Package 'spatial.tools'

February 20, 2015

Maintainer Jonathan Asher Greenberg <spatial-tools@estarcion.net>

License GPL (>= 2)

Title R functions for working with spatial data.

Type Package

LazyLoad yes

Author Jonathan Asher Greenberg

Description Spatial functions meant to enhance the core functionality of the package ``raster'', including a parallel processing engine for use with rasters.

Version 1.4.8

Date 2014-7-2

URL http://www.geog.illinois.edu/~jgrn/software-and-datasets/rasterengine-tutorial/

Depends parallel, iterators, foreach, rgdal, raster, R (>= 3.0.0)

Imports mmap, abind, doParallel, compiler

NeedsCompilation no

Repository CRAN

Date/Publication 2014-07-02 20:57:52

R topics documented:

`add_leading_zeroes` *Add Leading Zeroes to a Numeric Vector*

Description

Appends leading zeroes to a vector of numbers based on a string length or a maximum number.

Usage

```
add_leading_zeroes(number, number_length, max_number)
```

Arguments

`number`	A numeric vector.
`number_length`	The length of the output string.
`max_number`	A number to base the length of the output string on.

Value

A character vector.

Author(s)

Jonathan A. Greenberg

Examples

```
x=c(1:10)
add_leading_zeroes(x,number_length=4)
add_leading_zeroes(x,max_number=10000)
```

`bbox_to_SpatialPolygons` *Create a SpatialPolygons Bounding Box*

Description

Create a SpatialPolygons Bounding Box

Usage

```
bbox_to_SpatialPolygons(x, proj4string = CRS(as.character(NA)))
```

Arguments

`x`	Raster*, extent, bbox or matrix. See Details.
`proj4string`	CRS object. Used to define the CRS if it is missing from x.

Details

This function generates a SpatialPolygons object from either a Raster* object, an extent object, a bbox object, or a 2x2 Matrix that follows the form generated by the bbox() function (rows = x and y, cols = min and max). Note that with extent and matrix objects, the CRS will need to be set manually.

Value

A SpatialPolygons object.

Author(s)

Jonathan A. Greenberg

See Also

`bbox`, `extent`

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
bbox_to_SpatialPolygons(tahoe_highrez)
tahoe_highrez_extent <- extent(tahoe_highrez)
bbox_to_SpatialPolygons(tahoe_highrez_extent,
CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"))
tahoe_highrez_bbox <- bbox(tahoe_highrez)
bbox_to_SpatialPolygons(tahoe_highrez_bbox,
CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"))
```

---

binary_image_write *Writes image data to a flat binary file using col/row/band positioning.*

Description

Writes image data to a flat binary file using col/row/band positioning.

Usage

```
binary_image_write(filename, mode = real64(), image_dims,
  interleave = "BSQ", data, data_position)
```

Arguments

filename	Character. The path and filename of a "blank" binary file to store the image data.
mode	The mode of data on disk. Defaults to real64() (double precision floating point).
image_dims	Vector. Vector of length(image_dims)==3 representing the number of columns, rows and bands in the output image.
interleave	Character. The require output interleave. By default is "BSQ". OTHER INTERLEAVES CURRENTLY UNSUPPORTED.
data	Vector, matrix, array, or other data source which is coercible to a vector. This is the data to be written to the image.
data_position	List. A length==3 list, containing the column, row, and band positions ranges to write the output data.

Author(s)

Jonathan A. Greenberg

See Also

mmap,create_blank_raster

Examples

```
## Not run:
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
# Create a blank file using create_blank_raster
test_blank_file <- create_blank_raster(reference_raster=tahoe_highrez)
blank_raster <- brick(test_blank_file)
# It should be all 0s:
setMinMax(blank_raster)
# Write some ones to to the 100th line, columns 25 to 50, bands 1 and 3:
data_position <- list(25:50,100,c(1,3))
data1s <- array(1,dim=c(
length(data_position[[1]]),
length(data_position[[2]]),
length(data_position[[3]])))
plot(raster(test_blank_file,layer=1))
binary_image_write(filename=test_blank_file,
mode=real64(),image_dims=dim(tahoe_highrez),interleave="BSQ",
data=data1s,data_position=data_position)
setMinMax(blank_raster)
plot(raster(blank_raster,layer=1))

## End(Not run)
```

---

brickstack_to_raster_list

Converts a RasterBrick/RasterStack to a list of RasterLayers

Description

Converts a RasterBrick/RasterStack to a list of RasterLayers

Usage

```
brickstack_to_raster_list(x)
```

Arguments

x	A RasterBrick or RasterStack.

Value

A list of RasterLayers.

Author(s)

Jonathan A. Greenberg

Examples

```
# You can speed this up if a parallel backend is running, e.g.:
# sfQuickInit()
registerDoSEQ() # Just to avoid the warning from foreach.
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
tahoe_highrez_list <- brickstack_to_raster_list(tahoe_highrez)
tahoe_highrez_list
# sfQuickStop()
```

`build_raster_header` *Builds a raster header for a flat binary file.*

Description

Builds a raster header for a flat binary file.

Usage

```
build_raster_header(x_filename, reference_raster, out_nlayers,
  datatype = "FLT8S", format = "raster", bandorder = "BSQ",
  setMinMax = FALSE, additional_header = NULL, verbose = FALSE)
```

Arguments

`x_filename`	Character. The filename of the input binary file.
`reference_raster`	Raster*. A Raster* object containing the header information to be used.
`out_nlayers`	Numeric. The number of layers in the flat binary file (defaults to nlayers(reference_raster)).
`datatype`	Character. The dataType of the flat binary file. See ?dataType for available datatypes. Default is 'FLT8S'.
`bandorder`	Character. The bandorder ('BIP','BIL','BSQ') of the file. Default is 'BSQ'.
`format`	Character. The format of the header. See ?hdr for valid entries. Default is 'raster'. CURRENTLY UNSUPPORTED.
`setMinMax`	Logical. Set the min/max for the file (will take longer to execute)? Default=FALSE.
`additional_header`	Character. Create additional output headers for use with other GIS systems (see `hdr`). Set to NULL (default) to suppress.
`verbose`	logical. Enable verbose execution? Default is FALSE.

Author(s)

Jonathan A. Greenberg and Robert Hijmans (<spatial.tools@estarcion.net>)

See Also

`hdr`,`dataType`

Examples

