Package 'vaster'

March 10, 2026

Title Tools for Raster Grid Logic

Version 0.6.0

Description Provides raster grid logic, operations that describe a discretized rectangular domain and do not require access to materialized data. Grids are arrays with dimension and extent, and many operations are functions of dimension only: number of columns, number of rows, or they are a combination of the dimension and the extent the range in x and the range in y in that order. Here we provide direct access to this logic without need for connection to any materialized data or formats. Grid logic includes functions that relate the cell index to row and column, or row and column to cell index, row, column or cell index to position. These methods are described in Loudon, TV, Wheeler, JF, Andrew, KP (1980) <doi:10.1016/0098-3004(80)90015-1>, and implementations were in part derived from Hijmans R (2024) <doi:10.32614/CRAN.package.terra>.

NeedsCompilation yes

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/hypertidy/vaster, https://hypertidy.github.io/vaster/

BugReports https://github.com/hypertidy/vaster/issues

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Language en-US

License MIT + file LICENSE

Author Michael Sumner [aut, cre] (ORCID: <https://orcid.org/0000-0002-2471-7511>), Robert Hijmans [ctb] (Wrote original versions of abstract cell operations in raster package)

Maintainer Michael Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2026-03-10 11:50:02 UTC

Contents

`vaster-package` *vaster: Tools for Raster Grid Logic*

Description

Provides raster grid logic without requiring materialized data. Most raster operations are purely functions of dimension (ncol, nrow) and extent (xmin, xmax, ymin, ymax) - vaster provides these operations as simple functions on numbers, independent of any file format or geospatial library.

Core concepts

A raster grid is defined by two properties:

- **dimension**: `c(ncol, nrow)` - the number of columns and rows
- **extent**: `c(xmin, xmax, ymin, ymax)` - the bounding box

From these six numbers, all grid geometry can be computed: cell indices, coordinates, resolutions, and spatial queries.

Cell indexing

Cells are numbered from 1, starting at the top-left corner, proceeding right then down (row-major order). This matches the convention used by the raster and terra packages.

Main function families

Cell operations (cells):

- `cell_from_xy()`, `cell_from_row_col()`, `cell_from_rowcol_combine()` - cell index from position
- `cell_from_row()`, `cell_from_col()` - all cells in a row or column
- `cell_from_extent()`, `extent_from_cell()` - cell/extent conversion
- `xy_from_cell()`, `x_from_cell()`, `y_from_cell()` - coordinates from cell
- `rowcol_from_cell()`, `row_from_cell()`, `col_from_cell()` - row/column from cell

Coordinate operations (coordinates):

- `x_from_col()`, `y_from_row()` - coordinate from index
- `col_from_x()`, `row_from_y()` - index from coordinate
- `x_corner()`, `y_corner()` - corner coordinates of all cells
- `x_centre()`, `y_centre()` - centre coordinates of all cells
- `xy()` - centre coordinates of all cells as a matrix

Grid properties (grid):

- `x_res()`, `y_res()` - cell resolution
- `n_cell()`, `n_row()`, `n_col()` - counts
- `xlim()`, `ylim()` - extent as 2-element ranges
- `x_min()`, `x_max()`, `y_min()`, `y_max()` - individual extent edges
- `origin()` - grid origin (alignment anchor)

Grid modification:

- `vcrop()` - crop or extend a grid, snapped to alignment
- `align_extent()` - snap an extent to grid origin
- `extent_dimension()` - dimension for an aligned extent
- `intersect_extent()` - overlapping extent of two grids

- `snap_extent()` / `buffer_extent()` - snap extent to resolution

Cell adjacency (adjacency):

- `adjacency()` - neighbour cell indices (queen, rook, or bishop)

GDAL interoperability:

- `geo_transform0()`, `geo_world0()` - create geotransform / world file vectors
- `world_to_geotransform()`, `geotransform_to_world()` - convert between formats
- `extent_dim_to_gt()`, `gt_dim_to_extent()` - convert extent/dimension to/from geotransform
- `rasterio0()`, `rasterio_idx()`, `raster_sfio()` - GDAL RasterIO parameters
- `rasterio_to_sfio()`, `sfio_to_rasterio()` - convert between RasterIO formats
- `ts_te()`, `gdal_ts()`, `gdal_te()` - format dimension/extent for GDAL command line

