Package ‘gdalcubes’

August 27, 2020

Title Earth Observation Data Cubes from Satellite Image Collections

Version 0.3.1

Date 2020-08-25

Description Processing collections of Earth observation images as on-demand multispectral, multitemporal raster data cubes. Users
define cubes by spatiotemporal extent, resolution, and spatial reference system and let 'gdalcubes' automatically apply cropping, reprojection, and
resampling using the 'Geospatial Data Abstraction Library' ('GDAL'). Implemented functions on data cubes include reduction over space and time,
applying arithmetic expressions on pixel band values, moving window aggregates over time, filtering by space, time, bands, and predicates on pixel values,
exporting data cubes as 'netCDF' or 'GeoTIFF' files, and plotting. The package implements lazy evaluation and
multithreading. All computational parts are implemented in C++, linking to the 'GDAL', 'netCDF', 'CURL', and 'SQLite' libraries.
See Appel and Pebesma (2019) <doi:10.3390/data4030092> for further details.

Depends R (>= 3.4)

Imports Rcpp, RcppProgress, jsonlite, ncdf4

License MIT + file LICENSE

URL https://github.com/appelmar/gdalcubes_R

BugReports https://github.com/appelmar/gdalcubes_R/issues/

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

LinkingTo Rcpp, RcppProgress

Suggests knitr, magrittr, rmarkdown, stars, magick, sf

VignetteBuilder knitr

Copyright file inst/COPYRIGHTS

NeedsCompilation yes

SystemRequirements cxx11, gdal, libgdal, libproj, netcdf4

Author Marius Appel [aut, cre] (<https://orcid.org/0000-0001-5281-3896>),
Edzer Pebesma [ctb] (<https://orcid.org/0000-0001-8049-7069>),
Roger Bivand [ctb],
Lewis Van Winkle [cph],
Ole Christian Eidheim [cph],
Howard Hinnant [cph],
Adrian Colomitchi [cph],
Florian Dang [cph],
Paul Thompson [cph],
Tomasz Kamiński [cph],
Dropbox, Inc. [cph]

Maintainer Marius Appel <marius.appel@uni-muenster.de>

Repository CRAN

Date/Publication 2020-08-27 11:40:16 UTC

R topics documented:

add_collection_format *Download and install an image collection format from a URL*

Description

Download and install an image collection format from a URL

Usage

```
add_collection_format(url, name = NULL)
```

Arguments

url	URL pointing to the collection format JSON file
name	optional name used to refer to the collection format

Details

By default, the collection format name will be derived from the basename of the URL.

Examples

```
add_collection_format(
  "https://raw.githubusercontent.com/appelmar/gdalcubes/dev/formats/Sentinel1_IW_GRD.json")
```

add_images *Add images to an existing image collection*

Description

This function adds provided files or GDAL dataset identifiers and to an existing image collection by extracting datetime, image identifiers, and band information according to the collection's format.

Usage

```
add_images(
  image_collection,
  files,
  unroll_archives = TRUE,
  out_file = "",
  quiet = FALSE
)
```

Arguments

`image_collection`
: image_collection object or path to an existing collection file

`files`
: character vector with paths to image files on disk or any GDAL dataset identifiers (including virtual file systems and higher level drivers or GDAL subdatasets)

`unroll_archives`
: automatically convert .zip, .tar archives and .gz compressed files to GDAL virtual file system dataset identifiers (e.g. by prepending /vsizip/) and add contained files to the list of considered files

`out_file`
: path to output file, an empty string (the default) will update the collection inplace, whereas images will be added to a new copy of the image collection at the given location otherwise.

`quiet`
: logical; if TRUE, do not print resulting image collection if return value is not assigned to a variable

Value

image collection proxy object, which can be used to create a data cube using `raster_cube`

Examples

```
L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                       ".TIF", recursive = TRUE, full.names = TRUE)
L8_col = create_image_collection(L8_files[1:12], "L8_L1TP")
add_images(L8_col, L8_files[13:24])
```

`animate` *Animate a data cube as an image time series*

Description

Animate a data cube as an image time series

Usage

```
animate(
  x,
  ...,
  fps = 1,
  loop = 0,
  width = dev.size(units = "px")[1],
  height = dev.size(units = "px")[2],
  save_as = NULL,
  plot = TRUE
)
```

Arguments

`x`	a data cube proxy object (class cube)
`...`	parameters passed to plot.cube
`fps`	frames per second of the animation
`loop`	how many iterations, 0 = infinite
`width`	width (in pixels) of the animation
`height`	height (in pixels) of the animation
`save_as`	character path where the animation shall be stored as a gif file
`plot`	logical; plot the animation (default is TRUE)

Details

Animations can be created for single band data cubes or RGB plots of multi-band data cubes (by providing the argument rgb) only.

See Also

`plot.cube`

Examples

```
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P16D")

animate(select_bands(raster_cube(L8.col, v), c("B02", "B03", "B04")), rgb=3:1,
        zlim=c(0,20000), fps=1, loop=1)

animate(select_bands(raster_cube(L8.col, v), c("B05")), col=terrain.colors, key.pos=1)
```

`apply_pixel` *Apply a function over (multi-band) pixels*

Description

This generic function applies a function on pixels of a data cube, an R array, or other classes if implemented.

Usage

```
apply_pixel(x, ...)
```

Arguments

x	input data
...	additional arguments passed to method implementations

Value

return value and type depend on the class of x

See Also

apply_pixel.cube

apply_pixel.array

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")

L8.col = image_collection(file.path(tempdir(), "L8.db"))
apply_pixel(raster_cube(L8.col, v), "(B05-B04)/(B05+B04)", "NDVI")



d <- c(4,16,128,128)
x <- array(rnorm(prod(d)), d)
y <- apply_pixel(x, function(v) {
  v[1] + v[2] + v[3] - v[4]
})
```

apply_pixel.array *Apply a function over pixels in a four-dimensional (band, time, y, x) array*

Description

Apply a function over pixels in a four-dimensional (band, time, y, x) array

Usage

```
## S3 method for class 'array'
apply_pixel(x, FUN, ...)
```

Arguments

x	four-dimensional input array with dimensions band, time, y, x (in this order)
FUN	function that receives a vector of band values in a one-dimensional array
...	further arguments passed to FUN

Details

FUN is expected to produce a numeric vector (or scalar) where elements are interpreted as new bands in the result.

Note

This is a helper function that uses the same dimension ordering as gdalcubes. It can be used to simplify the application of R functions e.g. over time series in a data cube.

Examples

```
d <- c(4,16,32,32)
x <- array(rnorm(prod(d)), d)
y <- apply_pixel(x, function(v) {
  v[1] + v[2] + v[3] - v[4]
})
dim(y)
```

apply_pixel.cube *Apply arithmetic expressions over all pixels of a data cube*

Description

Create a proxy data cube, which applies arithmetic expressions over all pixels of a data cube. Expressions may access band values by name.

Usage

```
## S3 method for class 'cube'
apply_pixel(x, expr, names = NULL, keep_bands = FALSE, ..., FUN)
```

Arguments

`x`	source data cube
`expr`	character vector with one or more arithmetic expressions (see Details)
`names`	optional character vector with the same length as expr to specify band names for the output cube
`keep_bands`	logical; keep bands of input data cube, defaults to FALSE, i.e. original bands will be dropped
`...`	not used
`FUN`	user-defined R function that is applied on all pixels (see Details)

Details

The function can either apply simple arithmetic C expressions given as a character vector (expr argument), or apply a custom R reducer function if FUN is provided.

In the former case, gdalcubes uses the tinyexpr library to evaluate expressions in C / C++, you can look at the library documentation to see what kind of expressions you can execute. Pixel band values can be accessed by name.

FUN receives values of the bands from one pixel as a (named) vector and should return a numeric vector with identical length for all pixels. Elements of the result vectors will be interpreted as bands in the result data cube.

Value

a proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

# 1. Apply a C expression
L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
L8.ndvi = apply_pixel(L8.cube, "(B05-B04)/(B05+B04)", "NDVI")
L8.ndvi


plot(L8.ndvi)


# 2. Apply a user defined R function
L8.ndvi.noisy = apply_pixel(L8.cube, names="NDVI_noisy",
   FUN=function(x) {
       rnorm(1, 0, 0.1) + (x["B05"]-x["B04"])/(x["B05"]+x["B04"])
   })
L8.ndvi.noisy


plot(L8.ndvi.noisy)
```

apply_time   *Apply a function over (multi-band) pixel time series*

Description

This generic function applies a function on pixel time series of a data cube, an R array, or other classes if implemented. The resulting object is expected to have the same spatial and temporal shape as the input, i.e., no reduction is performed.

