

Package: grout (via r-universe)

October 14, 2026

Title Abstract Raster Tiling Schemes

Version 0.1.0.9001

Description Impose a tiling scheme on a raster grid defined by its dimension and extent. Computes tile counts, pixel offsets (suitable for spatial library windowed reads), and geographic extents for each tile. Also supports standard Web Mercator and geodetic tiling profiles with zoom levels, and provides a spatial map of what each chunk covers for interrogating virtual raster stores. Based on the analysis of Lamb (1994, ISBN-13: 978-0748403158) ``Tiling very large rasters" in 'Advances in GIS Research: Proceedings of the Sixth International Symposium on Spatial Data Handling', volume 1, pages 449-461.

License GPL-3

Encoding UTF-8

Depends R (>= 3.5.0)

Imports graphics, tibble, utils, vaster (>= 0.6.0)

Suggests testthat (>= 3.0.0), wk

URL <https://github.com/hypertidy/grout>

BugReports <https://github.com/hypertidy/grout/issues>

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-09-20 04:51:21 UTC

RemoteUrl <https://github.com/hypertidy/grout>

RemoteRef HEAD

RemoteSha c2e52adb3aad6600d0873772336564db3434f4b6

Contents

grout	2
plot.grout_tiles	3
tile_index	4
tile_spec	5
tile_zoom	6
tiles_from_extent	7
Index	9

grout	<i>Create a tiling scheme from a raster grid specification</i>
-------	--

Description

Given a grid (dimension + extent), compute how it maps onto tiles of a given block size. The result records the tile count in each dimension and the "dangle" - the number of extra pixels that arise when the grid dimensions do not divide evenly into the block size.

Usage

```
grout(  
  dimension,  
  extent = NULL,  
  blocksize = c(256L, 256L),  
  projection = NA_character_  
)
```

Arguments

- dimension integer vector 'c(ncol, nrow)' of the raster grid.
- extent numeric vector 'c(xmin, xmax, ymin, ymax)'. Defaults to 'c(0, ncol, 0, nrow)' (pixel-coordinate space).
- blocksize integer vector 'c(block_ncol, block_nrow)'. Defaults to 'c(256L, 256L)'.
- projection CRS string (e.g. "EPSG:4326"). Stored but not used computationally.

Details

Use [tile_index()] to turn the scheme into a data frame of pixel offsets and spatial extents, one row per tile.

Value

A "grout_tiles" object: a list with * '\$tileraster' - grid spec ('dimension', 'extent', 'projection') of the *tile* grid (one cell per tile). * '\$scheme' - internal "grout_tilescheme" with block sizes, tile counts, and dangle values.

See Also

[tile_index()], [tile_spec()]

Examples

```
## clean fit
grout(c(16, 12), extent = c(0, 16, 0, 12), blocksize = c(4L, 4L))

## dangle in both dimensions
grout(c(15, 13), extent = c(0, 15, 0, 13), blocksize = c(4L, 4L))

## default pixel-coordinate extent
grout(c(87, 61), blocksize = c(12L, 16L))
```

plot.grout_tiles	<i>Plot a tiling scheme</i>
------------------	-----------------------------

Description

Draws each tile as a rectangle (grey border) with the original raster extent overlaid as a dashed red outline.

Usage

```
## S3 method for class 'grout_tiles'
plot(x, ..., add = FALSE, border = "grey", lwd = 2)
```

Arguments

x	a "grout_tiles" object from [grout()].
...	passed to [vaster::plot_extent()].
add	add to the current plot? Default 'FALSE'.
border	colour for the tile borders. Default "grey".
lwd	line width for tile borders. Default '2'.

