Package: anglr (via r-universe)

September 20, 2024

Type Package

Title Mesh Topology and Visualization for Spatial Data

Version 0.8.0.9003

Description Gives direct access to generic 3D tools and provides a full suite of mesh-creation and 3D plotting functions. By extending the 'rgl' package conversion and visualization functions for the 'mesh3d' class a wide variety of complex spatial data can be brought into 3D scenes. These tools allow for spatial raster, polygons, and lines that are common in 'GIS' contexts to be converted into mesh forms with high flexibility and the ability to integrate disparate data types. Vector and raster data can be seamlessly combined as meshes, and surfaces can be set to have material properties based on data values or with image textures. Textures and other data combinations use projection transformations to map between coordinate systems, and objects can be easily visualized in an interactive scene at any stage. This package relies on the 'RTriangle' package for high-quality triangular meshing which is licensed restrictively under 'CC BY-NC-SA 4.0'.

License CC BY-NC-SA 4.0

Depends R (>= 3.4.0)

Imports crsmeta (>= 0.3.0), dplyr, gridBase, magrittr, palr, png, polyclip, raster, reproj (>= 0.4.2), rgl (>= 0.107.8), rlang, RTriangle, scales, silicate (>= 0.6.1), sp, terrainmeshr, tibble, unjoin, utils, viridis, colourvalues

Suggests covr, ggplot2, graticule, knitr, purrr, rmarkdown, sfheaders, testthat (>= 2.1.0), trip, gibble

Roxygen list(markdown = TRUE)

SystemRequirements PROJ library, OpenGL and GLU Library

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

VignetteBuilder knitr

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

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

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

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

RemoteRef HEAD

RemoteSha 6f611d76c9a40a4c7b98455021bc93d03d674462

Contents

anglr-package *Mesh topology and visualization for spatial mesh structures.*

Description

The anglr package gives direct access to generic 3D tools and provides a full suite of mesh-creation and 3D plotting functions. By extending the rgl package conversion and visualization functions for the `mesh3d` class a wide variety of complex spatial data can be brought into 3D scenes. These tools allow for spatial raster, polygons, and lines that are common in GIS contexts to be converted into mesh forms with high flexibility and the ability to integrate disparate data types. Vector and raster data can be seamlessly combined as meshes, and surfaces can be set to have material properties based on data values or with image textures. Textures and other data combinations use projection transformations to map between coordinate systems, and objects can be easily visualized in an interactive scene at any stage.

Details

The 'anglr' package show-cases extended features for *geo*-spatial data by extending and supporting the data models of the silicate package. Any kind of spatial data is intended to be supported, not just the geographic ones:

- coordinates beyond X and Y, or longitude and latitude
- storing attributes on vertices, primitives, branches (parts), or objects
- topology and geometry are distinguishable and not conflated
- spatial data can be represented as a graph of spatial primitives
- polygons as true surfaces, not just path structures with a 2D-only region-filling rule
- TBD higher dimensional primitives are possible
- TBD n-dimensional rasters with curvilinear coordinates, and the discrete-continuous distinction

Licensing

The anglr package is released with license CC BY-NC-SA 4.0 to match the one dependency RTriangle. Please note and respect the license of the RTriangle package used by the `DEL()` or `DEL0()` functions in anglr, and invoked within 3D plot methods. These return high-quality constrained Delaunay triangulations of polygonal regions, with the ability to control mesh characteristics including maximum triangle area, minimum internal angle, and conformance to the Delaunay criterion. If you are interested in a less restrictive license for high-quality meshing in R please get involved with the laridae package which aims to provide access to CGAL.

Creation

`as.mesh3d`	coercion function to convert most spatial data to mesh forms
`SC`	and other silicate models are all supported, including the structural forms SC0, TRI0, PATH0
`Spatial`	most spatial types can be used directly including raster and sf

DEL	create a mostly-Delaunay shape-preserving constrained triangulation

Merging disparate data

as.mesh3d	includes an image_texture argument to map an Raster RGB image onto surfaces
copy_down	copy Z values (from a raster, vector field, or constant) onto the vertices of a mesh

Plotting

As much as possible plotting will represent the true nature of the data given.

mesh_plot	plot in 2D, including curvilinear reprojections with rasters
plot3d	and related 3D plot functions in rgl can be used directly on most input types
globe	convert X,Y planar or angular to 3D on the surface of a globe, based on the data in longitude-latitude form
plot3d.SC	plot 1D topology in 3D geometry space
plot3d.TRI	plot 2D topology in 3D geometry space (DEL or TRI)

Options and technicalities

There is an option set for the maximum number of triangles that can be generated by the DEL() or DEL0() models when using the max_area argument. Inspect the limit with getOption("anglr.max.triangles") or set a new limit with options(anglr.max.triangles = <new limit>).