Usage

```
apply_time(x, ...)
```

Arguments

x	input data
...	additional arguments passed to method implementations

Value

return value and type depend on the class of x

See Also

apply_time.cube

apply_time.array

Examples

```
# 1. input is data cube
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
L8.ndvi = apply_pixel(L8.cube, "(B05-B04)/(B05+B04)", "NDVI")

# Apply a user defined R function
apply_time(L8.ndvi, names="NDVI_residuals",
   FUN=function(x) {
      y = x["NDVI",]
      if (sum(is.finite(y)) < 3) {
         return(rep(NA,ncol(x)))
      }
      t = 1:ncol(x)
      return(predict(lm(y ~ t)) -  x["NDVI",])})

# 2. input is array
d <- c(4,16,32,32)
x <- array(rnorm(prod(d)), d)
z <- apply_time(x, function(v) {
  y = matrix(NA, ncol=ncol(v), nrow=2)
  y[1,] = (v[1,] + v[2,]) / 2
  y[2,] = (v[3,] + v[4,]) / 2
  y
})
```

```
dim(z)
```

apply_time.array *Apply a function over pixel time series in a four-dimensional (band, time, y, x) array*

Description

Apply a function over pixel time series in a four-dimensional (band, time, y, x) array

Usage

```
## S3 method for class 'array'
apply_time(x, FUN, ...)
```

Arguments

x	four-dimensional input array with dimensions band, time, y, x (in this order)
FUN	function that receives a vector of band values in a one-dimensional array
...	further arguments passed to FUN

Details

FUN is expected to produce a matrix (or vector if result has only one band) where rows are interpreted as new bands and columns represent time.

Note

This is a helper function that uses the same dimension ordering as gdalcubes. It can be used to simplify the application of R functions e.g. over time series in a data cube.

Examples

```
d <- c(4,16,32,32)
x <- array(rnorm(prod(d)), d)
z <- apply_time(x, function(v) {
  y = matrix(NA, ncol=ncol(v), nrow=2)
  y[1,] = (v[1,] + v[2,]) / 2
  y[2,] = (v[3,] + v[4,]) / 2
  y
})
dim(z)
```

apply_time.cube *Apply a user-defined R function over (multi-band) pixel time series*

Description

Create a proxy data cube, which applies a user-defined R function over all pixel time series of a data cube. In contrast to reduce_time, the time dimension is not reduced, i.e., resulting time series must have identical length as the input data cube but may contain a different number of bands / variables. Example uses of this function may include time series decompositions, cumulative sums / products, smoothing, sophisticated NA filling, or similar.

Usage

```
## S3 method for class 'cube'
apply_time(x, names = NULL, keep_bands = FALSE, FUN, ...)
```

Arguments

x	source data cube
names	optional character vector to specify band names for the output cube
keep_bands	logical; keep bands of input data cube, defaults to FALSE, i.e., original bands will be dropped
FUN	user-defined R function that is applied on all pixel time series (see Details)
...	not used

Details

FUN receives a single (multi-band) pixel time series as a matrix with rows corresponding to bands and columns corresponding to time. In general, the function must return a matrix with the same number of columns. If re result contains only a single band, it may alternatively return a vector with length identical to the length of the input time series (number of columns of the input).

Value

a proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
L8.ndvi = apply_pixel(L8.cube, "(B05-B04)/(B05+B04)", "NDVI")

# Apply a user defined R function
L8.ndvi.resid = apply_time(L8.ndvi, names="NDVI_residuals",
   FUN=function(x) {
      y = x["NDVI",]
      if (sum(is.finite(y)) < 3) {
         return(rep(NA,ncol(x)))
      }
      t = 1:ncol(x)
      return(predict(lm(y ~ t)) -  x["NDVI",])
   })
L8.ndvi.resid


plot(L8.ndvi.resid)
```

---

as_array *Convert a data cube to an in-memory R array*

Description

Convert a data cube to an in-memory R array

Usage

```
as_array(x)
```

Arguments

x	data cube

Value

Four dimensional array with dimensions band, t, y, x

Note

Depending on the data cube size, this function may require substantial amounts of main memory, i.e. it makes sense for small data cubes only.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-05"),
              srs="EPSG:32618", nx = 100, ny=100, dt="P1M")
as_array(select_bands(raster_cube(L8.col, v), c("B04", "B05")))
```

---

as_json *Query data cube properties*

Description

gdalcubes uses a graph (currently a tree) to serialize data cubes (including chains of cubes). This function gives a JSON representation, which will be communicated to gdalcubes_server instances to create identical cube instances remotely.

Usage

```
as_json(obj)
```

Arguments

obj	a data cube proxy object (class cube)

Value

A JSON string representing a graph (currently a tree) that can be used to create the same chain of gdalcubes operations.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-04"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
cat(as_json(select_bands(raster_cube(L8.col, v), c("B04", "B05"))))
```

bands   *Query data cube properties*

Description

Query data cube properties

Usage

```
bands(obj)
```

Arguments

obj	a data cube proxy object (class cube)

Value

A data.frame with rows representing the bands and columns representing properties of a band (name, type, scale, offset, unit)

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
bands(raster_cube(L8.col, v))
```

chunk_apply *Apply an R function on chunks of a data cube*

Description

Apply an R function on chunks of a data cube

Usage

```
chunk_apply(cube, f)
```

Arguments

cube	source data cube
f	R function to apply over all chunks

Details

This function internally creates a gdalcubes stream data cube, which streams data of a chunk to a new R process. For reading data, the function typically calls `x <-read_chunk_as_array()` which then results in a 4 dimensional (band, time, y, x) array. Similarly `write_chunk_from_array(x)` will write a result array as a chunk in the resulting data cube. The chunk size of the input cube is important to control how the function will be exposed to the data cube. For example, if you want to apply an R function over complete pixel time series, you must define the chunk size argument in `raster_cube` to make sure that chunk contain the correct parts of the data.

Value

a proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
                          srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
f <- function() {
  x <- read_chunk_as_array()
  out <- reduce_time(x, function(x) {
    cor(x[1,], x[2,], use="na.or.complete", method = "kendall")
  })
  write_chunk_from_array(out)
}
L8.cor = chunk_apply(L8.cube, f)
```

`collection_formats` *List predefined image collection formats*

Description

gdalcubes comes with some predefined collection formats e.g. to scan Sentinel 2 data. This function lists available formats including brief descriptions.

Usage

```
collection_formats(print = TRUE)
```

Arguments

`print`	logical; should available formats and their descriptions be printed nicely, defaults to TRUE

Details

Image collection formats define how individual files / GDAL datasets relate to an image collection, i.e., which bands they contain, to which image they belong, and how to derive aquisition date/time. They are described as a set of regular expressions in a JSON file and used by gdalcubes to extract this information from the paths and/or filenames.

Value

data.frame with columns `name` and `description` where the former describes the unique identifier that can be used in `create_image_collection` and the latter gives a brief description of the format.

Examples

```
collection_formats()
```

create_image_collection

Create an image collection from a set of GDAL datasets or files

Description

This function iterates over files or GDAL dataset identifiers and extracts datetime, image identifiers, and band information according to a given collection format.

Usage

```
create_image_collection(
  files,
  format = NULL,
  out_file = tempfile(fileext = ".sqlite"),
  date_time = NULL,
  band_names = NULL,
  use_subdatasets = FALSE,
  unroll_archives = TRUE,
  quiet = FALSE
)
```

Arguments

`files` character vector with paths to image files on disk or any GDAL dataset identifiers (including virtual file systems and higher level drivers or GDAL subdatasets)

`format` collection format, can be either a name to use predefined formats (as output from `collection_formats`) or a path to a custom JSON format description file

`out_file` optional name of the output SQLite database file, defaults to a temporary file

`date_time` vector with date/ time for files; can be of class character, Date, or POSIXct (argument is only applicable for image collections without collection format)

`band_names` character vector with band names, length must match the number of bands in provided files (argument is only applicable for image collections without collection format)

`use_subdatasets` logical; use GDAL subdatasets of provided files (argument is only applicable for image collections without collection format)

`unroll_archives` automatically convert .zip, .tar archives and .gz compressed files to GDAL virtual file system dataset identifiers (e.g. by prepending /vsizip/) and add contained files to the list of considered files

`quiet` logical; if TRUE, do not print resulting image collection if return value is not assigned to a variable

Details

An image collection is a simple SQLite database file that indexes and references existing image files / GDAL dataset identifiers.

Collections can be created in two different ways: First, if a collection format is specified (argument format), date/time, bands, and metadata are automatically extracted from provided files / GDAL datasets. Second, image collections can be created without collection format by manually specifying date/time of images (argument date_time) and (optional) names of bands. In this case, however, all provided images must contain the same bands. If this is not possible for a dataset, a collection format must be used.

Value

image collection proxy object, which can be used to create a data cube using raster_cube

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}
```

---

cube_view *Create or update a spatiotemporal data cube view*

Description

Data cube views define the shape of a cube, i.e., the spatiotemporal extent, resolution, and spatial reference system (srs). They are used to access image collections as on-demand data cubes. The data cube will filter images based on the view's extent, read image data at the defined resolution, and warp / reproject images to the target srs automatically.

