Package ‘quadmesh’

April 15, 2020

Type Package
Title Quadrangle Mesh
Version 0.4.5
Description Create surface forms from matrix or ‘raster’ data for flexible plotting and conversion to other mesh types. The functions ‘quadmesh’ or ‘triangmesh’ produce a continuous surface as a ‘mesh3d’ object as used by the ‘rgl’ package. This is used for plotting raster data in 3D (optionally with texture), and allows the application of a map projection without data loss and many processing applications that are restricted by inflexible regular grid rasters. There are discrete forms of these continuous surfaces available with ‘dquadmesh’ and ‘dtriangmesh’ functions.
License GPL-3
LazyData TRUE
RoxygenNote 7.1.0
Depends R (>= 2.10)
Encoding UTF-8
Imports raster, gridBase, viridis, png, sp, geometry, reproj (>= 0.4.0), scales
Suggests datasets, knitr, rmarkdown, palr, rgl, testthat (>= 2.1.0), ncdf4, covr, proj4
URL https://github.com/hypertidy/quadmesh
BugReports https://github.com/hypertidy/quadmesh/issues
VignetteBuilder knitr
ByteCompile true
NeedsCompilation no
Author Michael D. Sumner [aut, cre]
Maintainer Michael D. Sumner <mdsumner@gmail.com>
Repository CRAN
Date/Publication 2020-04-15 08:20:02 UTC
R topics documented:

bary_index ............................................................. 2
etopo ................................................................. 3
llh2xyz ................................................................. 3
mesh_plot ............................................................. 4
qm_as_raster .......................................................... 6
qsc .................................................................... 7
quadmesh ............................................................... 7
triangmesh ............................................................ 9
triangulate_quads .................................................... 11
use_crs ............................................................... 12
worldll ............................................................... 13
xymap ............................................................... 13

Index

bary_index .............................. Barycentric triangle index for interpolation

Description

This function returns the barycentric weight for a grid of coordinates from a geographic raster.

Usage

bary_index(x, coords = NULL, grid = NULL, ...)

Arguments

x  a 'RasterLayer' source
coords  optional input coordinates
grid  target 'RasterLayer', a target regular grid
...  ignored

Details

It's not as fast as raster::projectRaster() (e.g. projectRaster(x, grid)) but it also accepts a coords argument and so can be used for non-regular raster reprojection.

'coords' may be 'NULL' or longitude, latitude in a 2-layer raster brick or stack as with mesh_plot.

Value

RasterLayer
etopo

Examples

```r
library(raster)
p_srs <- "+proj=stere +lat_0=-90 +lat_ts=-71 +datum=WGS84"
polar <- raster(extent(-5e6, 5e6, -5e6, 5e6), crs = p_srs, res = 25000)
etopo <- aggregate(etopo, fact = 4)
index <- bary_index(etopo, grid = polar)
ok <- !is.na(index$idx)
r <- setValues(polar, NA_integer_)
r[ok] <- colSums(matrix(values(etopo)[index$tri[, index$idx[ok]]], nrow = 3) * t(index$p)[, ok])
plot(r)
```

etopo

World topography map

Description

A simplified version of 'Etopo2'. The Etopo2 data set was reduced 20X to create this raster layer of global relief. See code in 'data-raw/topo.R'.

llh2xyz

Angular coordinates to X, Y, Z.

Description

Angular coordinates to X, Y, Z.

Usage

```r
llh2xyz(lonlatheight, rad = 6378137, exag = 1)
```

Arguments

- `lonlatheight`: matrix or data.frame of lon,lat,height values
- `rad`: radius of sphere
- `exag`: exaggeration to apply to height values (added to radius)

Value

matrix
mesh_plot

Mesh plot as a mesh

Description

Convert to a quadmesh and plot in efficient vectorized form using 'grid'.

Usage

```r
mesh_plot(
  x,
  crs = NULL,
  col = NULL,
  add = FALSE,
  zlim = NULL,
  ...,
  coords = NULL
)
```

```
# S3 method for class 'BasicRaster'
mesh_plot(
  x,
  crs = NULL,
  col = NULL,
  add = FALSE,
  zlim = NULL,
  ...,
  coords = NULL
)
```

```
# S3 method for class 'RasterLayer'
mesh_plot(
  x,
  crs = NULL,
  col = NULL,
  add = FALSE,
  zlim = NULL,
  ...,
  coords = NULL
)
```

```
# S3 method for class 'stars'
mesh_plot(
  x,
  crs = NULL,
  col = NULL,
  add = FALSE,
```
Arguments

x  object to convert to mesh and plot

crs  target map projection

col  colours to use, defaults to that used by graphics::image()

add  add to existing plot or start a new one

zlim  absolute range of data to use for colour scaling (if NULL the data range is used)

...  passed through to base::plot

coords  optional input raster of coordinates of each cell, see details
Details

The mesh may be reprojected prior to plotting using the 'crs' argument to define the target map projection in 'PROJ string' format. (There is no "reproject" function for quadmesh, this is performed directly on the x-y coordinates of the 'quadmesh' output). The 'col' argument are mapped to the input applied object data as in 'image', and applied relative to 'zlim' if su.