In terms of the RTriangle::triangulate() function the max_area argument controls and masks the a argument for RTriangle. It's possible to pass in values for q, Y, j, D, S, V, and Q - but we don't recommend experimenting with these unless you know what they are for. There is a coarse check for a limit on the number of triangles that can be created but general caution is advised when experimenting.

---

as.mesh3d *Convert to mesh object*

Description

The [as.mesh3d()][rgl::as.mesh3d()] generic function converts various objects to mesh3d objects. Methods are added to support a variety of spatial formats, which will include triangles or quads according to their inherent form. For quad-types the argument triangles can be specified to generate triangles from quads. The majority of conversions are done by model functions in the silicate package, and anglr adds models DEL(), DEL0(), and QUAD().

The mesh3d format is the rgl workhorse behind plot3d(), wire3d(), persp3d() and dot3d().'

A method for a numeric matrix is included, as are methods for sf, sp, raster, RTriangle, and silicate types. and for a matrix.

Usage

```
## S3 method for class 'TRI'
as.mesh3d(
  x,
  z,
  smooth = FALSE,
  normals = NULL,
  texcoords = NULL,
  ...,
  keep_all = TRUE,
  image_texture = NULL,
  meshColor = "faces"
)

## S3 method for class 'TRI0'
as.mesh3d(
  x,
  z,
  smooth = FALSE,
  normals = NULL,
  texcoords = NULL,
  ...,
  keep_all = TRUE,
  image_texture = NULL,
  meshColor = "faces"
)

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

## S3 method for class 'RasterLayer'
as.mesh3d(...)

## S3 method for class 'BasicRaster'
as.mesh3d(
  x,
  triangles = TRUE,
  smooth = FALSE,
  normals = NULL,
  texcoords = NULL,
  ...,
  keep_all = TRUE,
  image_texture = NULL,
  meshColor = "faces",
  max_triangles = NULL
)

## S3 method for class 'QUAD'
```

```
as.mesh3d(
  x,
  triangles = FALSE,
  smooth = FALSE,
  normals = NULL,
  texcoords = NULL,
  ...,
  keep_all = TRUE,
  image_texture = NULL,
  meshColor = "faces"
)

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

## S3 method for class 'SC0'
as.mesh3d(x, ...)

## S3 method for class 'sfc'
as.mesh3d(
  x,
  triangles = FALSE,
  smooth = FALSE,
  normals = NULL,
  texcoords = NULL,
  ...,
  keep_all = TRUE,
  image_texture = NULL,
  meshColor = "faces"
)

## S3 method for class 'sf'
as.mesh3d(x, ..., meshColor = "faces")

## S3 method for class 'Spatial'
as.mesh3d(
  x,
  triangles = FALSE,
  smooth = FALSE,
  normals = NULL,
  texcoords = NULL,
  ...,
  keep_all = TRUE,
  image_texture = NULL,
  meshColor = "faces"
)

## S3 method for class 'matrix'
```

```
as.mesh3d(
  x,
  triangles = FALSE,
  smooth = FALSE,
  normals = NULL,
  texcoords = NULL,
  ...,
  keep_all = TRUE,
  image_texture = NULL,
  meshColor = "faces"
)

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

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

## S3 method for class 'SC0'
as.mesh3d(x, ...)

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

## S3 method for class 'PATH0'
as.mesh3d(x, ...)

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

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

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

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

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

Arguments

x	a surface-alike, a matrix, or spatial object from raster, sp, sf, trip, or silicate
z	numeric vector or raster object (see details)
smooth	Whether to average normals at vertices for a smooth appearance.
normals	User-specified normals at each vertex. Requires `smooth = FALSE`.

texcoords	Texture coordinates at each vertex.
...	arguments collected and passed to rgl::tmesh3d() as the material argument
keep_all	whether to keep non-visible triangles
image_texture	an rgb object to texture the surface
meshColor	how should colours be interpreted? 'vertices' or 'faces', for more details see rgl::tmesh3d.
triangles	for quad input types, the quads may optionally be split into triangles
max_triangles	limit on triangles to create, passed to terrainmeshr

Details

When converting a matrix to mesh3d it is considered to be quad-based (area interpretation) within `xmin = 0, xmax = nrow(x), ymin = 0, ymax = ncol(x)`. Note that this differs from the `[0, 1, 0, 1]` interpretation of `image()`, but shares its orientation. Raster-types from the raster package are interpreted in the `t(ranspose), y-flip` orientation used by `plot(raster::raster(matrix))`.

The conversion function `as.mesh3d()` consolidates code from quadmesh and angstroms packages where the basic facilities were developed. The function as.mesh3d() is imported from rgl and re-exported, and understands all of the surface types from sf, sp, raster, and silicate, and can accept a raw matrix as input.

When creating a surface mesh there is an optional `z` argument to extract elevation values from a raster, and/or an `image_texture` argument to drape an image from a raster RGB object onto the surface. Map projections are automatically resolved to the coordinate system of the `x` argument.

Value

a mesh3d object

Implicit versus explicit topology

We support conversion to mesh for strictly linear types such as sf 'POLYGON', 'MULTIPOLYGON', 'MULTILINESTRING', 'LINESTRING' and their sp counterparts 'SpatialPolygons' and 'SpatialLines'. Even polygons are only *implicit surfaces* and so conversion and plotting functions try to reflect this inherent nature as much as possible. A mesh is inherently a surface, and so the method for polygons or lines will first call a surface-generating function, `silicate::TRI0()` or `DEL0()` in order to created the required primitives, while plot3d() will not do this. The key goal is *flexibility*, and so we can call a meshing function as.mesh3d() (does conversion) or persp3d() (a plot function, but requires conversion to surface) and they will choose an interpretation. An non-formal guideline is to use the cheapest method possible, i.e. `silicate::TRI0()`.

Much of the above is open for discussion, so please get in touch! Use the issues tab or ping me on twitter to clarify or discuss anything.

Elevation values with `z`

The 'z' argument can be a constant value or a vector of values to be used for each vertex. Alternatively, it may be a spatial raster object from which 'z' values are derived. If not set, the vertex 'z_' value from TRI/TRI0 is used, otherwise z = 0' is assumed.

Textures

Please see the documentation for rgl textures in `vignette("rgl", package = "rgl")`. The most important detail is that the `$material$color` property of a `mesh3d` not be set to "black" ("#000000" or equivalent), or it will not be visible at all. The only way to add a texture in mesh3d is as a PNG file on-disk, so anglr functions take an in-memory object and create the file if needed.

See Also

dot3d wire3d persp3d plot3d

Examples