Output and display:

- `vaster_long()` - cell coordinates as long-form matrix
- `vaster_listxyz()` - grid as x, y, z list for `graphics::image()`
- `vaster_boundary()`, `vaster_boundary_cell()` - boundary coordinates
- `plot_extent()`, `draw_extent()` - plot and interactively draw extents
- `extent_vrt()` - tile extents from VRT files

Utilities:

- `from_xyz()` - derive grid dimension and extent from XYZ points
- `fit_dims()` - compute dimension from aspect ratio and target size

Extent convention

vaster uses extent as `c(xmin, xmax, ymin, ymax)`, which differs from the bbox convention `c(xmin, ymin, xmax, ymax)`. Both represent the same information in different order.

If extent is not provided, the default is `c(0, ncol, 0, nrow)` - matching the convention adopted by stars, terra, and an improvement on base R's `image()` which scales to 0-1.

Author(s)

Maintainer: Michael Sumner <mdsumner@gmail.com> (ORCID)

Other contributors:

- Robert Hijmans (Wrote original versions of abstract cell operations in raster package) [contributor]

See Also

Useful links:

- https://github.com/hypertidy/vaster
- https://hypertidy.github.io/vaster/
- Report bugs at https://github.com/hypertidy/vaster/issues

`adjacency` *Cell adjacency*

Description

Find neighbouring cells by index, using only the grid dimension.

Usage

```
adjacency(dimension, cell, directions = "queen")
```

Arguments

`dimension`	integer vector of ncol, nrow
`cell`	integer vector of cell indices (1-based)
`directions`	character, one of `"queen"`, `"rook"`, or `"bishop"`

Details

Given cell indices into a grid of known dimension, return the indices of neighbouring cells. Out-of-bounds neighbours (at grid edges) are `NA`.

The `directions` argument controls which neighbours are returned:

- `"queen"` (default): all 8 neighbours (rook + bishop)
- `"rook"`: 4 cardinal neighbours (up, down, left, right)
- `"bishop"`: 4 diagonal neighbours

Column order for `"queen"` is: up, down, left, right, upleft, upright, downleft, downright. For `"rook"`: up, down, left, right. For `"bishop"`: upleft, upright, downleft, downright. Direction names refer to the raster convention where cell 1 is at the top-left.

Value

integer matrix with one row per cell and one column per neighbour direction. Column names indicate the direction. Out-of-bounds neighbours are `NA`.

Corner vertex values from area-based rasters

A key use case is converting area-based cell values to corner vertex values for mesh generation. Each corner vertex is shared by up to four cells. For example, the top-left corner of a cell is shared with the cell above, the cell to the left, and the cell diagonally above-left:

```
upleft  |  up
--------x------     vertex 'x' = mean of upleft, up, left, self
left    | self
```

The queen-connected neighbours give all the cells needed to compute every corner vertex. Averaging the appropriate neighbours provides weighted corner values from flat pixel areas, which is the basis for constructing continuous meshes from raster data (as used by quadmesh and anglr).

Examples

```
## 4x3 grid (4 columns, 3 rows), 12 cells
adjacency(c(4, 3), cell = 6)
adjacency(c(4, 3), cell = 6, directions = "rook")
adjacency(c(4, 3), cell = 6, directions = "bishop")

## corner cell has fewer valid neighbours
adjacency(c(4, 3), cell = 1)

## multiple cells at once
adjacency(c(4, 3), cell = 1:12)

## ------------------------------------------------
## Corner vertex interpolation from area values
## ------------------------------------------------
dm <- c(4, 3)
elev <- c(10, 12, 14, 16, 11, 13, 15, 17, 10, 11, 13, 14)

## The top-left corner of each cell is shared by self, the
## cell above, the cell to the left, and the cell diagonally
## above-left. Average these for the vertex value:
nb <- adjacency(dm, seq_along(elev))
vals <- cbind(elev, elev[nb[, "up"]],
              elev[nb[, "left"]], elev[nb[, "upleft"]])
corner <- rowMeans(vals, na.rm = TRUE)
matrix(corner, nrow = 3, byrow = TRUE)
```

`align_extent` *Crop an extent, snapped to the grain*

Description

A crop (or extend), it snaps the input extent to the origin of the input extent (based on the dimension) #' Note that snap is modelled on the behaviour of the raster package, and is different from projwin in GDAL (WIP to illustrate).