Usage

```
cube_view(
  view,
  extent,
  srs,
  nx,
  ny,
  nt,
  dx,
  dy,
  dt,
  aggregation,
  resampling,
  keep.asp = TRUE
)
```

Arguments

`view`	if provided, update this cube_view object instead of creating a new data cube view where fields that are already set will be overwritten
`extent`	spatioptemporal extent as a list e.g. from `extent` or an image collection object, see Details
`srs`	target spatial reference system as a string; can be a proj4 definition, WKT, or in the form "EPSG:XXXX"
`nx`	number of pixels in x-direction (longitude / easting)
`ny`	number of pixels in y-direction (latitude / northing)
`nt`	number of pixels in t-direction
`dx`	size of pixels in x-direction (longitude / easting)
`dy`	size of pixels in y-direction (latitude / northing)
`dt`	size of pixels in time-direction, expressed as ISO8601 period string (only 1 number and unit is allowed) such as "P16D"
`aggregation`	aggregation method as string, defining how to deal with pixels containing data from multiple images, can be "min", "max", "mean", "median", or "first"
`resampling`	resampling method used in gdalwarp when images are read, can be "near", "bilinear", "bicubic" or others as supported by gdalwarp (see `https://gdal.org/programs/gdalwarp.html`)
`keep.asp`	if TRUE, derive ny or dy automatically from nx or dx (or vice versa) based on the aspect ratio of the spatial extent

Details

The `extent` argument expects a simple list with elementes `left`, `right`, `bottom`, `top`, `t0` (start date/time), `t1` (end date/time) or an image collection object. In the latter case, the `extent` function is automatically called on the image collection object to get the full spatiotemporal extent of the collection. In the former case, datetimes are expressed as ISO8601 datetime strings.

The function can be used in two different ways. First, it can create data cube views from scratch by defining the extent, the spatial reference system, and for each dimension either the cell size (dx, dy, dt) or the total number of cells (nx, ny, nt). Second, the function can update an existing data cube view by overwriting specific fields. In this case, the extent or some elements of the extent may be missing.

In some cases, the extent of the view is automatically extended if the provided resolution would end within a pixel. For example, if the spatial extent covers an area of 1km x 1km and dx = dy = 300m, the extent would be enlarged to 1.2 km x 1.2km. The alignment will be reported to the user in a diagnostic message.

Value

A list with data cube view properties

Examples

```
L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                       ".TIF", recursive = TRUE, full.names = TRUE)
L8.col = create_image_collection(L8_files, "L8_L1TP")

# 1. Create a new data cube view specification
v = cube_view(extent=extent(L8.col,"EPSG:4326"), srs="EPSG:4326", dt="P1M",
          nx=1000, ny=500, aggregation = "mean", resampling="bilinear")
v

# 2. overwrite parts of an existing data cube view
vnew = cube_view(v, dt="P1M")
```

---

dim.cube *Query data cube properties*

Description

Query data cube properties

Usage

```
## S3 method for class 'cube'
dim(x)
```

Arguments

x	a data cube proxy object (class cube)

Value

size of a data cube (number of cells) as integer vector in the order t, y, x

See Also

size

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
dim(raster_cube(L8.col, v))
```

dimensions *Query data cube properties*

Description

Query data cube properties

Usage

```
dimensions(obj)
```

Arguments

obj	a data cube proxy object (class cube)

Details

Elements of the returned list represent individual dimensions with properties such as dimension boundaries, names, and chunk size stored as inner lists

Value

Dimension information as a list

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
dimensions(raster_cube(L8.col, v))
```

dimension_bounds *Query coordinate bounds for all dimensions of a data cube*

Description

Dimension values give the coordinates bounds the spatial and temporal axes of a data cube.

Usage

```
dimension_bounds(obj, datetime_unit = NULL)
```

Arguments

obj	a data cube proxy (class cube)
datetime_unit	unit used to format values in the datetime dimension, one of "Y", "m", "d", "H", "M", "S", defaults to the unit of the cube.

Value

list with elements t,y,x, each a list with two elements, start and end

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
dimension_bounds(raster_cube(L8.col, v))
```

dimension_values *Query coordinate values for all dimensions of a data cube*

Description

Dimension values give the coordinates along the spatial and temporal axes of a data cube.

Usage

```
dimension_values(obj, datetime_unit = NULL)
```

Arguments

`obj` a data cube proxy (class cube), or a data cube view object

`datetime_unit` unit used to format values in the datetime dimension, one of "Y", "m", "d", "H", "M", "S", defaults to the unit of the cube.

Value

list with elements t,y,x

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
dimension_values(raster_cube(L8.col, v))
```

`extent` *Derive the spatiotemporal extent of an image collection*

Description

Derive the spatiotemporal extent of an image collection

Usage

```
extent(x, srs = "EPSG:4326")
```

Arguments

`x` image collection proxy object

`srs` target spatial reference system

Value

a list with elements `left`, `right`, `bottom`, `top`, `t0` (start date/time), and `t1` (end date/time)

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
extent(L8.col,"EPSG:32618")
cube_view(extent=extent(L8.col,"EPSG:32618"),
          srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
```

fill_time *Fill NA data cube pixels by simple time series interpolation*

Description

Create a proxy data cube, which fills NA pixels of a data cube by nearest neighbor or linear time series interpolation.

Usage

```
fill_time(cube, method = "near")
```

Arguments

cube	source data cube
method	interpolation method, can be "near" (nearest neighbor), "linear" (linear interpolation), "locf" (last observation carried forward), or "nocb" (next observation carried backward)

Details

Please notice that completely empty (NA) time series will not be filled, i.e. the result cube might still contain NA values.

Value

a proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}
L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P3M", aggregation = "median")
L8.cube = raster_cube(L8.col, v, mask=image_mask("BQA", bits=4, values=16))
L8.rgb = select_bands(L8.cube, c("B02", "B03", "B04"))
L8.filled = fill_time(L8.rgb, "linear")
L8.filled


plot(L8.filled, rgb=3:1, zlim=c(5000,12000))
```

`filter_geom` *Filter data cube pixels by a polygon*

Description

Create a proxy data cube, which filters pixels by a spatial (multi)polygon For all pixels whose center is within the polygon, the original

Usage

```
filter_geom(cube, geom, srs = NULL)
```

Arguments

cube	source data cube
geom	either a WKT string, or an sfc or sfg object (sf package)
srs	string identifier of the polygon's coordinate reference system understandable for GDAL

Details

The resulting data cube will not be cropped but pixels outside of the polygon will be set to NAN.

If geom is provided as an sfc object with length > 1, geometries will be combined with `sf::st_combine()` before.

The geometry is automatically transformed to the data cube's spatial reference system if needed.

Value

a proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
L8.ndvi = apply_pixel(L8.cube, "(B05-B04)/(B05+B04)", "NDVI")
WKT = gsub(pattern='\\n',replacement="",x =
  "Polygon ((-74.3541 40.9254,
             -73.9813 41.2467,
             -73.9997 41.4400,
             -74.5362 41.1795,
             -74.6286 40.9137,
             -74.3541 40.9254))")
L8.ndvi.filtered = filter_geom(L8.ndvi, WKT, "EPSG:4326")
L8.ndvi.filtered

plot(L8.ndvi.filtered)
```

filter_pixel *Filter data cube pixels by a user-defined predicate on band values*

Description

Create a proxy data cube, which evaluates a predicate over all pixels of a data cube. For all pixels that fulfill the predicate, the original band values are returned. Other pixels are simply filled with NANs. The predicate may access band values by name.

Usage

```
filter_pixel(cube, pred)
```

Arguments

`cube` source data cube

`pred` predicate to be evaluated over all pixels

Details

gdalcubes uses and extends the tinyexpr library to evaluate expressions in C / C++, you can look at the library documentation to see what kind of expressions you can execute. Pixel band values can be accessed by name.

Value

a proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
L8.ndvi = apply_pixel(L8.cube, "(B05-B04)/(B05+B04)", "NDVI")
L8.ndvi.filtered = filter_pixel(L8.ndvi, "NDVI > 0.5")
L8.ndvi.filtered

plot(L8.ndvi.filtered)
```

`gdalcubes` *gdalcubes: Earth Observation Data Cubes from Satellite Image Collections*

Description

Processing collections of Earth observation images as on-demand multispectral, multitemporal raster data cubes. Users define cubes by spatiotemporal extent, resolution, and spatial reference system and let 'gdalcubes' automatically apply cropping, reprojection, and resampling using the 'Geospatial Data Abstraction Library' ('GDAL'). Implemented functions on data cubes include reduction over space and time, applying arithmetic expressions on pixel band values, moving window aggregates over time, filtering by space, time, bands, and predicates on pixel values, exporting data cubes as 'netCDF' or 'GeoTIFF' files, and plotting. The package implements lazy evaluation and multithreading. All computational parts are implemented in C++, linking to the 'GDAL', 'netCDF', 'CURL', and 'SQLite' libraries. See Appel and Pebesma (2019) <doi:10.3390/data4030092> for further details.

