Package: vaster (via r-universe)

August 23, 2024

Title Tools for Raster Grid Logic

Version 0.0.2.9004

Description Provides raster grid logic, the grid operations that don't require access to materialized data, i.e. most of them. Grids are arrays with dimension and extent, and many operations are functions of just the dimension 'nrows', 'ncols' or a combination of the dimension and the extent 'xmin', 'xmax', 'ymin', 'ymax'. 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. Cell index, and row,column posiiton exist independently of any other use of a raster grid.

License MIT + file LICENSE

NeedsCompilation yes

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

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

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

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Repository https://hypertidy.r-universe.dev

RemoteUrl https://github.com/hypertidy/vaster

RemoteRef HEAD

RemoteSha 0e934ba049b7eb6520a458eefee8d42fcb81a828

Contents

`adjacencies` *Adjacency, for use in creating area based meshes*

Description

Functions 'bottom left', 'top left', 'bottom right', and 'top right' named by their initials, provide very low level relative positional structures for use in raster logic. These are used to traverse the divide left by area-based rasters which are inherently a discrete value across a finite element. If we want that element as part of a continuous surface we need to find local relative values for its corners. Used in quadmesh and anglr packages, and useful for calculating neighbourhood values.

Usage

```
bl(x)

tl(x)

br(x)

tr(x)

la(x)

ta(x)

ra(x)

ba(x)

image0(x, ...)

image1(x, ...)

text0(x, ...)
```

Arguments

x	matrix
...	arguments passed to image()

Details

bl, tl, br, and tr originally lived in affinity

Value

matrix, padded by one row and one column relative to input

Examples

```
(m <- matrix(1:12, 3))
tl(m)
tr(m)
bl(m)
br(m)
tl(br(m))
image0(tl(br(m)))
text0(tl(br(m)))

## this gives neighbours in adjacent positions
m <- matrix(1:12, ncol = 3, byrow = TRUE)
```

```
matrix(c(t(la(m)), t(ta(m)), t(ra(m)), t(ba(m))), ncol = 4)

## this gives neighbours in all 8 adjacent and diagonal positions
image(matrix(rowMeans(matrix(c(t(la(m)), t(ta(m)), t(ra(m)),
 t(ba(m)), t(bl(m)), t(tl(m)), t(br(m)), t(tr(m))),  ncol = 8), na.rm = TRUE),
 4, 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()

---

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*

Description

Get extent from index values in VRT text.

Usage

```
extent_vrt(x)
```

Arguments

x	url or file path to VRT file

Details

(I can't understand XML tech so I hack the text as lines with strsplit)

Examples

```
#src <- "https://opentopography.s3.sdsc.edu/raster/NASADEM/NASADEM_be.vrt"
#src <- "https://opentopography.s3.sdsc.edu/raster/SRTM_GL1/SRTM_GL1_srtm.vrt"
#ex <- extent_vrt(src)
#op <- par(mar = rep(0, 4))
#plot(range(ex[,1:2]), range(ex[,3:4]), xlab = "", ylab = "", asp = "", type = "n")
#rect(ex[,1], ex[,3], ex[, 2], ex[,4])
#par(op)
```

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)

ymaximum (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 eotransform 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

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 paramers 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 algorith 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

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().

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, extent*

Description

Format properties for the GDAL options.

Usage

```
te(extent)

ts(dimension)

ts_te(dimension, extent)
```

Arguments

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

Value

string formatted for GDAL command line (-te -ts)

Examples

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

vaster_boundary *Grid boundary in native resolution*

Description

currently only return centre coords

Usage

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

Arguments

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

Examples

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

`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 elementx 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 to reduce or extend the area covered, in either dimension. This implies a new extent, snapped to the grain of the origin grid and a new size (dimension in x,y).

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.

Examples

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