Usage

```
align_extent(x, dimension, extent = NULL, snap = c("out", "near", "in"))
```

Arguments

`x`	extent
`dimension`	integer ncol, nrow
`extent`	numeric extent xmin,xmax,ymin,ymax
`snap`	out by default, may be near or in

Value

aligned extent

Examples

```
align_extent(c(4.5, 5.6, 2, 4), c(10, 5), c(0, 10, 0, 5))
```

`cells` *Cells*

Description

Functions that work with cells.

Usage

```
cell_from_xy(dimension, extent = NULL, xy)

cell_from_extent(dimension, extent = NULL, x_extent)

extent_from_cell(dimension, extent = NULL, cell)

rowcol_from_cell(dimension, extent = NULL, cell)

xy_from_cell(dimension, extent = NULL, cell)

x_from_cell(dimension, extent = NULL, cell)

y_from_cell(dimension, extent = NULL, cell)

col_from_cell(dimension, cell)

row_from_cell(dimension, cell)

cell_from_row(dimension, row)

cell_from_col(dimension, col)

cell_from_row_col(dimension, row, col)

cell_from_rowcol_combine(dimension, row, col)
```

Arguments

`dimension`	integer ncol, nrow
`extent`	numeric extent xmin,xmax,ymin,ymax
`xy`	matrix of coordinates

x_extent	extent to find cells of
cell	cells to find extent, or row,col, or xy of
row	row to find cell of
col	column to find cell of

Details

The cell is indexed from the top left corner and proceeds to the right, and then down scanning by rows. The n cell is a the bottom right corner. Orientation is different to R's native matrix order, but see (WiP doc and helpers for conversion).

Value

cell index

cells of extent

extent of cells

row,col of cells

xy from cells

x of cells

y of cells

col of cells

row of cells

cell of rows

cell of cols

cell of row,col

cell of row,col combined

Examples

```
cell_from_xy(c(10, 5), extent = c(0, 10, 0, 5), cbind(5, 4))
cell_from_extent(c(10, 5), c(0, 10, 0, 5), c(6, 7, 2, 3))
extent_from_cell(c(10, 5), c(0, 10, 0, 5), c(4, 5))
rowcol_from_cell(c(10, 5), c(0, 10, 0, 5), 3:5)
xy_from_cell(c(10, 5), c(0, 10, 0, 5), 4:6)
x_from_cell(c(10, 5), c(0, 10, 0, 5), 4:7)
y_from_cell(c(10, 5), c(0, 10, 0, 5), 4:7)
col_from_cell(c(10, 5),  4:7)
row_from_cell(c(10, 5),  4:7)
cell_from_row(c(10, 5), 4:7)
cell_from_col(c(10, 5), 4:7)
cell_from_row_col(c(10, 5), 1:4, 4:7)
cell_from_rowcol_combine(c(10, 5), 1:4, 4:7)
```

coordinates *Coordinates*

Description

Functions that work with coordinates.

Usage

```
x_corner(dimension, extent = NULL)

y_corner(dimension, extent = NULL)

x_centre(dimension, extent = NULL)

y_centre(dimension, extent = NULL)

x_from_col(dimension, extent = NULL, col)

y_from_row(dimension, extent = NULL, row)

col_from_x(dimension, extent = NULL, x)

row_from_y(dimension, extent = NULL, y)

xy(dimension, extent = NULL)
```

Arguments

dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax
col	column index
row	row index
x	x coordinate
y	y coordinate

Value

x coordinate of corners

y coordinate of corners

x coordinate of centres

y coordinate of centres

x coordinate of col (centre)

y coordinate of row (centre)

col of x coordinate

y coordinate (centre) of row

xy coordinate (centre) of grid

Examples

```
x_corner(c(10, 5), c(0, 10, 0, 5))
y_corner(c(10, 5), c(0, 10, 0, 5))
x_centre(c(10, 5), c(0, 10, 0, 5))
y_centre(c(10, 5), c(0, 10, 0, 5))
x_from_col(c(10, 5), c(0, 10, 0, 5), 2:3)
y_from_row(c(10, 5), c(0, 10, 0, 5), 2:3)
col_from_x(c(10, 5), c(0, 10, 0, 5), 3.5 + 1:2)
row_from_y(c(10, 5), c(0, 10, 0, 5), 2:3)
xy(c(10, 5), c(0, 10, 0, 5))
```