---

gdalcubes_debug_output

Enable or disable debug output from the gdalcubes C++ library

---

Description

Enable or disable debug output from the gdalcubes C++ library

Usage

```
gdalcubes_debug_output(debug = TRUE)
```

Arguments

debug	logical, TRUE if you want debug messages

Note

THIS FUNCTION IS DEPRECATED AND IS GOING TO BE REPLACED BY `gdalcubes_options`.

Examples

```
gdalcubes_debug_output(TRUE)
gdalcubes_debug_output(FALSE)
```

`gdalcubes_gdalformats` *Get available GDAL drivers*

Description

Get available GDAL drivers

Usage

```
gdalcubes_gdalformats()
```

Examples

```
gdalcubes_gdalformats()
```

`gdalcubes_gdalversion` *Get the GDAL version used by gdalcubes*

Description

Get the GDAL version used by gdalcubes

Usage

```
gdalcubes_gdalversion()
```

Examples

```
gdalcubes_gdalversion()
```

`gdalcubes_gdal_has_geos` *Check if GDAL was built with GEOS*

Description

Check if GDAL was built with GEOS

Usage

```
gdalcubes_gdal_has_geos()
```

Examples

```
gdalcubes_gdal_has_geos()
```

gdalcubes_options *Set or read global options of the gdalcubes package*

Description

Set global package options to change the default behavior of gdalcubes. These include how many threads are used to process data cubes, how created netCDF files are compressed, and whether or not debug messages should be printed.

Usage

```
gdalcubes_options(
  ...,
  threads,
  ncdf_compression_level,
  debug,
  cache,
  ncdf_write_bounds,
  use_overview_images
)
```

Arguments

`...` not used

`threads` number of threads used to process data cubes

`ncdf_compression_level`
integer; compression level for created netCDF files, 0=no compression, 1=fast compression, 9=small compression

`debug` logical; print debug messages

`cache` logical; TRUE if temporary data cubes should be cached to support fast reprocessing of the same cubes

`ncdf_write_bounds`
logical; write dimension bounds as additional variables in netCDF files

`use_overview_images`
logical; if FALSE, all images are read on original resolution and existing overviews will be ignored

Details

Data cubes can be processed in parallel where one thread processes one chunk at a time. Setting more threads than the number of chunks of a cube thus has no effect and will not further reduce computation times.

Caching has no effect on disk or memory consumption, it simply tries to reuse existing temporary files where possible. For example, changing only parameters to `plot` will not require rerunning the full data cube operation chain.

Passing no arguments will return the current options as a list.

Examples

```
gdalcubes_options(threads=4) # set the number of threads
gdalcubes_options() # print current options
```

gdalcubes_set_ncdf_compression   *Set compression level for netCDF files produced by gdalcubes*

Description

Set compression level for netCDF files produced by gdalcubes

Usage

```
gdalcubes_set_ncdf_compression(level = 2)
```

Arguments

`level`	integer; compression level, 0 = no compression, 1=fast compression, 9=small compression

Note

THIS FUNCTION IS DEPRECATED AND IS GOING TO BE REPLACED BY `gdalcubes_options`.

Examples

```
gdalcubes_set_ncdf_compression(9) # maximum compression
gdalcubes_set_ncdf_compression(0) # no compression
```

gdalcubes_set_threads   *Set the number of threads for parallel data cube processing*

Description

Data cubes can be processed in parallel where one thread processes one chunk at a time. Setting more threads than the number of chunks of a cube thus has no effect and will not further reduce computation times.

Usage

```
gdalcubes_set_threads(n = 1)
```

Arguments

`n`	number of threads

Note

THIS FUNCTION IS DEPRECATED AND IS GOING TO BE REPLACED BY `gdalcubes_options`.

Examples

```
gdalcubes_set_threads(1)
```

---

gdalcubes_use_cache *Enable or disable caching of cubes.*

Description

Enable or disable caching of cubes.

Usage

```
gdalcubes_use_cache(enable = TRUE)
```

Arguments

`enable`	logical, TRUE if you want to use the data cube cache

Details

Caching has no effect on disk or memory consumption, it simply tries to reuse existing temporary files where possible. For example, changing only parameters to `plot` will not require rerunning the full data cube operation chain.

Note

THIS FUNCTION IS DEPRECATED AND IS GOING TO BE REPLACED BY `gdalcubes_options`.

Examples

```
gdalcubes_use_cache(FALSE)
```

gdalcubes_version *Query gdalcubes version information*

Description

Query gdalcubes version information

Usage

```
gdalcubes_version()
```

Value

List with gdalcubes library version information

Examples

```
gdalcubes_version()
```

image_collection *Load an existing image collection from a file*

Description

This function will load an image collection from an SQLite file. Image collection files index and reference existing imagery. To create a collection from files on disk, use `create_image_collection`.

Usage

```
image_collection(path)
```

Arguments

`path`	path to an existing image collection file

Value

an image collection proxy object, which can be used to create a data cube using `raster_cube`

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
L8.col
```

image_mask *Create a mask for images in a raster data cube*

Description

Create an image mask based on a band and provided values to filter pixels of images read by `raster_cube`

Usage

```
image_mask(
  band,
  min = NULL,
  max = NULL,
  values = NULL,
  bits = NULL,
  invert = FALSE
)
```

Arguments

`band` name of the mask band

`min` minimum value, values between `min` and `max` will be masked

`max` maximum value, values between `min` and `max` will be masked

`values` numeric vector; specific values that will be masked.

`bits` for bitmasks, extract the given bits (integer vector) with a bitwise AND before filtering the mask values, bit indexes are zero-based

`invert` logical; invert mask

Details

Values of the selected mask band can be based on a range (by passing `min` and `max`) or on a set of values (by passing `values`). By default pixels with mask values contained in the range or in the values are masked out, i.e. set to NA. Setting `invert = TRUE` will invert the masking behavior. Passing `values` will override `min` and `max`.

Note

Notice that masks are applied per image while reading images as a raster cube. They can be useful to eliminate e.g. cloudy pixels before applying the temporal aggregation to merge multiple values for the same data cube pixel.

Examples

```
image_mask("SCL", values = c(3,8,9)) # Sentinel 2 L2A: mask cloud and cloud shadows
image_mask("BQA", bits=4, values=16) # Landsat 8: mask clouds
image_mask("B10", min = 8000, max=65000)
```

join_bands *Join bands of two identically shaped data cubes*

Description

Create a proxy data cube, which joins the bands of two identically shaped data cubes. The resulting cube will have bands from both input cubes.

Usage

```
join_bands(cube_list, cube_names = NULL)
```

Arguments

cube_list	a list with two or more source data cubes
cube_names	list or character vector with optional name prefixes for bands in the output data cube (see Details)

Details

The number of provided cube_names must match the number of provided input cubes. If no cube_names are provided, bands of the output cube will adopt original names from the input cubes (without any prefix). If any two of the input bands have identical names, prefixes default prefixes ("X1", "X2", ...) will be used.

Value

proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931, t0="2018-01", t1="2018-05"),
                          srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube.b04 = select_bands(raster_cube(L8.col, v), c("B04"))
L8.cube.b05 = select_bands(raster_cube(L8.col, v), c("B05"))
join_bands(list(L8.cube.b04,L8.cube.b05))

plot(join_bands(list(L8.cube.b04,L8.cube.b05)))
```

---

memsize *Query data cube properties*

Description

Query data cube properties

Usage

```
memsize(obj, unit = "MiB")
```

Arguments

obj	a data cube proxy object (class cube)
unit	Unit of data size, can be "B", "KB", "KiB", "MB", "MiB", "GB", "GiB", "TB", "TiB", "PB", "PiB"

Value

Total data size of data cube values expressed in the given unit

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
memsize(raster_cube(L8.col, v))
```

names.cube *Query data cube properties*

Description

Query data cube properties

Usage

```
## S3 method for class 'cube'
names(x)
```

Arguments

x	a data cube proxy object (class cube)

Value

Band names as character vector

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
names(raster_cube(L8.col, v))
```

nbands *Query data cube properties*

Description

Query data cube properties

Usage

```
nbands(obj)
```

Arguments

obj	a data cube proxy object (class cube)

Value

Number of bands

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
nbands(raster_cube(L8.col, v))
```

nt *Query data cube properties*

Description

Query data cube properties

Usage

```
nt(obj)
```

Arguments

`obj` a data cube proxy object (class cube)

Value

Number of pixels in the time dimension

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
nt(raster_cube(L8.col, v))
```

`nx` *Query data cube properties*

Description

Query data cube properties

Usage

```
nx(obj)
```

Arguments

`obj` a data cube proxy object (class cube)

Value

Number of pixels in the x dimension

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
nx(raster_cube(L8.col, v))
```

---

ny *Query data cube properties*

Description

Query data cube properties

Usage

```
ny(obj)
```

Arguments

obj	a data cube proxy object (class cube)

Value

Number of pixels in the y dimension

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
ny(raster_cube(L8.col, v))
```

pack_minmax — *Helper function to define packed data exports by min / max values*

Description

This function can be used to define packed exports in `write_ncdf` and `write_tif`. It will generate scale and offset values with maximum precision (unless simplify=TRUE).