```
## Not run:
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
test_blank_file <- create_blank_raster(filename=paste(tempfile(),".gri",sep=""),
reference_raster=tahoe_highrez,nlayers=2,
create_header=FALSE,format="raster",datatype="FLT8S",bandorder="BSQ")
test_blank_raster <- build_raster_header(x_filename=test_blank_file,
reference_raster=tahoe_highrez,out_nlayers=2,
datatype='FLT8S',format='raster',bandorder="BSQ",setMinMax=TRUE)
test_blank_raster

## End(Not run)
```

`create_blank_raster` *Create an empty raster and header.*

Description

This function creates an arbitrarily large raster as a flat binary file with (optionally) a proper header for use with other functions. This should create the blank files very quickly, as it is using some OS tricks to carve out a block of space rather than writing a bunch of 0s to disk sequentially.

Usage

```
create_blank_raster(filename = NULL, format = "raster",
  datatype = "FLT8S", bandorder = "BSQ", nrow = NULL, ncol = NULL,
  nlayers = NULL, create_header = TRUE, reference_raster = NULL,
  return_filename = TRUE, additional_header = NULL, overwrite = FALSE,
  verbose = FALSE)
```

Arguments

`filename`	Character. The output base filename of the blank file. Will use tempfile() if nothing is provided.
`format`	Character. Output format. Currently only supports "raster".
`datatype`	Character. Output number type. See ?dataType. Default is "FLT8S".
`bandorder`	Character. Output band interleave. Currently only supports "BSQ".
`nrow`	Numeric. Number of rows of the output raster. Defaults to nrow(reference_raster).
`ncol`	Numeric. Number of columns of the output raster. Defaults to ncol(reference_raster).
`nlayers`	Numeric. Number of layers of the output raster Defaults to nlayer(reference_raster).
`create_header`	Logical. Create a properly formatted header for the blank file?
`reference_raster`	Raster*. Reference raster to derive other information, e.g. resolution, projection, datum.

`return_filename`
: Logical. Return filename of the binary file (if TRUE, default) or the Raster* itself (if FALSE).

`additional_header`
: Character. Create additional output headers for use with other GIS systems (see `hdr`). Set to NULL (default) to suppress.

`overwrite`
: Logical. Overwrite an existing file with the same name?

`verbose`
: Logical. Enable verbose execution? Default is FALSE.

Details

create_blank_raster is designed to quickly create a binary file of the appropriate size using tricks with seek()/writeBin(). A large file can be created in a fraction of a second using this function. This file could, for example, be used with mmap to realize asynchronous or, OS permitting, parallel writes to a single file. Note that setMinMax are NOT performed on the output file (to save time), so on some systems you may see a warning.

Binary files of this type are used by a number of raster formats, including raster and ENVI.

Value

A character vector (return_filename==TRUE) or as Raster* object (return_filename==TRUE)

Author(s)

Jonathan A. Greenberg

See Also

`hdr`

Examples

```
## Not run:
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
test_blank_file <- create_blank_raster(reference_raster=tahoe_highrez)
file.info(test_blank_file)
test_blank_raster <- create_blank_raster(reference_raster=tahoe_highrez,return_filename=FALSE)
test_blank_raster

## End(Not run)
```

fix_extent *Forces a list of Raster*s to all have the same extent.*

Description

Forces a list of Raster*s to all have the same extent.

Usage

```
fix_extent(extent_reference, broken_extents)
```

Arguments

extent_reference	Raster*. A Raster* object that will provide the extent to all the other Raster*s. If unassigned, will assume it is the first Raster* in the broken_extents list.
broken_extents	list of Raster* objects. Raster* objects that will be coerced to the extent_reference's extent.

Author(s)

Jonathan A. Greenberg (<spatial.tools@estarcion.net>)

See Also

extent,stack

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
tahoe_highrez
tahoe_highrez_broken <- tahoe_highrez
# We'll "break" the extent:
extent(tahoe_highrez_broken) <- c(0,360,-90,90)
tahoe_highrez_broken
tahoe_highrez_fixed <- fix_extent(tahoe_highrez,tahoe_highrez_broken)
tahoe_highrez_fixed
```

focal_hpc *Engine for performing fast, easy-to-develop pixel and focal raster calculations with parallel processing capability.*

Description

Engine for performing fast, easy-to-develop pixel and focal raster calculations with parallel processing capability.

Usage

```
focal_hpc(x, fun, args = NULL, window_dims = c(1, 1),
  window_center = c(ceiling(window_dims[1]/2), ceiling(window_dims[2]/2)),
  filename = NULL, overwrite = FALSE, outformat = "raster",
  additional_header = "ENVI", datatype = "FLT8S", processing_unit = NULL,
  chunk_format = "array", minblocks = "max", blocksize = NULL,
  outbands = NULL, outfiles = NULL, setMinMax = FALSE,
  debugmode = FALSE, .packages = NULL, clearworkers = TRUE,
  verbose = FALSE, ...)
```

Arguments

x	Raster*. A Raster* used as the input into the function. Multiple inputs should be stack()'ed or list()'ed together.
fun	function. A focal function to be applied to the image. See Details.
args	list. Arguments to pass to the function (see ?mapply). Note that the 'fun' should explicitly name the variables.
window_dims	Vector. The size of a processing window in col x row order. Be default, a single pixel (c(1,1).
window_center	Vector. The local coordinate of the center of a processing window. By default the middle of the processing window. UNSUPPORTED.
chunk_format	Character. The format to send the chunk to the function. Can be "array" (default) or "raster".
minblocks	Numeric. The minimum number of chunks to divide the raster into for processing. Defaults to 1.
blocksize	Numeric. The size (in rows) for a block of data. If unset, focal_hpc will attempt to figure out an optimal blocksize.
filename	Character. Filename(s) of the output raster.
outformat	Character. Outformat of the raster. Must be a format usable by hdr(). Default is 'raster'. CURRENTLY UNSUPPORTED.
overwrite	Logical. Allow files to be overwritten? Default is FALSE.
processing_unit	Character. ("single"|"chunk") Will be auto-set if not specified ("chunk" for pixel-processing, "single" for focal processing). See Description.