`draw_extent` *Draw extent*

Description

Draw an extent with two clicks

Usage

```
draw_extent(show = TRUE, ...)
```

Arguments

show	the drawn extent
...	arguments pass to graphics::rect()

Value

an extent, numeric vector of xmin,xmax,ymin,ymax

Examples

```
if (interactive()) {
plot(1)
draw_extent(show = TRUE)  ## click twice on the plot
}
```

extent_dimension *Dimension for an aligned extent*

Description

input is the output of align_extent

Usage

```
extent_dimension(x, dimension, extent = NULL, snap = "out")
```

Arguments

x	and aligned extent
dimension	dimension of parent
extent	of parent
snap	out by default, may be near or in

Value

dimension

Examples

```
extent_dimension(c(.2, .8, 1.8, 3.2), c(10, 5), c(0, 10, 0, 5))
```

extent_dim_to_gt *Create geotransform from extent and dimension*

Description

Create the geotransform (see geo_transform0()) from extent and dimension.

Usage

```
extent_dim_to_gt(x, dimension)
```

Arguments

x	extent parameters, c(xmin,xmax,ymin,ymax)
dimension	dimensions x,y of grid (ncol,nrow)

Details

The dimension is always ncol, nrow.

Value

6-element geo_transform0()

Examples

```
extent_dim_to_gt(c(0, 5, 0, 10), c(5, 10))
```

---

extent_vrt *Extents from VRT (virtual raster data set of GDAL)*

Description

Get extent from index values in VRT text, these are the individual footprints of raster windows in a VRT. These can be arbitrary to a specific grid, but generally are used for tiled mosaics.

Usage

```
extent_vrt(x)
```

Arguments

x	url or file path to VRT file

Details

Each VRT raster element records it's relative position within the grid, so we grid logic to expand the actual extent of each element, and return those as a matrix of xmin,xmax,ymin,ymax.

Value

a matrix of 4 columns, xmin,xmax,ymin,ymax

Examples

```
#src <- "https://opentopography.s3.sdsc.edu/raster/NASADEM/NASADEM_be.vrt"
src <- gzfile(system.file("extdata/NASADEM_be.vrt.gz", package = "vaster"), "rt")
## read VRT from a URL or file (we use a connection here to keep package example small)
ex <- extent_vrt(src)
close(src)
plot_extent(ex)
```

fit_dims *Aspect ratio of dimension conflated with bbox*

Description

Generate an appropriate dimension (shape, ncol,nrow) from an input width(height). If height not specified we have a square.

Usage

```
fit_dims(size = 1024L, wh = c(size, size))
```

Arguments

size	seed dimension size
wh	distance across dimension span/s

Value

dimension `c(ncol, nrow)`

Examples

```
fit_dims(256, c(10, 20))
fit_dims(1024, c(102723, 1e5))
```

from_xyz *Derive a grid from XYZ points*

Description

This function is very liberal, it simply finds unique x values and unique y values, sorts them and finds the minimum difference between the points, then checks that rounded ratio of differences to this minimum is 1.

Usage

```
from_xyz(xyz, digits = 5)
```

Arguments

xyz	set of points xy or xyz (matrix or data frame)
digits	argument passed to `round()`

Details

The points can be the full grid set, a partial set, or a superset of the grid. The resolution will simply be the smallest actual difference found. (Zero is not possible because we `sort(unique(...))`).

The z-column if present is ignored.

Value

list with elements 'dimension', 'extent'

Examples

```
from_xyz(vaster_long(c(10, 5), c(0, 10, 0, 5)))
```

`geo_transform0` *Geo transform parameter creator*

Description

Basic function to create a geotransform as used by GDAL.

Usage

```
geo_transform0(px, ul, sh = c(0, 0))
```

Arguments

`px`	pixel resolution (XY, Y-negative)
`ul`	grid offset, top-left corner
`sh`	affine shear (XY)

Value

vector of parameters xmin, xres, yskew, ymax, xskew, yres

See Also

`geo_world0()` which uses the same parameters in a different order

Examples

```
geo_transform0(px = c(1, -1), ul = c(0, 0))
```

`geo_world0` *World file parameter creator*

Description

Basic function to create a 'world file' as used by various non-geo image formats

Reformat to world vector.