```
sf <- silicate::minimal_mesh
#sf <- silicate::inlandwaters
x <- silicate::TRI(sf)
library(rgl)
clear3d(); plot3d(x); view3d(phi = -10)
## simple convention to carry feature colours
sf$color_ <- c("firebrick", "dodgerblue")
clear3d(); plot3d(silicate::TRI(sf)); view3d(phi = -10)

# material properties for $material are collected in ...
# and will override the 'color_' mech
x$object$color_ <- "black"
clear3d(); plot3d(as.mesh3d(x, color = rainbow(14)))

## we cannot assume TRI triangles relate to features simply
##  but sometimes it does  (always does for TRI0)
cols <- c("black", "grey")[c(rep(1, 12), c(2, 2))]
clear3d(); plot3d(as.mesh3d(x, color = cols))

## smear by vertices meshColor
cols <- c("black", "grey")
clear3d(); plot3d(as.mesh3d(x, color = cols), meshColor = "vertices")

## other material properties
clear3d()
plot3d(as.mesh3d(x, color = cols, specular = "black"), meshColor = "vertices")
clear3d()
plot3d(as.mesh3d(x, color = cols, front = "lines", lwd = 5), meshColor = "vertices")
clear3d()
plot3d(as.mesh3d(x, color = viridis::viridis(20), alpha = 0.3), meshColor = "faces")
clear3d()
plot3d(as.mesh3d(x, color = viridis::viridis(5), alpha = 0.3), meshColor = "vertices")