outbands	Numeric. If known, how many bands in each output file? Assigning this and outfiles will allow focal_hpc to skip the pre-check.
outfiles	Numeric. If known, how many output files? Assigning this and outbands will allow focal_hpc to skip the pre-check.
setMinMax	Logical. Run a setMinMax() on each output file after processing (this will slow the processing down). Default is FALSE.
additional_header	Character. Create additional output headers for use with other GIS systems (see hdr). Set to NULL to suppress. Default is "ENVI".
datatype	Character. Output number type. See ?dataType. Default is "FLT8S".
debugmode	Logical or Numeric. If TRUE or 1, the function will enter debug mode during the test phase. If debugmode equals 2, the function will stop after the test phase, but won't explicitly enter debug mode. This is useful if the user function has a browser() statement within it. Note the inputs will be an array of size 2 columns, 1 row, and how ever many input bands.
.packages	Character. A character vector of package names needed by the function (parallel mode only).
clearworkers	Logical. Force the workers to clear all objects upon completing (releasing memory)? Default=TRUE.
verbose	logical. Enable verbose execution? Default is FALSE.
...	Additional parameters (none at present).

Details

focal_hpc is designed to execute a function on a Raster* object using foreach, to achieve parallel reads, executions and writes. Parallel random writes are achieved through the use of mmap, so individual image chunks can finish and write their outputs without having to wait for all nodes in the cluster to finish and then perform sequential writing. On Windows systems, random writes are possible but apparently not parallel writes. focal_hpc solves this by trying to write to a portion of the image file, and if it finds an error (a race condition occurs), it will simply retry the writes until it successfully finishes. On Unix-alikes, truly parallel writes should be possible.

Note that rasterEngine is a convienence wrapper for focal_hpc and, in general, should be used instead of focal_hpc directly.

focal_hpc operates in two modes, which have different input and outputs to the function:

Pixel based processing:

1) If chunk_format=="array" (default), the input to the function should assume an array of dimensions x,y,z where x = the number of columns in a chunk, y = the number of rows in the chunk, and z = the number of bands in the chunk. If chunk_format=="raster", the input to the function will be a raster subset. Note that we are ordering the array using standards for geographic data, (columns, rows, bands), not how R usually thinks of arrays (rows, columns, bands).

2) The output of the function should always be an array with the x and y dimensions matching the input, and an arbitrary number of band outputs. Remember to order the dimensions as columns, rows, bands (x,y,z).

Local window processing:

1) The function should be written to process a SINGLE window at a time, given the dimensions of window_dims, so the input to the function should assume a window of dimensions window_dims with a local center defined by window_center. As with before, the input can be passed to the function as an array (suggested) or a small raster.

2) The output should be a single pixel value, so can either be a single value, or a vector (which is assumed to be multiple bands of a single pixel).

The speed of the execution when running in parallel will vary based on the specific setup, and may, indeed, be slower than a sequential execution (e.g. with calc()), particularly on smaller files. Note that by simply running sfQuickStop(), focal_hpc will run in sequential mode.

Author(s)

Jonathan A. Greenberg (<spatial.tools@estarcion.net>)

See Also

rasterEngine, foreach, mmap, dataType, hdr

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
# Pixel-based processing:
ndvi_function <- function(x)
{
# Note that x is received by the function as a 3-d array:
red_band <- x[,,2]
nir_band <- x[,,3]
ndvi <- (nir_band - red_band)/(nir_band + red_band)
# The output of the function should also be a 3-d array,
# even if it is a single band:
ndvi <- array(ndvi,dim=c(dim(x)[1],dim(x)[2],1))
return(ndvi)
}

sfQuickInit(cpus=2)
tahoe_ndvi <- focal_hpc(x=tahoe_highrez,fun=ndvi_function)
sfQuickStop()

## Not run:
# Focal-based processing:
local_smoother <- function(x)
{
# Assumes a 3-d array representing
# a single local window, and return
# a single value or a vector of values.
smoothed <- apply(x,3,mean)
return(smoothed)
}

# Apply the function to a 3x3 window:
sfQuickInit(cpus=2)
```

```
tahoe_3x3_smoothed <- focal_hpc(x=tahoe_highrez,fun=local_smoother,window_dims=c(3,3))
sfQuickStop()

# Example with 7 x 7 window in full parallel mode:
sfQuickInit()
tahoe_7x7_smoothed <- focal_hpc(x=tahoe_highrez,fun=local_smoother,window_dims=c(7,7))
sfQuickStop()

## End(Not run)
```

`getValuesBlock_enhanced` *Easier-to-use function for grabbing a block of data out of a Raster*.*

Description

Easier-to-use function for grabbing a block of data out of a Raster*.

Usage

```
getValuesBlock_enhanced(x, r1 = 1, r2 = nrow(x), c1 = 1, c2 = ncol(x),
  lyrs = seq(nlayers(x)), format = "array", ...)
```

Arguments

`x`	Raster* Some input Raster* object.
`r1`	Numeric. The start row of the chunk.
`r2`	Numeric. The end row of the chunk.
`c1`	Numeric. The start column of the chunk.
`c2`	Numeric. The end row of the chunk.
`lyrs`	Numeric. Vector of layer IDs. Defaults to all layers (1:nlayers(x)).
`format`	Character. See Details.
`...`	Other parameters.

Details

This allows for a larger number of output formats to be generated when extracting chunks of data from a Raster* object. If format="array" (default), the chunk will be returned in a 3-d array with dimensions representing column,row,and layer. If "raster", the chunk will be returned as a Raster* object. If "data.frame", it will be returned as a data.frame. If "data.frame.dims", it will return a list, where the first component (named "values") is the same as the data.frame when using format="data.frame", and the second component (named "dim") is the dimensions of the extracted chunk.

Value

An array or raster object.

Author(s)

Jonathan A. Greenberg

See Also

getValues

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
mychunk <- getValuesBlock_enhanced(tahoe_highrez,r1=100,r2=110,c1=20,c2=50)
class(mychunk)
dim(mychunk)
mychunk_raster <- getValuesBlock_enhanced(tahoe_highrez,r1=100,r2=110,c1=20,c2=50,format="raster")
mychunk_raster
```

---

getValuesBlock_stackfix

Get a block of raster cell values (optimization fix for RasterStacks)

Description

A faster version of getValuesBlock for RasterStack.