Usage

```
geo_world0(px, ul, sh = c(0, 0))

geotransform_to_world(x)
```

Arguments

`px`	pixel resolution (XY, Y-negative)
`ul`	grid offset, top-left corner
`sh`	affine shear (XY)
`x`	geotransform parameters, as per `geo_transform0()`

Details

Note that xmin/xmax are *centre_of_cell* (of top-left cell) unlike the geotransform which is top-left *corner_of_cell*. The parameters are otherwise the same, but in a different order.

Value

vector of parameters xres, yskew, xskew, yres, xmin, ymax

world vector, as per `geo_world0()`

See Also

geo_transform0

Examples

```
geo_world0(px = c(1, -1), ul = c(0, 0))
(gt <- geo_transform0(px = c(1, -1), ul = c(0, 0)))
wf <- geotransform_to_world(gt)
world_to_geotransform(wf)
```

grid *Grid*

Description

Basic grid tools, cell, resolution, dimension, extent.

Usage

```
n_cell(dimension)

x_res(dimension, extent = NULL)

y_res(dimension, extent = NULL)

n_row(dimension)

n_col(dimension)

xlim(dimension, extent = NULL)

ylim(dimension, extent = NULL)

x_min(dimension, extent = NULL)

x_max(dimension, extent = NULL)

y_min(dimension, extent = NULL)

y_max(dimension, extent = NULL)
```

Arguments

dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax

Value

number of cells

x resolution (width of cell)

y resolution (height of cell)

number of rows

number of cols

x extent (corner to corner)

y extent (corner to corner)

x minimum (left edge)

x maximum (right edge)

y minimum (bottom edge)

y maximum (top edge)

Examples

```
n_cell(c(10, 5))
x_res(c(10, 5), c(0, 10, 0, 5))
y_res(c(10, 5), c(0, 10, 0, 5))
n_row(c(10, 5))
n_col(c(10, 5))
xlim(c(10, 5), c(0, 10, 0, 5))
ylim(c(10, 5), c(0, 10, 0, 5))
x_min(c(10, 5), c(0, 10, 0, 5))
x_max(c(10, 5), c(0, 10, 0, 5))
y_min(c(10, 5), c(0, 10, 0, 5))
y_max(c(10, 5), c(0, 10, 0, 5))
```

---

gt_dim_to_extent *Determine extent from geotransform vector and dimension*

Description

Create the extent (xlim, ylim) from the geotransform and dimensions of the grid.

Usage

```
gt_dim_to_extent(x, dim)
```

Arguments

x	geotransform parameters, as per geo_transform0()
dim	dimensions x,y of grid (ncol,nrow)

Details

The extent is `c(xmin, xmax, ymin, ymax)`.

Value

4-element extent c(xmin,xmax,ymin,ymax)

Examples

```
gt_dim_to_extent(geo_transform0(c(1, -1), c(0, 10)), c(5, 10))
```

intersect_extent *Intersect extent*

Description

Return the overlapping extent.

Usage

```
intersect_extent(x, dimension, extent = NULL)
```

Arguments

x	extent to intersect
dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax

Value

extent, a numeric vector of xmin,xmax,ymin,ymax

Examples

```
intersect_extent(c(0.5, 2.3, 1.2, 5), c(10, 5), c(0, 10, 0, 5))
```

origin *Origin of grid alignment*

Description

Origin of grid alignment

Usage

```
origin(dimension, extent = NULL)
```

Arguments

dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax

Value

coordinate of grid origin

Examples

```
origin(c(10, 5), c(0, 10, 0, 5))
```

plot_extent *Plot an extent*

Description

Plot an extent

Usage

```
plot_extent(x, ..., asp = 1, add = FALSE, border = "black")
```

Arguments

x	extent xmin,xmax,ymin,ymax
...	arguments passed to graphics::rect()
asp	aspect ratio (1 by default)
add	add to plot or initiated one, FALSE by default
border	colour of lines of extent

Value

nothing, used for plot side effect

Examples

```
plot_extent(c(-180, 180, -90, 90))
plot_extent(c(100, 150, -60, -30), add = TRUE, border = "firebrick")
```

rasterio0 *GDAL RasterIO parameter creator*

Description

Basic function to create the window parameters as used by GDAL RasterIO.