If coords is supplied, it is currently assumed to be a 2-layer RasterBrick with longitude and latitude as the cell values. These are used to geographically locate the resulting mesh, and will be transformed to the crs if that is supplied. This is modelled on the approach to curvilinear grid data used in the angstroms package. There the function angstroms::romsmap() and ‘angstroms::romscoords()” are used to separate the complicated grid geometry from the grid data itself. A small fudge is applied to extend the coordinates by 1 cell to avoid losing any data due to the half cell outer margin (get in touch if this causes problems!).

Value

nothing, used for the side-effect of creating or adding to a plot

Examples

mesh_plot(worldll)
rr <- raster::crop(worldll, raster::extent(-179, 179, -89, 89))
mesh_plot(rr, crs = "+proj=laea +datum=WGS84")

qm_as_raster

Quadmesh to raster

Description

Approximate re-creation of a raster from a quadmesh.

Usage

qm_as_raster(x, index = NULL)

Arguments

x 'mesh3d' object
index optional index to specify which z coordinate to use as raster values

Details

The raster is populated with the mean of the values at each corner, which is closest to the interpretation used to create mesh3d from rasters. This can be over ridden by setting 'index’ to 1, 2, 3, or 4.

Value

RasterLayer
Examples

qm_as_raster(quadmesh(etopo))

Description

The QSC is a set of six equal area projections for each side of the cube. Here a raw rendition of the cube is returned as six quad primitives in a mesh3d object.

Usage

qsc()

Details

It’s not clear if this is useful.

Value

mesh3d

References

https://github.com/OSGeo/proj.4/wiki/Qsc

Examples

library(rgl); #rgl.clear()
wire3d(qsc())
if ( rgl::rgl.useNULL()) rglwidget()

quadmesh

Create a quad-type mesh for use in rgl.

Description

Convert an object to a mesh3d (rgl::qmesh3d()) quadrangle mesh, with methods for raster::raster() and matrix.
quadmesh

Usage

dquadmesh(
   x,
   z = x,
   na.rm = FALSE,
   ..., 
   texture = NULL,
   texture_filename = NULL
)

## Default S3 method:
dquadmesh(
   x,
   z = x,
   na.rm = FALSE,
   ..., 
   texture = NULL,
   texture_filename = NULL
)

quadmesh(x, z = x, na.rm = FALSE, ..., texture = NULL, texture_filename = NULL)

## S3 method for class 'BasicRaster'
quadmesh(x, z = x, na.rm = FALSE, ..., texture = NULL, texture_filename = NULL)

## S3 method for class 'matrix'
quadmesh(x, z = x, na.rm = FALSE, ..., texture = NULL, texture_filename = NULL)

Arguments

x       raster object for mesh structure
z       raster object for height values
na.rm   remove quads where missing values?
...     ignored
texture    optional input RGB raster, 3-layers
texture_filename    optional input file path for PNG texture

Details

quadmesh() generates the cell-based interpretation of a raster (AREA) but applies a continuous interpretation of the values of the cells to each quad corner. dquadmesh splits the mesh and applies a discrete interpretation directly. Loosely, the quadmesh is a continuous surface and the dquadmesh is free-floating cells, but it's a little more complicated and depends on the options applied. (The interpolation) applied in the quadmesh case is not entirely consistent.

The output is described as a mesh because it is a dense representation of a continuous shape, in this case plane-filling quadrilaterals defined by index of four of the available vertices.
The \( z \) argument defaults to the input \( x \) argument, though may be set to NULL, a constant numeric value, or another raster. If the coordinate system of \( z \) and \( x \) don’t match the \( z \) values are queried by reprojection.

Any raster RGB object (3-layers, ranging in 0-255) may be used as a texture on the resulting mesh3d object. If texture is a palette raster it will be auto-expanded to RGB.

It is not possible to provide rgl with an object of data for texture, it must be a PNG file and so the in-memory texture argument is written out to PNG file (with a message). The location of the file may be set explicitly with texture_filename. Currently it’s not possible to not use the texture object in-memory.

**Value**

mesh3d

**Examples**

```r
library(raster)
data(volcano)
r <- setExtent(raster(volcano), extent(0, 100, 0, 200))
qm <- quadmesh(r)
```

---

**triangmesh**

Create a triangle-type mesh for use in rgl.

**Description**

Convert an object to a mesh3d (\texttt{rgl::tmesh3d()}) triangle mesh, with methods for \texttt{raster::raster()} and \texttt{matrix}.