Usage

```
getValuesBlock_stackfix(x, row = 1, nrows = 1, col = 1, ncols = (ncol(x)
  - col + 1), lyrs = (1:nlayers(x)))
```

Arguments

x	Raster* object
row	positive integer. Row number to start from, should be between 1 and nrow(x)
nrows	postive integer. How many rows? Default is 1
col	postive integer. Column number to start from, should be between 1 and ncol(x)
ncols	postive integer. How many columns? Default is the number of colums left after the start column
lyrs	integer (vector). Which layers? Default is all layers (1:nlayers(x))

Details

In certain cases, getValuesBlock may run very slowly on a RasterStack, particularly when the RasterStack is comprised of RasterBricks. This code attempts to fix the inefficiency by running the extract on each unique file of the RasterStack, rather than each unique layer.

Value

matrix or vector (if (x=RasterLayer), unless format='matrix')

Author(s)

Jonathan A. Greenberg

See Also

getValuesBlock

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
tahoe_highrez_stack <- stack(tahoe_highrez,tahoe_highrez,tahoe_highrez)
# getValuesBlock stack extraction:
system.time(tahoe_highrez_extract <- getValuesBlock(tahoe_highrez_stack))
# getValuesBlock_stackfix stack extraction:
system.time(tahoe_highrez_extract <- getValuesBlock_stackfix(tahoe_highrez_stack))
```

---

is.Raster *Tests if an input is a RasterLayer, RasterBrick, or a RasterStack.*

Description

Tests if an input is a RasterLayer, RasterBrick, or a RasterStack.

Usage

```
is.Raster(x)
```

Arguments

x	an R Object.

Value

A logical vector.

Author(s)

Jonathan A. Greenberg

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
is.Raster(tahoe_highrez)
tahoe_lidar_bareearth <-
raster(system.file("external/tahoe_lidar_bareearth.tif", package="spatial.tools"))
is.Raster(tahoe_lidar_bareearth)
is.Raster("character")
```

`list.raster.files` *Spiders a directory for raster files.*

Description

Spiders a directory for raster files.

Usage

```
list.raster.files(path = ".", pattern = NULL, recursive = FALSE,
  return_rasters = FALSE, return_bbox = TRUE,
  bbox_CRS = CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"))
```

Arguments

`path`	Character. The path to search for raster files.
`pattern`	Character. A regular expression to limit the files that are tested/returned.
`recursive`	Logical. Search nested subdirectories within the path?
`return_rasters`	Logical. Return all proper files as RasterBrick objects (TRUE) or as filenames (FALSE).
`return_bbox`	Logical. Return a SpatialPolygonsDataFrame with the bounding box geometry and the filenames as data.frame attributes.
`bbox_CRS`	CRS. If return_bbox==TRUE, the CRS that the output bbox SpatialPolygonsDataFrame should be in.

Details

This function searches through a path (potentially recursively), and returns the filename and/or a brick for each file that can be coerced safely to a RasterBrick object. Note that given different flavors of GDAL, this may return different results for the same directory on different computers. If a foreach parallel backend has been registered, the spidering will use parallel processing to check each file, potentially speeding it up.

Value

A list of filenames (return_rasters=FALSE) or a list of RasterBricks (return_rasters=TRUE) and/or a SpatialPolygonsDataFrame of the bounding boxes (return_bbox==TRUE).

Author(s)

Jonathan A. Greenberg

See Also

brick,list.files

Examples

```
{
search_folder <- system.file("external/", package="spatial.tools")
# sfQuickInit() # To potentially speed the search up.
list.raster.files(path=search_folder)
list.raster.files(path=search_folder,return_rasters=TRUE)
# sfQuickStop()
}
```

---

modify_raster_margins *Add/subtract rows and columns from Raster**

Description

Add/subtract rows and columns from Raster*

Usage

```
modify_raster_margins(x, extent_delta = c(0, 0, 0, 0), value = NA)
```

Arguments

x	A Raster* object.
extent_delta	Numeric vector. How many rows/columns to add/subtract to the left,right,top, and bottom of an image. Default is c(0,0,0,0) (no change).
value	Value to fill in when adding rows/columns.

Details

A quick way to add/subtract margins from a Raster* object. extent_delta is a four-element integer vector that describes how many rows/columns to add to the (left,right,top,bottom) of the image (in that order). Negative values remove rows, positive values add rows.

Value

A Raster* object.