Usage

```
rasterio0(
  src_offset,
  src_dim,
  out_dim = src_dim,
  resample = "NearestNeighbour"
)
```

Arguments

src_offset	index offset (0-based, top left)
src_dim	source dimension (XY)
out_dim	output dimension (XY, optional src_dim will be used if not set)
resample	resampling algorithm for GDAL see details

Details

Resampling algorithm is one of 'NearestNeighbour' (default), 'Average', 'Bilinear', 'Cubic', 'CubicSpline', 'Gauss', 'Lanczos', 'Mode', but more may be available given the version of GDAL in use.

Value

numeric vector of values specifying offset, source dimension, output dimension

Examples

```
rasterio0(c(0L, 0L), src_dim = c(24L, 10L))
```

---

rasterio_idx *RasterIO index window*

Description

The sf RasterIO is the RasterIO window in a list format used by the sf package, it contains the same information, and is created by raster_sfio().

Usage

```
rasterio_idx(dimension, extent)

raster_sfio(dimension, fact = 1, resample = "Nearest")
```

Arguments

dimension	ncols, nrows
extent	this is ignored
fact	a resizing factor
resample	resample algorithm for GDAL RasterIO

Value

RasterIO window vector 'c(x0, y0, nx0, ny0, nx, y)' see Details

Examples

```
rasterio_idx(dim(volcano))
```

rasterio_to_sfio *The sf/stars RasterIO list*

Description

We create the list as used by the stars/sf GDAL IO function 'gdal_read(, RasterIO_parameters)'.

Usage

```
rasterio_to_sfio(x)
```

Arguments

x	rasterio params as from `rasterio0()`

Details

Note that the input is a 4 or 6 element vector, with offset 0-based and output dimensions optional (will use the source window). The resample argument uses the syntax identical to that used in GDAL itself.

Value

list in sf RasterIO format

Examples

```
rio <- rasterio0(c(0L, 0L), src_dim = c(24L, 10L))
rasterio_to_sfio(rio)
```

sfio_to_rasterio *sf package RasterIO from RasterIO window vector*

Description

Basic function to create the window parameters as used by GDAL RasterIO, in format used by sf, in 'gdal_read(,RasterIO_parameters)'.

Usage

```
sfio_to_rasterio(x)
```

Arguments

x	a RasterIO parameter list

Value

a sf-RasterIO parameter list

Examples

```
sfio_to_rasterio(rasterio_to_sfio(rasterio0(c(0L, 0L), src_dim = c(24L, 10L))))
```

`snap_extent` *Snap extent to resolution (buffer extent)*

Description

Whole grain buffers.

Usage

```
snap_extent(x, res)

buffer_extent(x, res)
```

Arguments

`x`	extent (xmin, xmax, ymin, ymax)
`res`	resolution (a grain to align to)

Value

extent, snapped to the resolution

Examples

```
snap_extent(sort(rnorm(4)), 0.01)
```

`ts_te` *Target size and extent for GDAL command line*

Description

Format grid properties for GDAL command line options (`-ts` for target size, `-te` for target extent).

Usage

```
ts_te(dimension, extent)

gdal_te(extent)

gdal_ts(dimension)
```

Arguments

`dimension`	integer vector of ncol, nrow (target size)
`extent`	numeric vector of xmin, xmax, ymin, ymax (target extent)

Details

These functions generate the string arguments used by GDAL utilities like `gdalwarp` and `gdal_translate`. The `gdal_ts()` function is named after the GDAL `-ts` flag and `gdal_te()` after the GDAL `-te` flag.

Value

A character string formatted for GDAL command line:

- `ts_te()`: combined `-ts` and `-te` arguments
- `gdal_ts()`: `-ts ncol nrow` string
- `gdal_te()`: `-te xmin ymin xmax ymax` string (note: reordered for GDAL)

See Also

`vcrop()` for computing aligned extents

Examples

```
ts_te(c(10, 100), 1:4)

gdal_ts(c(10, 100))

gdal_te(1:4)

## use in a GDAL command (not run)
## Not run:
cmd <- sprintf("gdalwarp %s %s input.tif output.tif",
               gdal_ts(c(1000, 500)), gdal_te(c(-180, 180, -90, 90)))

## End(Not run)
```

`vaster_boundary` *Grid boundary in native resolution*

Description

Generates the coordinates around a grid boundary, with a coordinate for each grid cell corner.

