Package: textures (via r-universe)

July 21, 2026

Title Quad Mesh Primitives and Texture Mapping for Grids

Version 0.1.0

Description Generate quad mesh primitives from the compact specification of a regular grid, its dimension and extent. Provides fast generation of mesh indexes and vertices, an unexpanded intermediate form (the grid edge coordinates), and a compact serializable specification for meshes that are generated on demand. Meshes are 'mesh3d' objects as used by the 'rgl' package, constructed without requiring any graphics engine, with support for texture mapping (Heckbert (1986) <doi:10.1109/MCG.1986.276672>) where an image is draped over a mesh whose density is independent of the image resolution. A C++ header library is installed so that other packages may generate mesh components via 'LinkingTo'.

License GPL-3

Encoding UTF-8

LazyData true

Imports graphics, grDevices

Depends R (>= 3.6.0)

Suggests knitr, png, rgl, rmarkdown, spelling, testthat

LinkingTo cpp11

VignetteBuilder knitr

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

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

Config/roxygen2/version 8.0.0

Language en-US

NeedsCompilation yes

Author Michael D. Sumner [aut, cre] (ORCID: <https://orcid.org/0000-0002-2471-7511>)

Maintainer Michael D. Sumner <mdsumner@gmail.com>

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

Date/Publication 2026-07-21 09:30:02 UTC

RemoteUrl https://github.com/cran/textures

RemoteRef HEAD

RemoteSha e55ce16ba7c10792f8fecca0011f6251948f8d67

Contents

`break_mesh` *Break mesh, expand vertex instances*

Description

Break the topology of a mesh by expanding all vertices.

Usage

```
break_mesh(x)
```

Arguments

`x`	mesh3d, from e.g. `quad()`

Details

Data in a mesh and in rgl is inherently *topological*, but we can have primitives that independent, out of an original mesh. Expanding every vertex makes primitives geometrically independent, so attributes (colours, texture coordinates) can be constant per face rather than interpolated between shared vertices - the *discrete* rendering of a *continuous* mesh, at the cost of four times the vertices. This is the `dquadmesh` idea from quadmesh package.

Value

mesh3d

See Also

Other textures: quad(), quad_spec()

Examples

```
(mesh <- quad(c(3, 3)))
## same number of primitives, more vertices (every coordinate)
break_mesh(mesh)
```

---

ga_topo *Topographic image*

Description

Image of Australia as a map, its extent, and map projection.

Usage

```
ga_topo
```

Format

A list with an array, a numeric vector, and a character vector:

img image array with dimension 921,1025,3 - three slices Red, Green, Blue

extent the geographic extent of the array, in metres

crs the map projection of the geographic extent of img

Details

This dataset exists to demonstrate the full pattern: write img to PNG, texture it with [quad_texture()] using extent, and transform the mesh vertices with the crs - the image itself is never resampled.

(The image is web Mercator, aka 'EPSG:3857'. We've kept the proj string because it's the easiest way to modify a projection.)

Provenance

Copyright Commonwealth of Australia (Geoscience Australia) 2016. Creative Commons Attribution 4.0 International Licence.

The image is named 'Australian Topographic Base Map (Web Mercator)' and is from the following Geoscience Australia Web Map Tile Service (WMTS): https://services.ga.gov.au/gis/rest/services/Topographic_Base_Map/MapServer.

Code to obtain the image is in 'data-raw/ga_topo.R' at https://github.com/hypertidy/textures using the wmts package https://github.com/mdsumner/wmts.

`plot.mesh3d` *Plot (2D) for mesh3d*

Description

Draws the x-y geometry of a mesh in base graphics: quads (`ib`) and triangles (`it`) as polygons, segments (`is`) as lines, and bare vertices as points when no primitives are present.

Usage

```
## S3 method for class 'mesh3d'
plot(
  x,
  ...,
  asp = 1,
  add = FALSE,
  axes = TRUE,
  border = "black",
  col = NA,
  alpha = 1,
  lwd = 1,
  lty = 1
)
```

Arguments

`x`	mesh3d object (with any or all of quads, triangles, segments)
`...`	ignored
`asp`	aspect ratio, default 1
`add`	add to an existing plot, default `FALSE`
`axes`	draw axes (when starting a new plot), default `TRUE`
`border`	border colour for polygons, default 'black'
`col`	fill colour for polygons (line colour for segments), default `NA`
`alpha`	transparency in `[0, 1]`, default 1 (opaque)
`lwd`	line width
`lty`	line type

Value

the input mesh, invisibly, called for the side effect of graphics

Examples

```
plot(quad(c(5, 4), extent = c(0, 10, 0, 8)))
```

png_plot3d *Plot a PNG bitmap in 3D*

Description

This is the core idea of the textures package in minimum form: a single quad carrying a full-resolution image, with all interpolation done by the graphics engine.