Author(s)

Jonathan A. Greenberg

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
dim(tahoe_highrez)
# Remove one row and column from the top, bottom, left, and right:
tahoe_highrez_cropped <- modify_raster_margins(x=tahoe_highrez,extent_delta=c(-1,-1,-1,-1))
dim(tahoe_highrez_cropped)
# Add two rows to the top and left of the raster, and fill with the value 100.
tahoe_highrez_expand <- modify_raster_margins(x=tahoe_highrez,extent_delta=c(2,0,2,0),value=100)
dim(tahoe_highrez_expand)
```

---

predict_rasterEngine *Model predictions (including Raster* objects)*

Description

Model predictions (including Raster* objects)

Usage

```
predict_rasterEngine(object, na.rm.mode = TRUE, debugmode = FALSE, ...)
```

Arguments

object	a model object for which prediction is desired.
na.rm.mode	Logical. Attempt to fix missing data, even if the model object doesn't support na.rm? Default is TRUE.
debugmode	Logical. Internal debugging for the code, will be removed eventually. Default is FALSE.
...	additional arguments affecting the predictions produced.

Details

predict will operate normally, unless a parameter named "newdata" is found and it is of class Raster*. If this occurs, predict will use rasterEngine to perform a prediction. Currently, this works for predict.* statements in which the data to predict on is called by the parameter "newdata", the input data is in the form of a data.frame, and the output is a vector or matrix of numbers or factors.

predict will run in parallel if a cluster is registered with foreach via a do* statement, or if the user uses sfQuickInit().

Author(s)

Jonathan A. Greenberg (<spatial.tools@estarcion.net>)

See Also

predict

Examples

```
# This example creates a linear model relating a vegetation
# index (NDVI) to vegetation height, and applies it to a raster
# of NDVI.

# Load up a 3-band image:
tahoe_highrez <- setMinMax(
brick(system.file("external/tahoe_highrez.tif", package="spatial.tools")))

# Determine NDVI
ndvi_nodrop <- function(GRNIR_image)
{
red_band <- GRNIR_image[,,2,drop=FALSE]
nir_band <- GRNIR_image[,,3,drop=FALSE]
ndvi <- (nir_band-red_band)/(nir_band + red_band)
return(ndvi)
}

tahoe_ndvi <- rasterEngine(GRNIR_image=tahoe_highrez,fun=ndvi_nodrop)
names(tahoe_ndvi) <- "ndvi"

# Load up Lidar files
tahoe_lidar_highesthit <- setMinMax(
raster(system.file("external/tahoe_lidar_highesthit.tif", package="spatial.tools")))

tahoe_lidar_bareearth <- setMinMax(
raster(system.file("external/tahoe_lidar_bareearth.tif", package="spatial.tools")))

# Determine vegetation height:
LIDAR_height <- function(bareearth,firstreturn)
{
height <- firstreturn-bareearth
return(height)
}

tahoe_height <- rasterEngine(
bareearth=tahoe_lidar_bareearth,
firstreturn=tahoe_lidar_highesthit,
fun=LIDAR_height)
names(tahoe_height) <- "vegetation_height"

# Stack them:
tahoe_analysis_stack <- stack(tahoe_ndvi,tahoe_height)

# Pick some random points from the stack
randomly_extracted_data <- as.data.frame(sampleRandom(tahoe_analysis_stack,size=100))

# Generate a linear model from these points:
height_from_ndvi_model <- lm(vegetation_height~ndvi,data=randomly_extracted_data)

# Apply model to NDVI image:
# Enable parallel engine to run larger images faster:
```

```
# sfQuickInit()
height_from_ndvi_raster <- predict_rasterEngine(object=height_from_ndvi_model,newdata=tahoe_ndvi)
# sfQuickStop()
```

`projectRaster_rigorous` *Performs an area-weighted resampling of raster datasets.*

Description

Performs an area-weighted resampling of raster datasets.

Usage

```
projectRaster_rigorous(from, to, method = "mode", na.rm = FALSE,
  verbose = FALSE, ...)
```

Arguments

`from`	Raster* The sources raster to be resampled.
`to`	Raster* A target raster that the from will be resampled to (extent, resolution, projection).
`method`	Character. Default is "mode". See details.
`na.rm`	Logical. Remove NAs before calculating cell stats?
`verbose`	logical. Enable verbose execution? Default is FALSE.
`...`	Currently unsupported.

Details

This function is designed to solve the problem of resampling/reprojecting rasters using area-based, not point based (e.g. nearest neighbor, bilinear, cubic convolution), resampling. The output pixel is a function of the areas of the input pixels, so this should be used for resampling from a finer resolution to a coarser resolution.

The method defaults to "mode", which will return the value covering the largest area of the output pixel area. Other methods will be added in the future.

A word of warning: this algorithm is SLOW. The function uses focal_hpc, so we highly recommend using it with a foreach engine running (e.g. use sfQuickInit()). Keep in mind this is a "dirty" parallel problem, so different chunks may execute at different speeds and have different memory footprints.

Author(s)

Jonathan A. Greenberg

See Also

`projectRaster`, `extract`, `aggregate`

rasterEngine *Engine for performing fast, easy-to-develop pixel and focal raster calculations with parallel processing capability.*

Description

Engine for performing fast, easy-to-develop pixel and focal raster calculations with parallel processing capability.

Usage

