Package: affinity (via r-universe)

August 13, 2024

Title Raster Georeferencing, Grid Affine Transforms, Cell Abstraction

Version 0.2.5

Description Tools for raster georeferencing, grid affine transforms, and general raster logic. These functions provide converters between raster specifications, world vector, geotransform, 'RasterIO' window, and 'RasterIO window' in 'sf' package list format. There are functions to offset a matrix by padding any of four corners (useful for vectorizing neighbourhood operations), and helper functions to harvesting user clicks on a graphics device to use for simple georeferencing of images. Methods used are available from <https://en.wikipedia.org/wiki/World_file> and <https://gdal.org/user/raster_data_model.html>.

Depends R (>= 3.2.3)

License GPL-3

LazyData true

LazyDataCompression xz

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

URL https://github.com/hypertidy/affinity

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

Encoding UTF-8

Imports raster, reproj, stats

Suggests rmarkdown, covr, knitr

VignetteBuilder knitr

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

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

RemoteRef HEAD

RemoteSha d55748a57f1abc331974e9a516de0fa8ecf8e1d4

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)

image0(x, ...)

image1(x, ...)

text0(x, ...)
```

Arguments

x	matrix
...	arguments passed to image()

Details

Some tiny functions 'image0', 'image1', 'text0' exist purely to illustrate the ideas in a vignette.

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)))
```

---

affinething *Use affine logic interactively georegister a raster*

Description

User clicks are collected in a controlled way for use by `domath()`.

Usage

```
affinething(x, rgb = FALSE)
```

Arguments

x	a raster
rgb	use RGB plot for a raster with 3 layers

Details

Obtain control points for the simple affine transform (offset and scale) on an ungeoreferenced image.

Value

matrix of x,y coordinates in the space of the current raster extent

Examples

```
## Not run:
library(raster)
r <- raster("my_unreferenced_raster.png")
xy <- affinething(r)  ## click on two points that you know a location of
my_x <- c(1000, 2000)
my_y <- c(-1000, -500)
prj <- "+proj=laea +lon=147 +lat_0=-42" ## use your own map projection, that correspond to my_x/my_y
pt <- cbind(my_x, my_y)
## now convert those control points to an extent for your raster
ex <- domath(pt, xy, r, prj)

## now we can fix up the data
r <- raster::setExtent(r, ex)
raster::projection(r) <- prj
## hooray!

## End(Not run)
```

`assignproj` *Assign projection*

Description

Set the projection of a spatial object.

Usage

```
assignproj(x, proj = "+proj=longlat +datum=WGS84")
```

Arguments

`x`	spatial object for use with `raster::projection()`
`proj`	PROJ.4 string

Value

a spatial object with the projection set

domath *Calculate the math of an affine transform*

Description

Given relative location and absolute locations, convert to an actual real world extent for a matrix of data.

Usage

```
domath(pts, xy, r = NULL, proj = NULL)
```

Arguments

pts	known points of 'xy'
xy	'xy' obtain from affinething
r	raster in use
proj	optional projection, if the pts are longlat and the raster is not

Details

Convert known geographic points with raw graphic control points and a reference raster to an extent for the raster in geography.

Value

raster extent

See Also

affinething()

Examples

```
## not a real example, but the extent we could provide volcano if the second set of points
## described the real world positions of the first set of points within the matrix
domath(cbind(c(147, 148), c(-42, -43)), cbind(c(0.2, 0.3), c(0.1, 0.5)), raster::raster(volcano))
```

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, dim)
```

Arguments

x	extent parameters, c(xmin,xmax,ymin,ymax)
dim	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))
```

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)
```

`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))
```

`monterey` *Monterey Bay elevation*

Description

Extent is in the examples, stolen from rayshader.

Usage

```
monterey
```

Format

An object of class `matrix` (inherits from `array`) with 270 rows and 270 columns.

Details

A matrix 270x270 of topography. Used in `affinething()` examples.

Examples

```
ex <- c(-122.366765, -121.366765, 36.179392, 37.179392)
raster::setExtent(raster::t(raster::raster(monterey)), ex)
```

`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 `raster_io0()`

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 <- raster_io0(c(0L, 0L), src_dim = c(24L, 10L))
rasterio_to_sfio(rio)
```

raster_io *GDAL RasterIO parameter creator*

Description

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

Usage

```
raster_io0(
  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

```
raster_io0(c(0L, 0L), src_dim = c(24L, 10L))
```

raster_to_gt *Geotransform from raster object*

Description

Return the geotransform defining the raster's offset and resolution.

Usage

```
raster_to_gt(x)
```

Arguments

x	raster object (the raster package, extends BasicRaster)

Details

The geotransform vector is six coefficients xmin, xres, yskew, ymax, xskew, yres, values relative to the top left corner of the top left pixel. "yres" the y-spacing is traditionally negative.

Value

a geotransform vector

Examples

```
raster_to_gt(raster::raster(volcano))
```

raster_to_rasterio *RasterIO window from raster object*

Description

Return the RasterIO window vector defining the raster's offset and resolution and dimensions.

Usage

```
raster_to_rasterio(x)

raster_to_sfio(x)
```

Arguments

x	a raster object (BasicRaster, from raster package)

Details

The RasterIO window is a six element vector of offset (x,y), dimension of source (nx0, ny0) and dimension of output (nx, ny).

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_to_sfio()`.

Value

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

Examples

```
raster_to_rasterio(raster::raster(volcano))
```

`raster_to_world` *World vector from raster object.*

Description

Return the world transform defining the raster's offset and resolution.

Usage

```
raster_to_world(x)
```

Arguments

`x`	raster object (the raster package, extends BasicRaster)

Details

The world vector is the values xres, yres, xmin, ymax relative to the centre of the top left pixel. "yres" the y-spacing is traditionally negative.

Value

a geotransform vector

Examples

```
raster_to_world(raster::raster(volcano))
```

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(raster_io0(c(0L, 0L), src_dim = c(24L, 10L))))
```

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