Usage

```
vaster_boundary(dimension, extent = NULL)
```

Arguments

dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax

Details

The orientation is starts along the bottom and goes counter-clockwise.

Value

matrix of xy coordinates

Examples

```
vaster_boundary(c(3, 4))

plot(vaster_boundary(c(36, 18), c(-180, 180, -90, 90)))
```

---

vaster_boundary_cell *Grid boundary cell index*

Description

This is for indexing coordinate arrays to get their values (the cell index of the outer row and columns).

Usage

```
vaster_boundary_cell(dimension)
```

Arguments

dimension	integer ncol, nrow

Details

The orientation is the same as for vaster_boundary().

Value

a matrix of xy coordinates

Examples

```
vaster_boundary_cell(c(3, 4))
cell <- vaster_boundary_cell(c(3, 4))
plot(vaster_boundary(c(3, 4)))
text(xy <- xy_from_cell(c(3, 4), cell = vaster_boundary_cell(c(3, 4))), lab = cell)
lines(xy)
```

vaster_listxyz *Image xyz list*

Description

Generate list of x and y rectilinear coordinates with z matrix.

Usage

```
vaster_listxyz(dimension, extent = NULL, data = NULL)
```

Arguments

dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax
data	data values (length of the product of 'dimension')

Details

The rectilinear coordinates are degenerate (just a product of extent/dimension).

Value

list with elements x,y,z as per graphics::image

Examples

```
vaster_listxyz(c(10, 5), c(0, 10, 0, 5))
## see https://gist.github.com/mdsumner/b844766f28910a3f87dc2c8a398a3a13
```

vaster_long *Convert to long form coordinates*

Description

Matrix of xyz values in raster order.

Usage

```
vaster_long(dimension, extent = NULL, data = NULL, raster_order = TRUE)
```

Arguments

dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax
data	data values
raster_order	use raster order or native R matrix order

Details

Use 'raster_order = FALSE' for traditional R matrix x,y order

Value

matrix of coordinates x,y

Examples

```
vaster_long(c(10, 5), c(0, 10, 0, 5))
# see https://gist.github.com/mdsumner/b844766f28910a3f87dc2c8a398a3a13
```

---

vcrop *Virtual grid modification*

Description

To modify a grid is to align an extent to the grid origin. Modification includes reducing or extending the area covered in either dimension. This implies a new extent, snapped to the grain of the origin grid, and a new dimension (ncol, nrow).

Usage

```
vcrop(x, dimension, extent = NULL, ..., snap = "out")
```

Arguments

x	extent of candidate grid (vector of xmin, xmax, ymin, ymax)
dimension	integer ncol, nrow
extent	numeric extent xmin,xmax,ymin,ymax
...	ignored
snap	one of "out" (default), "near", or "in"

Details

This works for any grid: the input extent can be within the original, an extension of the original, or completely non-intersecting the original grid.

Value

A list with two components:

extent numeric vector (xmin, xmax, ymin, ymax) - the new extent, snapped to grid alignment

dimension integer vector (ncol, nrow) - the dimension of the modified grid

See Also

align_extent() for just the extent snapping, extent_dimension() for just the dimension calculation

Examples

```
## any arbitrary extent
(x <- c(sort(runif(2, -180, 180)), sort(runif(2, -90, 90))))

vcrop(x, c(360, 180), c(-180, 180, -90, 90))

## crop to a smaller region
vcrop(c(0, 10, 0, 10), c(360, 180), c(-180, 180, -90, 90))

## extend beyond original (snapped to grid)
vcrop(c(-200, 200, -100, 100), c(360, 180), c(-180, 180, -90, 90))
```

---

world_to_geotransform *Create geotransform from world vector*

Description

Convert world vector (centre offset) and x,y spacing to geotransform format.

Usage

```
world_to_geotransform(x)
```

Arguments

x	worldfile parameters, as per geo_world0()

Value

geotransform vector, see geo_transform0()

Examples

```
(wf <- geo_world0(px = c(1, -1), ul = c(0, 0)))
gt <- world_to_geotransform(wf)
geotransform_to_world(gt)
```

Index