```
rasterEngine(x, fun = NULL, args = NULL, window_dims = c(1, 1),
  window_center = c(ceiling(window_dims[1]/2), ceiling(window_dims[2]/2)),
  filename = NULL, overwrite = FALSE, outformat = "raster",
  additional_header = "ENVI", datatype = "FLT8S", processing_unit = NULL,
  chunk_format = "array", minblocks = "max", blocksize = NULL,
  outbands = NULL, outfiles = NULL, setMinMax = FALSE,
  compileFunction = FALSE, debugmode = FALSE, .packages = NULL,
  clearworkers = TRUE, verbose = FALSE, ...)
```

Arguments

x	Raster*. A Raster* used as the input into the function. This is optional, as long as some Raster* was defined in "..."
fun	Function. A focal function to be applied to the image. See Details.
args	List. Arguments to pass to the function (see ?mapply). Note that the 'fun' should explicitly name the variables.
window_dims	Vector. The size of a processing window in col x row order. Be default, a single pixel (c(1,1).
window_center	Vector. The local coordinate of the center of a processing window. By default the middle of the processing window. CURRENTLY UNSUPPORTED.
filename	Character. Filename(s) of the output raster.
overwrite	Logical. Allow files to be overwritten? Default is FALSE.
outformat	Character. Outformat of the raster. Must be a format usable by hdr(). Default is 'raster'. CURRENTLY UNSUPPORTED.
processing_unit	Character. ("single"|"chunk") Will be auto-set if not specified ("chunk" for pixel-processing, "single" for focal processing). See Details.
chunk_format	Character. The format to send the chunk to the function. Can be "array" (default) or "raster".
minblocks	Numeric. The minimum number of chunks to divide the raster into for processing. Defaults to 1.
blocksize	Numeric. The size (in rows) for a block of data. If unset, rasterEngine will attempt to figure out an optimal blocksize.

`outbands` Numeric. If known, how many bands in each output file? Assigning this and outfiles will allow focal_hpc to skip the pre-check.

`outfiles` Numeric. If known, how many output files? Assigning this and outbands will allow focal_hpc to skip the pre-check.

`setMinMax` Logical. Run a setMinMax() on each output file after processing (this will slow the processing down). Default is FALSE.

`additional_header` Character. Create additional output headers for use with other GIS systems (see `hdr`). Set to NULL to suppress. Default is "ENVI".

`datatype` Character. Output number type. See ?dataType. Default is "FLT8S".

`compileFunction` Logical. Runs a byte-code compilation on the user function before running. Setting this to TRUE may speed up the execution. Default is FALSE.

`debugmode` Logical. If TRUE, the function will enter debug mode during the test phase. Note the inputs will be an array of size 2 columns, 1 row, and how ever many input bands.

`.packages` Character. A character vector of package names needed by the function (parallel mode only).

`clearworkers` Logical. Force the workers to clear all objects upon completing (releasing memory)? Default=TRUE.

`verbose` Logical. Enable verbose execution? Default is FALSE.

`...` Raster*s. Named variables pointing to Raster* objects. See Details.

Details

rasterEngine is designed to execute a function on one or multiple Raster* object(s) using foreach, to achieve parallel reads, executions and writes. Parallel random writes are achieved through the use of mmap, so individual image chunks can finish and write their outputs without having to wait for all nodes in the cluster to finish and then perform sequential writing. On Windows systems, random writes are possible but apparently not parallel writes. rasterEngine solves this by trying to write to a portion of the image file, and if it finds an error (a race condition occurs), it will simply retry the writes until it successfully finishes. On Unix-alikes, truly parallel writes should be possible.

rasterEngine is a convienence wrapper for `focal_hpc` and, in general, should be used instead of focal_hpc directly.

rasterEngine operates in two modes, which have different input and outputs to the function:

Pixel based processing:

1) If chunk_format=="array" (default), the input to the function should assume an array of dimensions x,y,z where x = the number of columns in a chunk, y = the number of rows in the chunk, and z = the number of bands in the chunk. If chunk_format=="raster", the input to the function will be a raster subset. Note that we are ordering the array using standards for geographic data, (columns, rows, bands), not how R usually thinks of arrays (rows, columns, bands).

2) If a single file is to be output from rasterEngine, the output of the function should always be an array with the x and y dimensions matching the input, and an arbitrary number of band outputs. Remember to order the dimensions as columns, rows, bands (x,y,z).

If a multiple file output is required, the output of the function should return a list of arrays each with an equivalent number of columns and rows. The first element of the list will be assigned to the first filename (if provided).

Local window processing:

1) The function should be written to process a SINGLE window at a time, given the dimensions of window_dims, so the input to the function should assume a window of dimensions window_dims with a local center defined by window_center. As with before, the input can be passed to the function as an array (suggested) or a small raster.

2) The output should be a single pixel value, so can either be a single value, or a vector (which is assumed to be multiple bands of a single pixel).

The speed of the execution when running in parallel will vary based on the specific setup, and may, indeed, be slower than a sequential execution (e.g. with calc()), particularly on smaller files. Note that by simply running sfQuickStop(), rasterEngine will run in sequential mode.

A fuller tutorial is available at http://publish.illinois.edu/jgrn/software-and-datasets/rasterengine-tutorial/

Author(s)

Jonathan A. Greenberg (<spatial.tools@estarcion.net>)

See Also

focal_hpc, foreach, mmap, dataType, hdr

Examples

```
# Pixel-based processing on one band:
apply_multiplier <- function(inraster,multiplier)
{
# Note that inraster is received by this function as a 3-d array (col,row,band)
multiplied_raster <- inraster * multiplier
return(multiplied_raster)
}

tahoe_lidar_highesthit <-
setMinMax(raster(system.file(
"external/tahoe_lidar_highesthit.tif", package="spatial.tools")))

# Note that you can use any parallel backend that can be registered with foreach.
# sfQuickInit() will spawn a PSOCK cluster using the parallel package.
# sfQuickInit(cpus=2)
tahoe_lidar_highesthit_multiplied <- rasterEngine(
inraster=tahoe_lidar_highesthit,
fun=apply_multiplier,
args=list(multiplier=3.28084))
# sfQuickStop()

## Not run:
# Pixel-based processing on more than one band:
ndvi <- function(GRNIR_image)
```

```
{
# The input array will have dim(GRNIR_image)[3] equal
# to 3, because the input image has three bands.
# Note: the following two lines return an array,
# so we don't need to manually set the dim(ndvi) at the
# end.  If we didn't use drop=FALSE, we'd need to
# coerce the output into a 3-d array before returning it.
red_band <- GRNIR_image[,,2,drop=FALSE]
nir_band <- GRNIR_image[,,3,drop=FALSE]
ndvi <- (nir_band - red_band)/(nir_band + red_band)
# The following is not needed in this case:
# dim(ndvi) <- c(dim(GRNIR_image)[1],dim(GRNIR_image)[2],1)
return(ndvi)
}

tahoe_highrez <- setMinMax(
brick(system.file("external/tahoe_highrez.tif", package="spatial.tools")))

sfQuickInit(cpus=2)
tahoe_ndvi <- rasterEngine(GRNIR_image=tahoe_highrez,fun=ndvi)
sfQuickStop()

# Focal-based processing:
mean_smoother <- function(inraster)
{
smoothed <- apply(inraster,3,mean)
return(smoothed)
}

# Apply the function to a 3x3 window:
sfQuickInit(cpus=2)
tahoe_3x3_smoothed <- rasterEngine(inraster=tahoe_highrez,fun=mean_smoother,window_dims=c(3,3))
sfQuickStop()