Usage

```
png_plot3d(pngfile, dim = c(1, 1))
```

Arguments

pngfile	path to a PNG format image file
dim	specify dimensions of quad grid see quad()

Details

Consider that the vertices on each corner can be changed by value arbitrarily, this will affect the way this will be represented geometrically and visually. If the internal mesh of the quad is denser, this provides automatic image reprojection work entirely by the graphics engine.

Value

returns a mesh3d with 1 quad and the image file textured to it, as a side effect creates a 3D interactive plot

Examples

```
file <- system.file("extdata/Rlogo.png", package = "textures")
png_plot3d(file)
```

quad *Quad mesh objects*

Description

Create a simple quad mesh3d object. quad() is the eager convenience form of quad_mesh(quad_spec(...)).

Usage

```
quad(dimension = c(1L, 1L), extent = NULL, ydown = FALSE)

quad_texture(dimension = c(1L, 1L), extent = NULL, ydown = FALSE, texture = "")

segs(dimension = c(1L, 1L), extent = NULL, ydown = FALSE)
```

Arguments

dimension	dimensions of mesh (using matrix() and image() orientation)
extent	optional extent of mesh xmin, xmax, ymin, ymax
ydown	should y-coordinate be counted from top (default FALSE)
texture	file path to PNG image (may not exist)

Details

Use quad() to create a mesh3d object with quad indexes to the vertices, this is defined in the rgl package by qmesh3d() and has elements vb (the homogeneous coordinates 4xn) and ib (the quad index 4xn).

Use segs() to create a mesh3d object with segment indexes, exactly analogous to the mesh created by quad() just only containing the quad edges/segments - note that segments are unique.

The meshColor is currently hardcoded as 'vertices'.

Use quad_texture() to create a mesh3d object additionally with texcoords and texture properties.

Value

mesh3d with quads and material texture settings as per inputs

See Also

Other textures: break_mesh(), quad_spec()

Examples

```
qm <- quad()
## orientation is low to high, x then y
qm <- quad(dim(volcano))
scl <- function(x) (x - min(x, na.rm = TRUE))/diff(range(x, na.rm = TRUE))
qm$meshColor <- "faces"
qm$material$color <- hcl.colors(12, "YlOrRd", rev = TRUE)[scl(volcano) * 11 + 1]
if (requireNamespace("rgl")) {
 rgl::plot3d(qm)
 }
```

---

quad_index *Quad mesh primitives*

Description

Generate the components of a quad mesh from the compact specification of a regular grid: the dimension (number of cells nx, ny) and optionally the extent (xmin, xmax, ymin, ymax).

Usage

```
quad_index(dimension, ydown = FALSE)

quad_edges(dimension, extent = NULL, ydown = FALSE)

quad_vertex(dimension, extent = NULL, ydown = FALSE)
```

Arguments

dimension	number of cells in the grid (nx, ny), a single value is recycled
ydown	should the y coordinate be counted from the top (image/raster orientation), default FALSE
extent	extent of the grid c(xmin, xmax, ymin, ymax), default is the unit square

Details

quad_index() gives the index of the four corner vertices of every cell, a 4 x (nx * ny) matrix of 1-based vertex indexes. Quads are in cell order, x fastest (the matrix() and image() convention, and raster cell order when ydown = TRUE). Winding is counter-clockwise for y-up grids.

quad_edges() gives the unexpanded form of the vertices, the x and y coordinates of the cell edges as two vectors of length nx + 1 and ny + 1. This is the compact intermediate: the full vertex set is their outer expansion, and for many purposes (transform bounds, axis coordinates, texture alignment) the margins are all that is needed.

quad_vertex() gives the materialized vertices, a (nx + 1) * (ny + 1) row matrix of x, y coordinates, x fastest, matching the index in quad_index().