# TRI0 - index is stored structurally, not relations
x0 <- silicate::TRI0(sf)
clear3d(); plot3d(x0); view3d(phi = -10)

# (TRI0 - it *is* guaranteed that triangle order is native)
clear3d(); plot3d(as.mesh3d(x0,  color = rainbow(14)))
```

```
## arbitrarily drape polygons over raster
r <- raster::setExtent(raster::raster(volcano), raster::extent(-0.1, 1.1, -0.1, 1.1))
clear3d();shade3d(as.mesh3d(DEL(silicate::minimal_mesh, max_area = 0.001), z =r))
aspect3d(1, 1, 0.5)


library(rgl)
## get sf extent
ext <- sf_extent(silicate::inlandwaters)
r1 <- raster::setExtent(raster::raster(volcano), ext)
clear3d();shade3d(as.mesh3d(DEL(silicate::inlandwaters, max_area = 1e9), z =r1))
aspect3d(1, 1, .2)

## fake news
rgl::wire3d(as.mesh3d(r1))
```

`as_pslg` *Planar Straight Line Graph*

Description

Create a 'pslg' which is a mesh of segments used in the RTriangle package.

Usage

```
as_pslg(x)

## Default S3 method:
as_pslg(x)
```

Arguments

x	data model (understood by `SC0()`)

Details

The pslg is constructed from from unique vertices x, y and their segments. In the context of triangulating the functions `DEL0()` and `DEL()` do more to ensure that input polygons have their holes culled out (or classified by invisible triangles.)

Value

object of class RTriangle::pslg from the RTriangle package

Examples

```
data("minimal_mesh", package = "silicate")
as_pslg(minimal_mesh)
```

auto_3d *Auto aspect ratio*

Description

Automatically modify the aspect ratio of a scene to rescale drastically different data ranges into a cube.

Usage

```
auto_3d(...)
```

Arguments

...	unused, check for input of arguments which are ignored with a message

Details

This is a simple alias to rgl::aspect3d(1).

This is typically used to rescale data in different units, for example longitude and latitude in degrees and elevation in metres.

Note that running `rgl::aspect3d("iso")` which show the realistic ratio of the plot axes, ignoring units. Running this function is equivalent to `rgl::aspectd(1)` (or `rgl::aspect(x = 1, y = 1, z = 1)`) which sets the *apparent* ratios of the current bounding box.

Value

the original value of `rgl::par3d()` before update

Examples

```
topo <- copy_down(silicate::SC(simpleworld), gebco)
plot3d(topo)
## update aspect ratio to be an apparent cube, not a needle
auto_3d()
```

cad_tas *Cadastre and Contour*

Description

Coincident polygon cadastre layer and line contour layer.

Details

These two sf layers are cad_tas a sf polygons layer of a small region of cadastral parcels, and cont_tas a sf lines layer of the same region with elevation contours of the underlying topography.

These layers are fused together in an in-progress example.

`cont_tas` has an elevation value for each line in `cont_tas[["ELEVATION"]]`.

These data sets are derived from the LIST Cadastral Parcels and LIST Contours 5m from theLIST Copyright State of Tasmania. These data are distributed under the Creative Commons Attribution 3.0 Australia License.

Examples

```
plot3d(cont_tas)

plot3d(copy_down(silicate::SC0(cont_tas), "ELEVATION"))
auto_3d()
```

`copy_down` *Copy down values to vertices*

Description

Copy down provides ways to transfer object level data values to vertex level.

Usage

```
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'mesh3d'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'SC'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'SC0'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'TRI'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'TRI0'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'DEL0'
copy_down(x, z = NULL, ..., .id = "z_")
```

```
## S3 method for class 'PATH'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'PATH0'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'ARC'
copy_down(x, z = NULL, ..., .id = "z_")

## S3 method for class 'QUAD'
copy_down(x, z = NULL, ..., .id = "z_")
```

Arguments

x	a mesh3d or a silicate object
z	object specifying values to copy down, a vector of values, a column name, a raster (see details)
...	currently ignored
.id	character value, the name of the resulting column in the vertices, default is "z_"

Details

Various methods are used depending on the second argument z.

If z is a raster (BasicRaster) a numeric value for each vertex is found by bilinear interpolation using raster::extract(raster, vertex, method = "bilinear"). Vertices are transformed into the space used by the raster if possible.

If z is a character value, that column is taken from the object table.

The .id argument must be character and exist as a column name in the object table.

If z is a vector or a constant value it's simply copied down.

No checking is done on the type of the result, and so there's nothing to stop the use of the recycling rule to expand out values, and nothing to stop the use of non numeric values being copied down.

Use silicate::TRI0() or DEL0() or silicate::SC0() to convert various spatial formats into suitable forms for this function.

Value

a mesh3d or silicate model with vertex values copied to vertices (depending on the input argument 'x')

Examples