# Example with 7 x 7 window in full parallel mode:
sfQuickInit()
tahoe_7x7_smoothed <- rasterEngine(inraster=tahoe_highrez,fun=mean_smoother,window_dims=c(7,7))
sfQuickStop()

## End(Not run)
```

`raster_to_filenames` *Extract filenames from all Raster* objects.*

Description

Extract filenames from all Raster* objects.

Usage

```
raster_to_filenames(x, unique = FALSE)
```

Arguments

x	Raster*. A Raster* object (even one without values/in memory) to determine the filename(s).
unique	Logical. Only return unique filenames? If FALSE, one filename per layer.

Details

This is an expansion of filename() that allows for RasterStacks, in-memory Raster*s, and Raster*s without values. If a filename is not found, the entry will be "".

Value

Character vector of filenames.

Author(s)

Jonathan A. Greenberg

See Also

filename

Examples

```
{
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
raster_to_filenames(tahoe_highrez)
raster_to_filenames(tahoe_highrez,unique=TRUE)
nodata <- raster()
raster_to_filenames(nodata)
}
```

---

raster_to_GLT *Creates a geometry lookup (GLT) file from a raster*

Description

Creates a geometry lookup (GLT) file from a raster

Usage

```
raster_to_GLT(x)
```

Arguments

x	Raster* The raster containing the geographic information to be used as the basis for the GLT.

Details

This function produces a two-band brick where the pixel values for the first band are the column numbers, and the pixel values for the second band are the row numbers of the corresponding pixels in the input Raster* file.

Author(s)

Jonathan A. Greenberg

See Also

rowColFromCell, cellFromRow

Examples

```
tahoe_lidar_highesthit <-
raster(system.file("external/tahoe_lidar_highesthit.tif", package="spatial.tools"))
tahoe_lidar_highesthit_glt <- raster_to_GLT(tahoe_lidar_highesthit)
plot(tahoe_lidar_highesthit_glt)
setMinMax(tahoe_lidar_highesthit_glt)
```

---

raster_to_IGM *Creates an input geometry (IGM) file from a raster*

Description

Creates an input geometry (IGM) file from a raster

Usage

```
raster_to_IGM(x)
```

Arguments

x	Raster* The raster containing the geographic information to be used as the basis for the IGM.

Details

This function produces a two-band brick where the pixel values for the first band are the geographic x-coordinates, and the pixel values for the second band are the geographic y-coordinates of the corresponding pixels in the input Raster* file.

Author(s)

Jonathan A. Greenberg

See Also

xyFromCell, cellFromRow

Examples

```
tahoe_lidar_highesthit <-
raster(system.file("external/tahoe_lidar_highesthit.tif", package="spatial.tools"))
tahoe_lidar_highesthit_igm <- raster_to_IGM(tahoe_lidar_highesthit)
plot(tahoe_lidar_highesthit_igm)
setMinMax(tahoe_lidar_highesthit_igm)
```

---

remove_file_extension *remove_file_extension*

Description

Strips a file extension from a filename.

Usage

```
remove_file_extension(filename, extension_delimiter = ".")
```

Arguments

filename	Character. The input filename.
extension_delimiter	Character. The extension or extension delimiter (default ".") to remove.

Author(s)

Jonathan A. Greenberg <spatial.tools@estarcion.net>

Examples

```
myfilename="my.file.gri"
remove_file_extension(myfilename,".")
remove_file_extension(myfilename,".file.gri")
```

sfQuickInit *Quickly initializes a parallel cluster and registers it with foreach.*

Description

Quickly initializes a parallel cluster and registers it with foreach.

Usage

```
sfQuickInit(cpus, methods = FALSE, ...)
```

Arguments

cpus	Number of cpus. Will default to the max available cpus.
methods	Logical. Load the methods package? (if FALSE, faster startup). Default=FALSE.
...	parameters to pass to sfInit()

Details

(Even more) quickly start a parallel cluster with half of available cpus, parallel = TRUE, and type = "SOCK" and registers it with foreach.

Author(s)

Jonathan A. Greenberg

Examples

```
sfQuickInit(cpus=2)
sfQuickStop()
```

sfQuickStop *Quickly stops a parallel snowfall cluster and deregisters it from foreach.*

Description

Quickly stops a parallel snowfall cluster and deregisters it from foreach.

Usage

```
sfQuickStop(kill = FALSE, ...)
```

Arguments

kill	Logical. If TRUE, attempt to force-quit cluster if you get an out-of-control situation. Default=FALSE.
...	parameters to pass to sfStop()

Details

(Even more) quickly stop a snowfall cluster and sets foreach back to sequential mode via registerDoSEQ().

Author(s)

Jonathan A. Greenberg

Examples

```
sfQuickInit(cpus=2)
sfQuickStop()
```

---

spatial_sync_raster *Spatially Sync Rasters*

Description

Aligns ("syncs") a Raster to a reference Raster.

Usage

```
spatial_sync_raster(unsynced, reference, method = "ngb", size_only = FALSE,
  raster_size, verbose = FALSE, ...)
```

Arguments

unsynced	A Raster object to be aligned to the reference raster.
reference	A Raster object to be used as the reference for syncing. Syncing will use the reference's projection, resolution, and extent.
method	Method used to compute values for the new RasterLayer. Either 'ngb' (nearest neighbor) or 'bilinear' (bilinear interpolation).
verbose	verbose=TRUE gives feedback on the process (UNSUPPORTED AT PRESENT).
size_only	TODO
raster_size	TODO
...	parameters to be passed to writeRaster

Details

Uses bilinear or nearest neighbor resampling to align a raster to the extent and projection of a reference raster and match the resolution of the reference raster. This is helpful in preparing multiple files of different projections, resolutions, extents, and rotations for performing map algebra or change detection.

Value

Returns a RasterLayer, RasterBrick or RasterStack object synced to the reference raster object.

Author(s)

Jonathan A. Greenberg (<spatial.tools@estarcion.net>)

`spatial_sync_vector` *Matches a vector's projection to another vector or raster object's projection.*

Description

Matches a vector's projection to another vector or raster object's projection.

Usage