**Usage**

```r
triangmesh(
  x,
  z = x,
  na.rm = FALSE,
  ..., 
  texture = NULL,
  texture_filename = NULL
)
```

```r
## S3 method for class 'matrix'
triangmesh(
  x,
  z = x,
  na.rm = FALSE,
  ..., 
  texture = NULL,
```
## S3 method for class 'BasicRaster'
triangmesh(
  x,
  z = x,
  na.rm = FALSE,
  ...,  
  texture = NULL,
  texture_filename = NULL
)

dtriangmesh(
  x,
  z = x,
  na.rm = FALSE,
  ...,  
  texture = NULL,
  texture_filename = NULL
)

## Default S3 method:

dtriangmesh(
  x,
  z = x,
  na.rm = FALSE,
  ...,  
  texture = NULL,
  texture_filename = NULL
)

### Arguments

- **x**
  - raster object for mesh structure
- **z**
  - raster object for height values
- **na.rm**
  - remove quads where missing values?
- **...**
  - ignored
- **texture**
  - optional input RGB raster, 3-layers
- **texture_filename**
  - optional input file path for PNG texture

### Details

`triangmesh()` generates the point-based interpretation of a raster (POINT) with the obvious continuous interpretation. `dtriangmesh` splits the mesh so that each primitive is independent. This is more coherent than the analogous distinction for quadmesh, though both will appear the same on creation.
The output is described as a mesh because it is a dense representation of a continuous shape, in this case plane-filling triangles defined by index of three of the available vertices.

The \( z \) argument defaults to the input \( x \) argument, though may be set to NULL, a constant numeric value, or another raster. If the coordinate system of \( z \) and \( x \) don’t match the \( z \) values are queried by reprojection.

Any raster RGB object (3-layers, ranging in 0-255) may be used as a texture on the resulting mesh3d object. It is not possible to provide rgl with an object of data for texture, it must be a PNG file and so the in-memory texture argument is written out to PNG file (with a message). The location of the file may be set explicitly with texture_filename. Currently it’s not possible to not use the texture object in-memory.

Value

mesh3d (primitivetype triangle)

Examples

```r
library(raster)
r <- setExtent(raster(volcano), extent(0, nrow(volcano), 0, ncol(volcano)))
tm <- triangmesh(r)
#rgl::shade3d(tm)

## jitter the mesh just enough to show that they are distinct in the discrete case
a <- dtriangmesh(r)
a$v[3L, ] <- jitter(a$v[3L, ], factor = 10)
##rgl.clear(); rgl::shade3d(a, col = "grey"); aspect3d(1, 1, 0.2); rglwidget()
```

```
triangulate_quads     Triangles from quads
```

Description

Convert quad index to triangles, this converts the ‘rgl mesh3d (ib)’ quad index to the complementary triangle index ‘(ii)’.

Usage

```r
triangulate_quads(quad_index, clockwise = FALSE)
```

Arguments

- `quad_index`: the ‘ib’ index of quads from ‘quadmesh’
- `clockwise`: if true triangles are wound clockwise, if false anticlockwise. This affects which faces rendering engines consider to be the ‘front’ and ‘back’ of the triangle. If your mesh appears ‘inside out’, try the alternative setting.
Details

Triangle pairs from each quad are interleaved in the result, so that neighbour triangles from a single quad are together.

Value

matrix of triangle indices

Examples

```r
triangulate_quads(cbind(c(1, 2, 4, 3), c(3, 4, 6, 5)))
qm <- quadmesh(raster::crop(etopo, raster::extent(140, 160, -50, -30)))
tri <- triangulate_quads(qm$ib)
plot(t(qm$vb))
tri_avg <- colMeans(matrix(qm$vb[3, tri], nrow = 3), na.rm = TRUE)
scl <- function(x) (x - min(x))/diff(range(x))
tri_col <- grey(seq(0, 1, length = 100))[scl(tri_avg) * 99 + 1]
## tri is qm$ib converted to triangles for the same vertex set
polygon(t(qm$vb)[rbind(tri, NA), ])
polygon(t(qm$vb)[rbind(tri, NA), ], col = tri_col)
```

use_crs

In-use coordinate system

Description

Set or return the coordinate system currently in use.

Usage

```r
use_crs(crs = NULL)
```

Arguments

- `crs`: provide PROJ string to set the value

Details

If argument `crs` is NULL, the function returns the current value (which may be ‘NULL”).

Examples

```r
## Not run:
use_crs()
use_crs("+proj=laea +datum=WGS84")
use_crs()
## End(Not run)
```
worldll

**World raster map**

**Description**

A rasterized version of wrld_simpl, created by burning the country polygon ID number into a one-degree world raster. (This is a very out of date polygon data set used for example only). See code in `data-raw/worldll.R`.

xymap

**World map**

**Description**

The world coastline coordinates. A simple matrix of lon, lat, separated by NA.

**Details**

From the maps package, see 'data-raw/xymap.R'.
Index

bary_index, 2

dquadmesh (quadmesh), 7
dtriangmesh (triangmesh), 9

etopo, 3

graphics::image(), 5

llh2xyz, 3

mesh_plot, 4

qm_as_raster, 6
qsc, 7
quadmesh, 7

raster::projectRaster(), 2
raster::raster(), 7, 9
rgl::qmesh3d(), 7
rgl::tmesh3d(), 9

triangmesh, 9
triangulate_quads, 11

use_crs, 12

worldll, 13

xymap, 13