Usage

```
pack_minmax(type = "int16", min, max, simplify = FALSE)
```

Arguments

`type`	target data type of packed values (one of "uint8", "uint16", "uint32", "int16", or "int32")
`min`	numeric; minimum value(s) of original values, will be packed to the 2nd lowest value of the target data type
`max`	numeric; maximum value(s) in original scale, will be packed to the highest value of the target data type
`simplify`	logical; round resulting scale and offset to power of 10 values

Details

Nodata values will be mapped to the lowest value of the target data type.

Arguments min and max must have length 1 or length equal to the number of bands of the data cube to be exported. In the former case, the same values are used for all bands of the exported target cube, whereas the latter case allows to use different ranges for different bands.

Note

Using simplify=TRUE will round scale values to the next smaller power of 10.

Examples

```
ndvi_packing = pack_minmax(type="int16", min=-1, max=1)
ndvi_packing
```

plot.cube *Plot a gdalcubes data cube*

Description

Plot a gdalcubes data cube

Usage

```
## S3 method for class 'cube'
plot(
  x,
  y,
  ...,
  nbreaks = 11,
  breaks = NULL,
  col = grey(1:(nbreaks - 1)/nbreaks),
  key.pos = NULL,
  bands = NULL,
  t = NULL,
  rgb = NULL,
  zlim = NULL,
  periods.in.title = TRUE,
  join.timeseries = FALSE,
  axes = TRUE,
  ncol = NULL,
  nrow = NULL,
  na.color = "#AAAAAA"
)
```

Arguments

`x`	a data cube proxy object (class cube)
`y`	__not used__
`...`	further arguments passed to `image.default`
`nbreaks`	number of breaks, should be one more than the number of colors given
`breaks`	actual breaks used to assign colors to values; if missing, the function subsamples values and uses equally sized intervals between min and max or zlim[0] and zlim[1] if defined
`col`	color definition, can be a character vector with nbreaks - 1 elements or a function such as `heat.colors`
`key.pos`	position for the legend, 1 (bottom), 2 (left), 3 (top), or 4 (right). If NULL (the default), do not plot a legend.
`bands`	integer vector with band numbers to plot (this must be band numbers, not band names)

`t` integer vector with time indexes to plot (this must be time indexes, not date / time)

`rgb` bands used to assign RGB color channels, vector of length 3 (this must be band numbers, not band names)

`zlim` vector of length 2, defining the minimum and maximum values to either derive breaks, or define black and white values in RGB plots

`periods.in.title` logical value, if TRUE, the title of plots includes the datetime period length as ISO 8601 string

`join.timeseries` logical, for pure time-series plots, shall time series of multiple bands be plotted in a single plot (with different colors)?

`axes` logical, if TRUE, plots include axes

`ncol` number of columns for arranging plots with `layout()`, see Details

`nrow` number of rows for arranging plots with `layout()`, see Details

`na.color` color used to plot NA pixels

Details

The style of the plot depends on provided parameters and on the shape of the cube, i.e., whether it is a pure time series and whether it contains multiple bands or not. Multi-band, multi-temporal images will be arranged with `layout()` such that bands are represented by the x axis and time is represented by the y axis. Time series plots can be combined to a single plot by setting `join.timeseries = TRUE`. For other cases, a default arrangement of the plots is derived, trying to reach a square overall plot. The layout can be controlled with `ncol` and `nrow`, which define the number of rows and columns in the plot layout. Typically, only one of `ncol` and `nrow` is provided. For multi-band, multi-temporal plots, the actual number of rows or columns can be less if the input cube has less bands or time slices.

Note

If caching is enabled for the package (see `gdalcubes_use_cache`), repeated calls of plot for the same data cube will not reevaluate the cube. Instead, the temporary result file will be reused, if possible.

Some parts of the function have been copied from the stars package (c) Edzer Pebesma

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
```

```
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")

plot(select_bands(raster_cube(L8.col, v), c("B02", "B03", "B04")), rgb=3:1)

L8.cube = select_bands(raster_cube(L8.col, v), c("B04", "B05"))
L8.ndvi = apply_pixel(L8.cube, "(B05-B04)/(B05+B04)", "NDVI")
plot(reduce_time(L8.ndvi, "median(NDVI)"), key.pos=1, zlim=c(0,1))
```

---

print.cube *Print data cube information*

Description

Prints information about the dimensions and bands of a data cube.

Usage

```
## S3 method for class 'cube'
print(x, ...)
```

Arguments

x	Object of class "cube"
...	Further arguments passed to the generic print function

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
print(raster_cube(L8.col, v))
```

`print.cube_view` *Print data cube view information*

Description

Prints information about a data cube view, including its dimensions, spatial reference, aggregation method, and resampling method.

Usage

```
## S3 method for class 'cube_view'
print(x, ...)
```

Arguments

`x`	Object of class "cube_view"
`...`	Further arguments passed to the generic print function

Examples

```
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
print(v)
```

`print.image_collection` *Print image collection information*

Description

Prints information about images in an image collection.

Usage

```
## S3 method for class 'image_collection'
print(x, ..., n = 6)
```

Arguments

`x`	Object of class "image_collection"
`...`	Further arguments passed to the generic print function
`n`	Number of images for which details are printed

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
print(L8.col)
```

proj4 *Query data cube properties*

Description

Query data cube properties

Usage

```
proj4(obj)
```

Arguments

obj	a data cube proxy object (class cube)

Value

The spatial reference system expressed as proj4 string

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
proj4(raster_cube(L8.col, v))
```

query_points *Query data cube values at irregular spatiotemporal points*

Description

This function will overlay provided spatiotemporal points with a data cube and return all band values of the cells for each query point, as a data.frame where rows correspond to points and columns represent bands. If needed, point coordinates are automatically transformed to the SRS of the data cube.

Usage

```
query_points(x, px, py, pt, srs)
```

Arguments

`x`	source data cube
`px`	vector of x coordinates
`py`	vector of y coordinates
`pt`	vector of date/ time coordinates
`srs`	spatial reference system string identifer (as GDAL understands)

Details

Date and time of the query points can be provided as vector of class character, Date, or POSIXct.

Value

a data.frame with one row per point and one column per data cube band or variable

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}
L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01-01", t1="2018-12-02"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P14D")
L8.cube = raster_cube(L8.col, v)
L8.rgb = select_bands(L8.cube, c("B02", "B03", "B04"))

x = seq(from = 388941.2, to = 766552.4, length.out = 10)
y = seq(from = 4345299, to = 4744931, length.out = 10)
```

```
t = seq(as.Date("2018-01-01"), as.Date("2018-12-02"), length.out = 10 )

query_points(L8.rgb, x, y, t, srs(L8.rgb))
```

query_timeseries *Query data cube timeseries at irregular spatial points*

Description

This function will overlay provided spatial points with a data cube and return time series of all bands of the cells for each query point, as a list of data.frame (one data frame per band) where rows correspond to points and columns represent time. If needed, point coordinates are automatically transformed to the SRS of the data cube.

Usage

```
query_timeseries(x, px, py, srs)
```

Arguments

x	source data cube
px	vector of x coordinates
py	vector of y coordinates
srs	spatial reference system string identifer (as GDAL understands)

Value

a list of data.frames (one per band / variable) with one row per point and one column per data cube time slice

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}
L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01-01", t1="2018-12-02"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P14D")
L8.cube = raster_cube(L8.col, v)
L8.rgb = select_bands(L8.cube, c("B02", "B03", "B04"))

x = seq(from = 388941.2, to = 766552.4, length.out = 10)
```

```
y = seq(from = 4345299, to = 4744931, length.out = 10)

query_timeseries(L8.rgb, x, y, srs(L8.rgb))
```

raster_cube *Create a data cube from an image collection*

Description

Create a proxy data cube, which loads data from a given image collection according to a data cube view

Usage

```
raster_cube(image_collection, view, mask = NULL, chunking = c(1, 256, 256))
```

Arguments

image_collection
: Source image collection as from image_collection or create_image_collection

view
: A data cube view defining the shape (spatiotemporal extent, resolution, and spatial reference), if missing, a default overview is used

mask
: mask pixels of images based on band values, see image_mask

chunking
: Vector of length 3 defining the size of data cube chunks in the order time, y, x.