For very large grids where the vertex count or index length exceeds the integer maximum, quad_index() returns a matrix of doubles (exact for integer values to 2^53), which R subsetting and 'mesh3d' usage accept natively. Take care not to apply as.integer() to such an index.

Value

quad_index() a 4-row matrix of vertex indexes (integer, or double for very large grids), quad_edges() a list with x and y edge coordinate vectors, quad_vertex() a 2-column matrix of vertex coordinates @family textures

Examples

```
quad_index(c(2, 3))
quad_edges(c(2, 3), extent = c(0, 10, 0, 20))
quad_vertex(c(2, 3), extent = c(0, 10, 0, 20))
```

quad_spec *Compact quad mesh specification*

Description

A quad mesh on a regular grid is completely determined by a tiny recipe: the grid dimension, its extent, and the y orientation. `quad_spec()` records that recipe as a plain, serializable list without materializing any vertices or indexes.

Usage

```
quad_spec(
  dimension = c(1L, 1L),
  extent = NULL,
  ydown = FALSE,
  crs = NULL,
  texture = NULL
)

quad_mesh(x, ...)

## S3 method for class 'quad_spec'
quad_mesh(x, ...)

## S3 method for class 'quad_spec'
as.mesh3d(x, ...)

## S3 method for class 'quad_spec'
print(x, ...)
```

Arguments

`dimension`	number of cells in the grid (nx, ny), a single value is recycled
`extent`	extent of the grid `c(xmin, xmax, ymin, ymax)`, default is the unit square
`ydown`	should the y coordinate be counted from the top, default `FALSE`
`crs`	optional coordinate reference system, stored but not used by textures itself
`texture`	optional file path to a PNG image to texture onto the mesh when materialized
`x`	a quad_spec object
`...`	ignored, or passed between methods

Details

`quad_mesh()` materializes the specification as a 'mesh3d' object (as used by the 'rgl' package, but constructed without it). If the spec carries a `texture` image file path, texture coordinates are included: these are the extent-normalized vertex coordinates, so they remain valid for any mesh whose vertices lie in the extent.

With the 'rgl' package installed, `as.mesh3d()` works directly on a quad_spec (registered on-demand, 'rgl' is not required).

The mesh density given by `dimension` is independent of the pixel dimension of any `texture` image: a coarse mesh may carry a full-resolution image, the graphics engine interpolates within each quad.

A spec can be stored, serialized, and sent where a materialized mesh cannot sensibly be - materialization is deferred to the consumer.

Value

`quad_spec()` a list with class 'quad_spec', `quad_mesh()` a mesh3d object, the print method returns its input invisibly

See Also

Other textures: `break_mesh()`, `quad()`

Examples

```
spec <- quad_spec(c(20, 10), extent = c(100, 160, -60, -30))
spec
mesh <- quad_mesh(spec)
str(mesh$vb)
```

---

set_scene *rgl defaults*

Description

Quick defaults for rgl static plot

Usage

```
set_scene(
  interactive = FALSE,
  zoom = 0.5,
  phi = 0,
  theta = 0,
  light_phi = -45,
  light_theta = 0
)
```

Arguments

`interactive`	mouse interactive plot, default `FALSE`
`zoom`	zoom for rgl::view3d default `0.5` which is closer than rgl 1
`phi`, `theta`	polar coordinates, passed to `rgl::view3d()`
`light_phi`, `light_theta`	polar coordinates, passed to `rgl::light3d()` as `phi` and `theta` respectively

Details

This function sets the size of the window to 1024x1024, sets the view position at directly vertical `phi = 0, theta = 0`, makes the view non-interactive (zoom is enabled, but no pivot or pan). It turns off the lights and puts a new light in front of the viewer (to avoid shiny glare), sets the aspect ratio to 'iso' ("fill the box"), and attempts to 'bringtotop', but I think that has to happen interactively with `rgl::rgl.bringtotop()` (especially for animating or snapshotting scenes to file).

`phi` and `theta` use `0` and `0` respectively, phi is different from rgl's default in order to look straight down on the quad (along the z axis)

`light_phi` and `light_theta` use `-45` and `0` respectively, phi is different from rg's default to put the light source forwards (y+) from the viewer when looking straight down

Value

nothing

Examples

```
## see README and in-dev examples in rough-examples.R
rgl::plot3d(rnorm(10), rnorm(10), rnorm(1)); set_scene()
```

Index