Value

the input 'x', invisibly

Examples

```
g <- grout(c(44, 30), blocksize = c(12L, 12L))
plot(g)

## overlay a second scheme
g2 <- grout(c(44, 30), blocksize = c(8L, 8L))
plot(g2, add = TRUE, border = "steelblue")
```

tile_index

*Tile index: pixel offsets and spatial extents for each tile***Description**

Returns a data frame with one row per tile in a [grout()] scheme, giving the GDAL-style pixel offset ('offset_x', 'offset_y'), the actual pixel dimensions of each tile ('ncol', 'nrow'), and the geographic extent.

Usage

```
tile_index(x, tile = NULL, clip = TRUE)
```

Arguments

x	a "grout_tiles" object from [grout()].
tile	optional integer vector of 1-based tile indices to return. The default 'NULL' returns every tile in row-major order. Values may be given in any order and may repeat; one row is returned per element, in the order supplied.
clip	clip the edge tiles to the pixels that exist (the default), or with 'FALSE' report every tile at the full block size.

Details

Tiles along the right or bottom edge may be smaller than the block size when there is a "dangle" (the raster dimensions are not an exact multiple of the block size). By default those tiles are clipped to the pixels that actually exist, which is what a windowed read wants. With 'clip = FALSE' every tile reports the full block size and the edge tiles extend past the grid, which is what a chunk of a tiled format actually occupies on disk - a GDAL block or a Zarr chunk at the margin is a whole block, padded. The unclipped counts are not recoverable from the clipped index, so this has to be asked for when the index is built.

Every column is a function of the tile index and the scheme, so the index does not have to be materialized in full. Pass 'tile' to evaluate only the tiles wanted, in the order wanted; the rows returned are identical to 'tile_index(x)[tile,]' but are computed directly rather than by building and subsetting the whole table. This matters for schemes with many tiles, where the full index is large but the tiles of interest are few.

Column layout: * 'tile' - 1-based tile index (row-major, left-to-right top-to-bottom). * 'offset_x', 'offset_y' - 0-based pixel offsets from the top-left corner of the raster, suitable for passing directly to GDAL 'ReadRaster()'. * 'tile_col', 'tile_row' - 1-based tile grid coordinates. * 'ncol', 'nrow' - pixel dimensions of this tile. * 'xmin', 'xmax', 'ymin', 'ymax' - geographic extent of this tile.

Value

a [tibble::tibble()].

Examples

```
g <- grout(c(87, 61), extent = c(0, 1, 0, 1), blocksize = c(32L, 16L))
tile_index(g)

## only the tiles wanted, in the order wanted
tile_index(g, tile = c(4, 1, 9))

## full blocks, including the dangle past the grid edge
tile_index(g, clip = FALSE)

## edge case: one tile
tile_index(grout(c(61, 87), blocksize = c(61L, 87L)))
```

tile_spec	<i>Tile specification for a standard tiling profile</i>
-----------	---

Description

Given a raster grid and a zoom level, find all tiles that intersect the grid extent within a standard global tiling profile (Mercator, geodetic, or native raster).

Usage

```
tile_spec(
  dimension,
  extent,
  zoom = 0L,
  blocksize = c(256L, 256L),
  profile = c("mercator", "geodetic", "raster"),
  crs = NA_character_,
  xyz = FALSE
)
```

Arguments

dimension	integer ‘c(ncol, nrow)’ of the source raster.
extent	numeric ‘c(xmin, xmax, ymin, ymax)’ of the source raster. Must be in the coordinate system of the chosen ‘profile’.
zoom	integer zoom level (0+).
blocksize	tile size in pixels, default ‘c(256L, 256L)’.
profile	one of “mercator”, “geodetic”, “raster”.
crs	CRS string, only used when ‘profile = "raster"’.
xyz	if ‘TRUE’, use XYZ tile row convention (0 at top).

Details

Profiles: * `"mercator"` - Web Mercator (EPSG:3857), global extent +/-20037508.34 m. * `"geodetic"` - geographic coordinates (EPSG:4326), global extent +/-180, +/-90. * `"raster"` - uses the input `'extent'` as the full domain.