Details

The following steps will be performed when the data cube is requested to read data of a chunk:

1. Find images from the input collection that intersect with the spatiotemporal extent of the chunk 2. For all resulting images, apply gdalwarp to reproject, resize, and resample to an in-memory GDAL dataset 3. Read the resulting data to the chunk buffer and optionally apply a mask on the result 4. Update pixel-wise aggregator (as defined in the data cube view) to combine values of multiple images within the same data cube pixels

Value

A proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
raster_cube(L8.col, v)

 # using a mask on the Landsat quality bit band to filter out clouds
 raster_cube(L8.col, v, mask=image_mask("BQA", bits=4, values=16))
```

raster_cube_dummy *Create a dummy data cube with a fill value*

Description

Create a data cube with a constant fill value for one or more bands from a data cube view. Use this cube for testing.

Usage

```
raster_cube_dummy(view, nbands = 1, fill = 1, chunking = c(16, 256, 256))
```

Arguments

view	a data cube view defining the shape (spatiotemporal extent, resolution, and spatial reference)
nbands	number of bands
fill	fill value
chunking	vector of length 3 defining the size of data cube chunks in the order time, y, x.

Value

a proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
v = cube_view(extent=list(left=388941.2, right=766552.4,
                         bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
             srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube_dummy(v, 1, 2.345)

plot(L8.cube, zlim=c(0,4))
```

---

read_chunk_as_array *Read chunk data of a data cube from stdin or a file*

Description

This function can be used within function passed to chunk_apply in order to read a data cube chunk as a four-dimensional R array. It works only for R processes, which have been started from the gdalcubes C++ library. The resulting array has dimensions band, time, y, x (in this order).

Usage

```
read_chunk_as_array(with.dimnames = TRUE)
```

Arguments

with.dimnames	if TRUE, the resulting array will contain dimnames with coordinates, datetime, and band names

Value

four-dimensional array

Note

Call this function ONLY from a function passed to chunk_apply.

This function only works in R sessions started from gdalcubes streaming.

Examples

```

# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
```

```
v = cube_view(extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
                          srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
f <- function() {
  x <- read_chunk_as_array()
  out <- reduce_time(x, function(x) {
    cor(x[1,], x[2,], use="na.or.complete", method = "kendall")
  })
  write_chunk_from_array(out)
}
L8.cor = chunk_apply(L8.cube, f)
plot(L8.cor, zlim=c(0,1), key.pos=1)
```

reduce_space *Reduce multidimensional data over space*

Description

This generic function applies a reducer function over a data cube, an R array, or other classes if implemented.

Usage

```
reduce_space(x, ...)
```

Arguments

x	object to be reduced
...	further arguments passed to specific implementations

Value

return value and type depend on the class of x

See Also

reduce_space.cube

reduce_space.array

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
reduce_space(raster_cube(L8.col, v) , "median(B02)")


d <- c(4,16,32,32)
x <- array(rnorm(prod(d)), d)
y <- reduce_space(x, function(v) {
  apply(v, 1, mean)
})
```

reduce_space.array — *Apply a function over space and bands in a four-dimensional (band, time, y, x) array and reduce spatial dimensions*

Description

Apply a function over space and bands in a four-dimensional (band, time, y, x) array and reduce spatial dimensions

Usage

```
## S3 method for class 'array'
reduce_space(x, FUN, ...)
```

Arguments

x four-dimensional input array with dimensions band, time, y, x (in this order)

FUN function which receives one spatial slice in a three-dimensional array with dimensions bands, y, x as input

... further arguments passed to FUN

Details

FUN is expected to produce a numeric vector (or scalar) where elements are interpreted as new bands in the result.

Note

This is a helper function that uses the same dimension ordering as gdalcubes streaming. It can be used to simplify the application of R functions e.g. over spatial slices in a data cube.

Examples

```
d <- c(4,16,32,32)
x <- array(rnorm(prod(d)), d)
# reduce individual bands over spatial slices
y <- reduce_space(x, function(v) {
  apply(v, 1, mean)
})
dim(y)
```

`reduce_space.cube` *Reduce a data cube over spatial (x,y or lat,lon) dimensions*

Description

Create a proxy data cube, which applies one or more reducer functions to selected bands over spatial slices of a data cube

Usage

```
## S3 method for class 'cube'
reduce_space(x, expr, ..., FUN, names = NULL)
```

Arguments

`x` source data cube

`expr` either a single string, or a vector of strings defining which reducers will be applied over which bands of the input cube

`...` optional additional expressions (if `expr` is not a vector)

`FUN` a user-defined R function applied over pixel time series (see Details)

`names` character vector; if FUN is provided, names can be used to define the number and name of output bands

Details

Notice that expressions have a very simple format: the reducer is followed by the name of a band in parantheses. You cannot add more complex functions or arguments.

Possible reducers currently are "min", "max", "sum", "prod", "count", "mean", "median", "var", "sd".

Value

proxy data cube object

Note

Implemented reducers will ignore any NAN values (as na.rm=TRUE does).

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.b02 = select_bands(L8.cube, c("B02"))
L8.b02.median = reduce_space(L8.b02, "median(B02)")
L8.b02.median

plot(L8.b02.median, key.pos=1)
```

---

reduce_time *Reduce multidimensional data over time*

Description

This generic function applies a reducer function over a data cube, an R array, or other classes if implemented.

Usage

```
reduce_time(x, ...)
```

Arguments

x	object to be reduced
...	further arguments passed to specific implementations

Value

return value and type depend on the class of x

See Also

reduce_time.cube

reduce_time.array

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
reduce_time(raster_cube(L8.col, v) , "median(B02)", "median(B03)", "median(B04)")

d <- c(4,16,32,32)
x <- array(rnorm(prod(d)), d)
y <- reduce_time(x, function(v) {
  apply(v, 1, mean)
})
```

reduce_time.array *Apply a function over time and bands in a four-dimensional (band, time, y, x) array and reduce time dimension*

Description

Apply a function over time and bands in a four-dimensional (band, time, y, x) array and reduce time dimension

Usage

```
## S3 method for class 'array'
reduce_time(x, FUN, ...)
```

Arguments

x	four-dimensional input array with dimensions band, time, y, x (in this order)
FUN	function which receives one time series in a two-dimensional array with dimensions bands, time as input
...	further arguments passed to FUN

Details

FUN is expected to produce a numeric vector (or scalar) where elements are interpreted as new bands in the result.

Note

This is a helper function that uses the same dimension ordering as gdalcubes streaming. It can be used to simplify the application of R functions e.g. over time series in a data cube.

Examples

```
d <- c(4,16,32,32)
x <- array(rnorm(prod(d)), d)
# reduce individual bands over pixel time series
y <- reduce_time(x, function(v) {
  apply(v, 1, mean)
})
dim(y)
```

---

reduce_time.cube *Reduce a data cube over the time dimension*

Description

Create a proxy data cube, which applies one or more reducer functions to selected bands over pixel time series of a data cube

Usage

```
## S3 method for class 'cube'
reduce_time(x, expr, ..., FUN, names = NULL)
```

Arguments

x	source data cube
expr	either a single string, or a vector of strings defining which reducers will be applied over which bands of the input cube
...	optional additional expressions (if `expr` is not a vector)
FUN	a user-defined R function applied over pixel time series (see Details)
names	character vector; if FUN is provided, names can be used to define the number and name of output bands

Details

The function can either apply a built-in reducer if expr is given, or apply a custom R reducer function if FUN is provided.

In the former case, notice that expressions have a very simple format: the reducer is followed by the name of a band in parantheses. You cannot add more complex functions or arguments. Possible reducers currently are "min", "max", "sum", "prod", "count", "mean", "median", "var", "sd", "which_min", and "which_max".

User-defined R reducer functions receive a two-dimensional array as input where rows correspond to the band and columns represent the time dimension. For example, one row is the time series of a specific band. FUN should always return a numeric vector with the same number of elements, which will be interpreted as bands in the result cube. Notice that it is recommended to specify the names of the output bands as a character vector. If names are missing, the number and names of output bands is tried to be derived automatically, which may fail in some cases.

Value

proxy data cube object

Note

Implemented reducers will ignore any NAN values (as na.rm=TRUE does)

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-01", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.rgb = select_bands(L8.cube, c("B02", "B03", "B04"))
L8.rgb.median = reduce_time(L8.rgb, "median(B02)", "median(B03)", "median(B04)")
L8.rgb.median


plot(L8.rgb.median, rgb=3:1)


# user defined reducer calculating interquartile ranges
L8.rgb.iqr = reduce_time(L8.rgb, names=c("iqr_R", "iqr_G","iqr_B"), FUN = function(x) {
    c(diff(quantile(x["B04",],c(0.25,0.75), na.rm=TRUE)),
      diff(quantile(x["B03",],c(0.25,0.75), na.rm=TRUE)),
      diff(quantile(x["B02",],c(0.25,0.75), na.rm=TRUE)))
})
L8.rgb.iqr

plot(L8.rgb.iqr, key.pos=1)
```

select_bands *Select bands of a data cube*

Description

Create a proxy data cube, which selects specific bands of a data cube. The resulting cube will drop any other bands.

