpatRaster,numeric'
rowFromY(object, y)

## S4 method for signature 'SpatRaster,numeric,numeric'
cellFromRowCol(object, row, col)

## S4 method for signature 'SpatRaster,numeric,numeric'
cellFromRowColCombine(object, row, col)

## S4 method for signature 'SpatRaster,numeric,numeric'
rowColCombine(object, row, col)

## S4 method for signature 'SpatRaster,numeric'
rowFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
colFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
rowColFromCell(object, cell)

## S4 method for signature 'SpatRaster,matrix'
cellFromXY(object, xy)
```

Arguments

`object`	SpatRaster
`cell`	integer. cell number(s)
`col`	integer. column number(s) or missing (equivalent to all columns)
`row`	integer. row number(s) or missing (equivalent to all rows)
`x`	x coordinate(s)
`y`	y coordinate(s)
`xy`	matrix of x and y coordinates

Value

xFromCol, yFromCol, xFromCell, yFromCell: vector of x or y coordinates

xyFromCell: matrix(x,y) with coordinate pairs

colFromX, rowFromY, cellFromXY, cellFromRowCol, rowFromCell, colFromCell: vector of row, column, or cell numbers

rowColFromCell, rowColCombine: matrix of row and column numbers

See Also

crds

Examples

```
r <- rast()

xFromCol(r, c(1, 120, 180))
yFromRow(r, 90)
xyFromCell(r, 10000)
xyFromCell(r, c(0, 1, 32581, ncell(r), ncell(r)+1))

cellFromRowCol(r, 5, 5)
cellFromRowCol(r, 1:2, 1:2)
cellFromRowCol(r, 1, 1:3)

# all combinations
cellFromRowColCombine(r, 1:2, 1:2)

colFromX(r, 10)
rowFromY(r, 10)
xy <- cbind(lon=c(10,5), lat=c(15, 88))
cellFromXY(r, xy)

# if no row/col specified all are returned
range(xFromCol(r))
length(yFromRow(r))
```

---

zonal *Zonal statistics*

Description

Compute zonal statistics, that is summarize values of a SpatRaster for each "zone" defined by another SpatRaster, or by a SpatVector with polygon geometry.

If fun is a true R function, the <SpatRaster,SpatRaster> method may fail when using very large SpatRasters, except for the functions ("mean", "min", "max", "sum", "isNA", and "notNA").

You can also summarize values of a SpatVector for each polygon (zone) defined by another SpatVector.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
zonal(x, z, fun="mean", ..., w=NULL, wide=TRUE,
as.raster=FALSE, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRaster,SpatVector'
zonal(x, z, fun="mean", na.rm=FALSE, w=NULL, weights=FALSE,
```

```
exact=FALSE, touches=FALSE, small=TRUE, as.raster=FALSE,
as.polygons=FALSE, wide=TRUE, filename="", wopt=list())

## S4 method for signature 'SpatVector,SpatVector'
zonal(x, z, fun=mean, ..., weighted=FALSE, as.polygons=FALSE)
```

Arguments

x	SpatRaster or SpatVector
z	SpatRaster with cell-values representing zones or a SpatVector with each polygon geometry representing a zone. z can have multiple layers to define intersecting zones
fun	function to be applied to summarize the values by zone. Either as character: "mean", "min", "max", "sum", "isNA", and "notNA" and, for relatively small SpatRasters, a proper function
...	additional arguments passed to fun, such as `na.rm=TRUE`
w	SpatRaster with weights. Should have a single-layer with non-negative values
wide	logical. Should the values returned in a wide format? For the `SpatRaster, SpatRaster` method this only affects the results when `nlyr(z) == 2`. For the `SpatRaster, SpatVector` method this only affects the results when `fun=table`
as.raster	logical. If `TRUE`, a SpatRaster is returned with the zonal statistic for each zone
filename	character. Output filename (ignored if `as.raster=FALSE`
overwrite	logical. If `TRUE`, `filename` is overwritten
wopt	list with additional arguments for writing files as in `writeRaster`
weights	logical. If `TRUE` and `y` has polygons, the approximate fraction of each cell that is covered is returned as well, for example to compute a weighted mean
exact	logical. If `TRUE` and `y` has polygons, the exact fraction of each cell that is covered is returned as well, for example to compute a weighted mean
touches	logical. If `TRUE`, values for all cells touched by lines or polygons are extracted, not just those on the line render path, or whose center point is within the polygon. Not relevant for points; and always considered `TRUE` when `weights=TRUE` or `exact=TRUE`
small	logical. If `TRUE`, values for all cells in touched polygons are extracted if none of the cells center points is within the polygon; even if `touches=FALSE`
weighted	logical. If `TRUE`, a weighted.mean is computed and `fun` is ignored. Weights are based on the length of the lines or the area of the polygons in `x` that intersect with `z`. This argument is ignored of `x` is a SpatVector or points
as.polygons	logical. Should the zonal statistics be combined with the geometry of `z`?
na.rm	logical. If `TRUE`, `NA`s are removed

Value

A `data.frame` with a value for each zone, or a SpatRaster, or SpatVector of polygons.

See Also

See global for "global" statistics (i.e., all of x is considered a single zone), app for local statistics, and extract for an alternative way to summarize values of a SpatRaster with a SpatVector. With aggregate you can compute statistics for cell blocks defined by a number of rows and columns.

Examples

```
### SpatRaster, SpatRaster
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
z <- rast(r)
values(z) <- rep(c(1:2, NA, 3:4), each=20)
names(z) <- "zone"
zonal(r, z, "sum", na.rm=TRUE)

# with weights
w <- init(r, "col")
zonal(r, z, w=w, "mean", na.rm=TRUE)

# multiple layers
r <- rast(system.file("ex/logo.tif", package = "terra"))
# zonal layer
z <- rast(r, 1)
names(z) <- "zone"
values(z) <- rep(c(1:2, NA, c(3:4)), each=ncell(r)/5, length.out=ncell(r))

zonal(r, z, "mean", na.rm = TRUE)

# raster of zonal values
zr <- zonal(r, z, "mean", na.rm = TRUE, as.raster=TRUE)


### SpatRaster, SpatVector
x <- rast(ncol=2,nrow=2, vals=1:4, xmin=0, xmax=1, ymin=0, ymax=1, crs="+proj=utm +zone=1")
p <- as.polygons(x)
pp <- shift(p, .2)
r <- disagg(x, 4)

zonal(r, p)
zonal(r, p, sum)
zonal(x, pp, exact=TRUE)
zonal(c(x, x*10), pp, w=x)


### SpatVector, SpatVector

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)[,c(2,4)]

p <- spatSample(v, 100)
values(p) <- data.frame(b2=1:100, ssep1=100:1)
```

```
zonal(p, v, mean)
```

`zoom` *Zoom in on a map*

Description

Zoom in on a map (plot) by providing a new extent, by default this is done by clicking twice on the map.

Usage

```
## S4 method for signature 'SpatRaster'
zoom(x, e=draw(), maxcell=100000, layer=1, new=FALSE, ...)

## S4 method for signature 'SpatVector'
zoom(x, e=draw(), new=FALSE, ...)
```

Arguments

`x`	SpatRaster
`e`	SpatExtent
`maxcell`	positive integer. Maximum number of cells used for the map
`layer`	positive integer to select the layer to be used
`new`	logical. If `TRUE`, the zoomed in map will appear on a new device (window)
`...`	additional arguments passed to `plot`

Value

SpatExtent (invisibly)

See Also

`draw`, `plot`

Index