`'tile_row'` in the output uses TMS orientation (row 0 at the bottom) by default. Set `'xyz = TRUE'` for XYZ/slippy-map orientation (row 0 at the top).

Value

a data frame with columns `'tile'`, `'tile_col'`, `'tile_row'`, `'zoom'`, `'xmin'`, `'xmax'`, `'ymin'`, `'ymax'`, `'ncol'`, `'nrow'`, `'crs'`.

See Also

`[tile_zoom()]` to determine the appropriate zoom level.

Examples

```
tile_spec(c(8194, 8194), c(140, 155, -45, -30), profile = "geodetic")

tile_spec(c(2048, 248), c(140, 155, -45, -30), zoom = 5,
          profile = "geodetic", blocksize = c(512L, 512L))
```

tile_zoom	<i>Natural maximum zoom for a raster grid</i>
-----------	---

Description

Returns the largest zoom level at which the tile resolution is still coarser than (or equal to) the native raster resolution. This is the zoom at which the data can be served without upsampling.

Usage

```
tile_zoom(
  dimension,
  extent,
  blocksize = c(256L, 256L),
  profile = c("mercator", "geodetic", "raster")
)
```

Arguments

dimension	integer <code>'c(ncol, nrow)'</code> of the source raster.
extent	numeric <code>'c(xmin, xmax, ymin, ymax)'</code> of the source raster. Must be in the coordinate system of the chosen <code>'profile'</code> .
blocksize	tile size in pixels, default <code>'c(256L, 256L)'</code> .
profile	one of <code>"mercator"</code> , <code>"geodetic"</code> , <code>"raster"</code> .

Value

integer zoom level.

Examples

```
tile_zoom(c(8194, 8194), c(140, 155, -45, -30), profile = "geodetic")
```

tiles_from_extent	<i>Which tiles does an extent intersect?</i>
-------------------	--

Description

Solve for the tile indices a region overlaps, without building the tile index. The answer costs a few arithmetic operations regardless of how many tiles the scheme has, because it is computed from the tile grid rather than found by testing every tile.

Usage

```
tiles_from_extent(x, extent, tol = 1e-08)
```

Arguments

x	a "grout_tiles" object from [grout()].
extent	numeric vector 'c(xmin, xmax, ymin, ymax)', in the coordinate system of the scheme.
tol	snapping tolerance, in tile widths. A region edge within 'tol' of a tile boundary is treated as being on it, which keeps the answer stable when an extent has been arrived at by arithmetic rather than given exactly.

Details

The result is intended to be handed straight to [tile_index()]:

```
"" tile_index(x, tile = tiles_from_extent(x, region)) ""
```

Tiles are treated as half open, so a region that only touches a tile boundary does not pick up the tile on the far side of it: a tile owns its left and top edges and not its right and bottom ones. A region with zero width or height (a point) still lands on one tile. Regions are clamped to the scheme, so a region larger than the grid returns every tile, and a region entirely outside it returns 'integer(0)'.

Note that the tile grid can extend past the grid it tiles, when there is a dangle, so tiles can be returned for a region that lies outside the extent passed to [grout()] but inside the tiled area.

Value

integer vector of 1-based tile indices, in increasing (row-major) order, or 'integer(0)' when the region does not overlap the scheme.

See Also

[[tile_index\(\)](#)]

Examples

```
g <- grout(c(87, 61), blocksize = c(12L, 16L))

tiles_from_extent(g, c(20, 40, 20, 40))

## straight into tile_index()
tile_index(g, tile = tiles_from_extent(g, c(20, 40, 20, 40)))

## a whole-grid region is every tile; a region outside is none
length(tiles_from_extent(g, c(0, 87, 0, 61)))
tiles_from_extent(g, c(1000, 2000, 1000, 2000))
```

Index

grout, [2](#)

plot.grout_tiles, [3](#)

tile_index, [4](#)

tile_spec, [5](#)

tile_zoom, [6](#)

tiles_from_extent, [7](#)