Usage

```
select_bands(cube, bands)
```

Arguments

cube	source data cube
bands	character vector with band names

Value

proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

For performance reasons, `select_bands` should always be called directly on a cube created with `raster_cube` and drop all unneded bands. This allows to reduce RasterIO and warp operations in GDAL.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}
```

```
L8.col = image_collection(file.path(tempdir(), ”L8.db”))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0=”2018-04”, t1=”2018-07”),
              srs=”EPSG:32618”, nx = 497, ny=526, dt=”P1M”)
L8.cube = raster_cube(L8.col, v)
L8.rgb = select_bands(L8.cube, c(”B02”, ”B03”, ”B04”))
L8.rgb

plot(L8.rgb, rgb=3:1)
```

select_time *Select time slices of a data cube*

Description

Create a proxy data cube, which selects specific time slices of a data cube. The time dimension of the resulting cube will be irregular / labeled.

Usage

```
select_time(cube, t)
```

Arguments

cube	source data cube
t	character vector with date/time

Value

proxy data cube object

Note

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), ”L8.db”))) {
  L8_files <- list.files(system.file(”L8NY18”, package = ”gdalcubes”),
                         ”.TIF”, recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, ”L8_L1TP”, file.path(tempdir(), ”L8.db”))
}

L8.col = image_collection(file.path(tempdir(), ”L8.db”))
```

```
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-07"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.rgb = select_bands(L8.cube, c("B02", "B03", "B04"))
L8.rgb = select_time(L8.rgb, c("2018-04", "2018-07"))
L8.rgb

plot(L8.rgb, rgb=3:1)
```

size *Query data cube properties*

Description

Query data cube properties

Usage

```
size(obj)
```

Arguments

obj	a data cube proxy object (class cube)

Value

size of a data cube (number of cells) as integer vector in the order t, y, x

See Also

dim.cube

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
size(raster_cube(L8.col, v))
```

srs *Query data cube properties*

Description

Query data cube properties

Usage

```
srs(obj)
```

Arguments

`obj`	a data cube proxy object (class cube)

Value

The spatial reference system expressed as a string readable by GDAL

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-06"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
srs(raster_cube(L8.col, v))
```

st_as_stars.cube *Coerce gdalcubes object into a stars object*

Description

The function materializes a data cube as a temporary netCDF file and loads the file with the stars package.

Usage

```
st_as_stars.cube(.x, ...)
```

Arguments

`.x`	data cube object to coerce
`...`	not used

Value

stars object

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-04"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
if(require("stars"))
  st_as_stars(select_bands(raster_cube(L8.col, v), c("B04", "B05")))
```

`translate_cog` *Convert complete image collections to cloud-optimized GeoTIFFs*

Description

This function translates all images of a gdalcubes image collection to cloud-optimized GeoTIFF files. The output contains converted imagery as an additional copy (original files are not deleted) and a new image collection file.

Usage

```
translate_cog(
  collection,
  target_dir = tempfile(pattern = "image_collection_"),
  overwrite = TRUE,
  creation_options = c("BLOCKSIZE=256", "COMPRESS=DEFLTE", "LEVEL=1",
    "RESAMPLING=CUBIC")
)
```

Arguments

`collection` path to an existing image collection file

`target_dir` directory where the output will be stored, will be created if necessary

`overwrite` logical; if TRUE existing files will be overwritten

`creation_options` further settings of the GDAL COG driver; see https://gdal.org/drivers/raster/cog.html

Value

path to the new image collection file for use as argument to `image_collection`

Note

This function requires the GDAL COG driver, which was added in GDAL version 3.1.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
L8.col

if ("COG" %in% gdalcubes_gdalformats()) {
  L8.cog.col = translate_cog(L8.col)
  L8.cog.col
}
```

`translate_gtiff` *Convert complete image collections to cloud-optimized GeoTIFFs*

Description

This function translates all images of a gdalcubes image collection to GeoTIFF files. The output contains converted imagery as an additional copy (original files are not deleted) and a new image collection file.

Usage

```
translate_gtiff(
  collection,
  target_dir = tempfile(pattern = "image_collection_"),
  overwrite = TRUE,
  creation_options = c("TILED=YES", "COMPRESS=DEFLATE", "ZLEVEL=1",
    "COPY_SRC_OVERVIEWS=TRUE")
)
```

Arguments

`collection`	path to an existing image collection file
`target_dir`	directory where the output will be stored, will be created if necessary
`overwrite`	logical; if TRUE existing files will be overwritten
`creation_options`	further settings of the GDAL GTiff driver; see https://gdal.org/drivers/raster/gtiff.html

Details

The functions `translate_gtiff` and `translate_cog` have the same purpose to convert imagery to optimized GeoTIFF files. The latter uses the recent COG GDAL driver, whereas the former uses the normal GTiff driver. Depending on additional creation options and the input images, files creted with `translate_gtiff` may or may not contain overview images.

Value

path to the new image collection file for use as argument to `image_collection`

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
L8.col


L8.tif.col = translate_gtiff(L8.col)
L8.tif.col
```

window_time *Apply a moving window operation over time*

Description

This generic function applies a reducer function over a moving window over the time dimension of a data cube, an R array, or other classes if implemented.

Usage

```
window_time(x, ...)
```

Arguments

x	object to be reduced
...	further arguments passed to specific implementations

Value

value and type depend on the class of x

See Also

window_time.cube

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931, t0="2018-01", t1="2018-07"),
                          srs="EPSG:32618", nx = 400, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.nir = select_bands(L8.cube, c("B05"))
window_time(L8.nir, window = c(2,2), "min(B05)")
window_time(L8.nir, kernel=c(-1,1), window=c(1,0))


plot(window_time(L8.nir, kernel=c(-1,1), window=c(1,0)), key.pos=1)
```

window_time.cube *Apply a moving window function over the time dimension of a data cube*

Description

Create a proxy data cube, which applies one ore more moving window functions to selected bands over pixel time series of a data cube. The fuction can either use a predefined agggregation function or apply a custom convolution kernel.

Usage

```
## S3 method for class 'cube'
window_time(x, expr, ..., kernel, window)
```

Arguments

`x`	source data cube
`expr`	either a single string, or a vector of strings defining which reducers wlil be applied over which bands of the input cube
`...`	optional additional expressions (if expr is not a vector)
`kernel`	numeric vector with elements of the kernel
`window`	integer vector with two elements defining the size of the window before and after a cell, the total size of the window is window[1] + 1 + window[2]

Details

The function either applies a kernel convolution (if the `kernel` argument is provided) or a general reducer function over moving temporal windows. In the former case, the kernel convolution will be applied over all bands of the input cube, i.e., the output cube will have the same number of bands as the input cubes. If a kernel is given and the `window` argument is missing, the window will be symmetric to the center pixel with the size of the provided kernel. For general reducer functions, the window argument must be provided and several expressions can be used to create multiple bands in the output cube.

Notice that expressions have a very simple format: the reducer is followed by the name of a band in parantheses. You cannot add more complex functions or arguments.

Possible reducers currently are "min", "max", "sum", "prod", "count", "mean", "median".

Value

proxy data cube object

Note

Implemented reducers will ignore any NAN values (as na.rm=TRUE does).