```
spatial_sync_vector(unsynced, reference, verbose = TRUE)
```

Arguments

`unsynced`	The vector to be projected.
`reference`	A raster or vector object who's projection the unsynced will be matched to.
`verbose`	Logical. Verbose logging?

Author(s)

Jonathan A. Greenberg (<spatial.tools@estarcion.net>)

Examples

```
tahoe_highrez_training_points_utm <- readOGR(
dsn=system.file("external", package="spatial.tools"),
layer="tahoe_highrez_training_points_utm")
print(projection(tahoe_highrez_training_points_utm))
tahoe_lidar_bareearth <-
raster(system.file("external/tahoe_lidar_bareearth.tif", package="spatial.tools"))
print(projection(tahoe_lidar_bareearth))
tahoe_highrez_training_points_utm_synced <-
spatial_sync_vector(tahoe_highrez_training_points_utm,tahoe_lidar_bareearth)
print(projection(tahoe_highrez_training_points_utm_synced))
```

subset_raster_by_names *Subsets a raster based on its layer names*

Description

Subsets a raster based on its layer names

Usage

```
subset_raster_by_names(x, subset_names, allnames = TRUE)
```

Arguments

x	A Raster* object to be subsetted.
subset_names	Character. A vector of layer names to use to subset the Raster*.
allnames	Logical. Make sure all subset_names are contained by x?

Author(s)

Jonathan A. Greenberg (<spatial.tools@estarcion.net>)

tahoe_highrez.tif *High resolution false color infrared image from the Lake Tahoe Basin.*

Description

High resolution false color infrared image from the Lake Tahoe Basin.

Author(s)

Jonathan A. Greenberg <spatial.tools@estarcion.net>

Examples

```
tahoe_highrez <- brick(system.file("external/tahoe_highrez.tif", package="spatial.tools"))
plotRGB(tahoe_highrez)
```

tahoe_highrez_training *Point and polygon files for use with spatial.tools*

Description

Point and polygon files for use with spatial.tools

Author(s)

Jonathan A. Greenberg <spatial.tools@estarcion.net>

Examples

```
tahoe_highrez_training_polygons <- readOGR(
dsn=system.file("external", package="spatial.tools"),layer="tahoe_highrez_training")
spplot(tahoe_highrez_training_polygons,zcol="Class")
tahoe_highrez_training_points <- readOGR(
dsn=system.file("external", package="spatial.tools"),layer="tahoe_highrez_training_points")
spplot(tahoe_highrez_training_points,zcol="SPECIES")
tahoe_highrez_training_points_utm <- readOGR(
dsn=system.file("external", package="spatial.tools"),
layer="tahoe_highrez_training_points_utm")
print(projection(tahoe_highrez_training_points_utm))
```

tahoe_lidar_bareearth.tif *Lidar-derived bare earth digital elevation model from the Lake Tahoe Basin.*

Description

Lidar-derived bare earth digital elevation model from the Lake Tahoe Basin.

Author(s)

Jonathan A. Greenberg <spatial.tools@estarcion.net>

Examples

```
tahoe_lidar_bareearth <-
raster(system.file("external/tahoe_lidar_bareearth.tif", package="spatial.tools"))
plot(tahoe_lidar_bareearth)
```

tahoe_lidar_highesthit.tif

Lidar-derived highest hit (aka canopy) digital elevation model from the Lake Tahoe Basin.

Description

Lidar-derived highest hit (aka canopy) digital elevation model from the Lake Tahoe Basin.

Author(s)

Jonathan A. Greenberg <spatial.tools@estarcion.net>

Examples

```
tahoe_lidar_highesthit <-
raster(system.file("external/tahoe_lidar_highesthit.tif", package="spatial.tools"))
plot(tahoe_lidar_highesthit)
```

which.max.simple *Location of Maximum Value*

Description

Locates the largest value of the input object.

Usage

```
which.max.simple(x, na.rm = TRUE, tie_value = "NA")
```

Arguments

x	a numeric object
na.rm	a logical indicating whether missing values should be removed.
tie_value	A character indicating how to deal with ties. Can be "NA" (returns an NA if a tie is found) or "random" (returns a single randomly chosen member of the ties if a tie is found) or "first" (returns the first class found).

Value

An integer of length 1 giving the index of the maximum of x or NA if the maximum of x is not unique, x has no non-NAs, or na.rm=F.

Author(s)

Jonathan A. Greenberg, Alison R. Mynsberge

See Also

which.max, which, max

Examples

```
## Not run:

x<-c(2:4,1,1,NA)
y<-c(4,1:3,NA,4)
## The index is only calculated for a unique maximum
which.max.simple(x)
which.max.simple(y)
which.max.simple(y,na.rm=FALSE)
which.max.simple(x,na.rm=FALSE)

## End(Not run)
```

---

which.min.simple *Location of Minimum Value*

Description

Locates the smallest value of the input object.

Usage

```
which.min.simple(x, na.rm = TRUE, tie_value = "NA")
```

Arguments

x	a numeric object
na.rm	a logical indicating whether missing values should be removed.
tie_value	A character indicating how to deal with ties. Can be "NA" (returns an NA if a tie is found) or "random" (returns a single randomly chosen member of the ties if a tie is found) or "first" (returns the first class found).

Value

An integer of length 1 giving the index of the minimum of x or NA if the minimum of x is not unique, x has no non-NAs, or na.rm=F.

Author(s)

Jonathan A. Greenberg, Alison R. Mynsberge

See Also

which.min, which, min

Examples

```
## Not run:

x<-c(4,1:3,NA,4)
y<-c(2:4,1,1,NA)
## The index is only calculated for a unique minimum
which.min.simple(x)
which.min.simple(y)
which.min.simple(y,na.rm=FALSE)
which.min.simple(x,na.rm=FALSE)

## End(Not run)
```

Index