This function returns a proxy object, i.e., it will not start any computations besides deriving the shape of the result.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931, t0="2018-01", t1="2018-07"),
                          srs="EPSG:32618", nx = 400, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.nir = select_bands(L8.cube, c("B05"))
L8.nir.min = window_time(L8.nir, window = c(2,2), "min(B05)")
L8.nir.min

L8.nir.kernel = window_time(L8.nir, kernel=c(-1,1), window=c(1,0))
L8.nir.kernel
```

write_chunk_from_array

Write chunk data of a cube to stdout or a file

Description

This function can be used within function passed to chunk_apply in order to pass four-dimensional R arrays as a data cube chunk to the gdalcubes C++ library. It works only for R processes, which have been started from the gdalcubes C++ library. The input array must have dimensions band, time, y, x (in this order).

Usage

```
write_chunk_from_array(v)
```

Arguments

v	four-dimensional array with dimensions band, time, y, and x

Note

Call this function ONLY from a function passed to chunk_apply.

This function only works in R sessions started from gdalcubes streaming.

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931, t0="2018-01", t1="2018-12"),
                          srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
f <- function() {
  x <- read_chunk_as_array()
  out <- reduce_time(x, function(x) {
    cor(x[1,], x[2,], use="na.or.complete", method = "kendall")
  })
  write_chunk_from_array(out)
}
L8.cor = chunk_apply(L8.cube, f)
plot(L8.cor, zlim=c(0,1), key.pos=1)
```

---

write_ncdf *Export a data cube as netCDF file(s)*

Description

This function will read chunks of a data cube and write them to a single (the default) or multitple (if chunked = TRUE) netCDF file(s). The resulting file(s) uses the enhanced netCDF-4 format, supporting chunking and compression.

Usage

```
write_ncdf(
  x,
  fname = tempfile(pattern = "gdalcubes", fileext = ".nc"),
  overwrite = FALSE,
  write_json_descr = FALSE,
  with_VRT = FALSE,
  pack = NULL,
  chunked = FALSE
)
```

Arguments

`x` a data cube proxy object (class cube)

`fname` output file name

`overwrite` logical; overwrite output file if it already exists

`write_json_descr` logical; write a JSON description of x as additional file

`with_VRT` logical; write additional VRT datasets (one per time slice)

`pack` reduce output file size by packing values (see Details), defaults to no packing

`chunked` logical; if TRUE, write one netCDF file per chunk; defaults to FALSE

Details

The resulting netCDF file(s) contain three dimensions (t, y, x) and bands as variables.

If `write_json_descr` is TRUE, the function will write an addition file with the same name as the NetCDF file but ".json" suffix. This file includes a serialized description of the input data cube, including all chained data cube operations.

To reduce the size of created files, values can be packed by applying a scale factor and an offset value and using a smaller integer data type for storage (only supported if `chunked = TRUE`). The `pack` argument can be either NULL (the default), or a list with elements `type`, `scale`, `offset`, and `nodata`. `type` can be any of "uint8", "uint16" , "uint32", "int16", or "int32". `scale`, `offset`, and `nodata` must be numeric vectors with length one or length equal to the number of data cube bands (to use different values for different bands). The helper function `pack_minmax` can be used to derive offset and scale values with maximum precision from minimum and maximum data values on original scale.

If `chunked = TRUE`, names of the produced files will start with `name` (with removed extension), followed by an underscore and the internal integer chunk number.

Value

returns (invisibly) the path of the created netCDF file(s)

Note

Packing is currently ignored if `chunked = TRUE`

See Also

`https://www.unidata.ucar.edu/software/netcdf/docs/`

`gdalcubes_set_ncdf_compression`

`pack_minmax`

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-04"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
write_ncdf(select_bands(raster_cube(L8.col, v), c("B04", "B05")), fname=tempfile(fileext = ".nc"))
```

write_tif *Export a data cube as a collection of GeoTIFF files*

Description

This function will time slices of a data cube as GeoTIFF files in a given directory.

Usage

```
write_tif(
  x,
  dir = tempfile(pattern = ""),
  prefix = basename(tempfile(pattern = "cube_")),
  overviews = FALSE,
  COG = FALSE,
  rsmpl_overview = "nearest",
  creation_options = NULL,
  write_json_descr = FALSE,
  pack = NULL
)
```

Arguments

`x` a data cube proxy object (class cube)

`dir` destination directory

`prefix` output file name

`overviews` logical; generate overview images

`COG` logical; create cloud-optimized GeoTIFF files (forces overviews=TRUE)

`rsmpl_overview` resampling method for overviews (image pyramid) generation (see https://gdal.org/programs/gdaladdo.html for available methods)

creation_options
: additional creation options for resulting GeoTIFF files, e.g. to define compression (see https://gdal.org/drivers/raster/gtiff.html#creation-options)

write_json_descr
: logical; write a JSON description of x as additional file

pack
: reduce output file size by packing values (see Details), defaults to no packing

Details

If write_json_descr is TRUE, the function will write an additional file with name according to prefix (if not missing) or simply cube.json This file includes a serialized description of the input data cube, including all chained data cube operations.

Additional GDAL creation options for resulting GeoTIFF files must be passed as a named list of simple strings, where element names refer to the key. For example, creation_options = list("COMPRESS" = "DEFLATE","ZLEVEL" = "5") would enable deflate compression at level 5.

To reduce the size of created files, values can be packed by applying a scale factor and an offset value and using a smaller integer data type for storage. The pack argument can be either NULL (the default), or a list with elements type, scale, offset, and nodata. type can be any of "uint8", "uint16" , "uint32", "int16", or "int32". scale, offset, and nodata must be numeric vectors with length one or length equal to the number of data cube bands (to use different values for different bands). The helper function pack_minmax can be used to derive offset and scale values with maximum precision from minimum and maximum data values on original scale.

If overviews=TRUE, the numbers of pixels are halved until the longer spatial dimensions counts less than 256 pixels. Setting COG=TRUE automatically sets overviews=TRUE.

Value

returns (invisibly) a vector of paths pointing to the created GeoTIFF files

See Also

pack_minmax

Examples

```
# create image collection from example Landsat data only
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}

L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(extent=list(left=388941.2, right=766552.4,
              bottom=4345299, top=4744931, t0="2018-04", t1="2018-04"),
              srs="EPSG:32618", nx = 497, ny=526, dt="P1M")
write_tif(select_bands(raster_cube(L8.col, v), c("B04", "B05")), dir=)
```

zonal_statistics *Query summary statistics of data cube values over polygons*

Description

This function will overlay spatial polygons with a data cube and compute summary statistics of pixel values within the polygons over all time slices.

Usage

```
zonal_statistics(
  x,
  geom,
  expr,
  out_path = tempfile(fileext = ".gpkg"),
  overwrite = FALSE,
  ogr_layer = NULL,
  as_stars = FALSE
)
```

Arguments

`x`	source data cube
`geom`	Either an sf object, or a path to an OGR dataset (Shapefile, GeoPackage, or similar) with input (multi)polygon geometries
`expr`	character vector of summary statistics expressions, describing pairs of aggregation functions and data cube bands (e.g. "mean(band1)")
`out_path`	path to where resulting GeoPackage will be written to
`overwrite`	logical; overwrite `out_path` if file already exists, defaults to FALSE
`ogr_layer`	If the input OGR dataset has multiple layers, a layer can be chosen by name
`as_stars`	logical; if TRUE, the created gpkg file will be loaded as a stars vector data cube

Details

The function creates a single GeoPackage output file containing:

- A single layer "geom" containing the geometries (and feature identifiers) only.
- Attribute tables (layers without geometry) for each time slice of the data cube containing summary statistics as columns. Corresponding layer names start with "attr_", followed by date and time.
- Virtual spatial views for each time slice, joining the geometries and attribute tables. Corresponding layer names start with "map_", followed by date and time.

You will most-likely want to use the spatial view layers directly e.g. with the sf package.

Available summary statistics currently include "min", "max", "mean", "median", "count", "sum", "prod", "var", and "sd".

Value

character length-one vector containing the path to the resulting GeoPackage file (see Details) or a stars object (if as_stars is TRUE)

Note

Currently, the spatial reference systems of the data cube and the features must be identical.

This function requires GDAL with built-in GEOS support, which can checked with gdalcubes_gdal_has_geos)

Examples

```
# if not already done in other examples
if (!file.exists(file.path(tempdir(), "L8.db"))) {
  L8_files <- list.files(system.file("L8NY18", package = "gdalcubes"),
                         ".TIF", recursive = TRUE, full.names = TRUE)
  create_image_collection(L8_files, "L8_L1TP", file.path(tempdir(), "L8.db"))
}
L8.col = image_collection(file.path(tempdir(), "L8.db"))
v = cube_view(srs="EPSG:32618", dy=300, dx=300, dt="P1M",
              aggregation = "median", resampling = "bilinear",
              extent=list(left=388941.2, right=766552.4,
                          bottom=4345299, top=4744931,
                          t0="2018-01-01", t1="2018-04-30"))
L8.cube = raster_cube(L8.col, v)
L8.cube = select_bands(L8.cube, c("B04", "B05"))
L8.ndvi = apply_pixel(L8.cube, "(B05-B04)/(B05+B04)", "NDVI")
L8.ndvi

# toy example: overlay NDVI data with NYC districts
if (gdalcubes_gdal_has_geos()) {
  x = zonal_statistics(L8.ndvi, system.file("nycd.gpkg", package = "gdalcubes"),
                       expr = "median(NDVI)")
  x
}
```

Index