```
library(raster)
r <- raster(volcano)
cl <- silicate::SC(rasterToContour(r))
plot3d(copy_down(cl, r))
## looks funny?
auto_3d()
```

```
sc <- copy_down(SC0(cont_tas), "ELEVATION")
sc$object$color_ <- hcl.colors(nrow(sc$object), "YlOrRd")
 plot3d(sc)

 ## a planar straight line graph with x, y (UTM) and ELEVATION (metres)
 sc
```

`cst10` *Antarctic coastline*

Description

Antartica features and coastline, with somewhat spurious precision.

Usage

```
cst10
```

Format

An object of class `SpatialPolygonsDataFrame` with 372 rows and 2 columns.

Details

In sp format. Interesting for exploring precision issues with DEL0() see issue 7.

Examples

```
p <- PATH0(cst10)
# DEL0(p)  ## fails as does DEL0(cst10)
# fails at 14 and crashes R in DEL0() in R version 4.0.0 RC (2020-04-17 r78247) on
# windows
p$vertex$x_ <- signif(p$vertex$x_, 13)
p$vertex$y_ <- signif(p$vertex$y_, 13)
DEL0(p)

## compare 14993 unique vertices after rounding to
silicate::sc_vertex(p)     ## 21875

silicate::sc_vertex(cst10) ## 21875
```

DEL *Convert object to a constrained-Delaunay triangulation*

Description

This *relational-form* Delaunay-based triangulation model is analogous to the 'TRI' model in the silicate package and formally extends the class of that model. A primitives-based shape-constrained triangulation. The Delaunay model is the *mostly Delaunay* scheme used by the provable-quality meshers.

Usage

```
DEL(x, ..., max_area = NULL)

## Default S3 method:
DEL(x, ..., max_area = NULL)

## S3 method for class 'PATH0'
DEL(x, ..., max_area = NULL)

## S3 method for class 'TRI'
DEL(x, ..., max_area = NULL)

## S3 method for class 'TRI0'
DEL(x, ..., max_area = NULL)

## S3 method for class 'SC'
DEL(x, ..., max_area = NULL)

## S3 method for class 'SC0'
DEL(x, ..., max_area = NULL)

## S3 method for class 'PATH'
DEL(x, ..., max_area = NULL)
```

Arguments

x	input model
...	passed to the underlying Triangle library, see RTriangle::triangulate()
max_area	the maximum area of a triangle

Details

Compare DEL() to its *structural-form* counterpart DEL0(). DEL() records a visible property on the triangle table, and this is queried by other functions for the status of triangles that belonged to a hole within a surface. These triangle are obviously useful, so they are kept but default to visible = FALSE and so are not plotted.

The Delaunay model is a constrained triangulation with a variety of constraint and qualification types. The Delaunay model has the odd but defining characteristic of not being always consistent with the Delaunay criterion. Edge inclusion is non-negotiable, but other constraints include (limit, or avoid) Steiner vertex insertion, a limit on the maximum area of a triangle, minimum triangle angle and strict adherence to the Delaunay criterion.

The Delaunay criterion forms the basis of this model, and is its defining characteristic without being strictly adhered to. This is awkward to describe but is the key property. From Wikipedia: The "Delaunay triangulation (also known as a Delone triangulation) for a given set P of discrete points in a plane is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid sliver triangles. . . . The Delaunay triangulation corresponds to the dual graph of the Voronoi diagram of P" https://en.wikipedia.org/wiki/Delaunay_triangulation.

This strict criterion is *relaxed* in small measure, to ensure that all edge-inputs are preserved and to allow further constraints such as triangle size and internal angle to be specified.

Value

DEL model

Warning

Please take care with the `max_area` argument. The units are not taken into account, the value refers only to the planar area of the x/y coordinates as they are so this is not a real world area, but a mathematical property of the data. There is a safety check for a very large number of triangles, and this may be overridden by replying 'Yes' to the prompt.

Topology

The DEL model cannot currently mesh point features, and it cannot mesh linear features if they include z or other vertex attributes - for now use `DEL0()` which can do those.

Licensing

The anglr package is released with license CC BY-NC-SA 4.0 to match the one dependency RTriangle. Please note and respect the license of the RTriangle package used by the `DEL()` or `DEL0()` functions in anglr, and invoked within 3D plot methods. These return high-quality constrained Delaunay triangulations of polygonal regions, with the ability to control mesh characteristics including maximum triangle area, minimum internal angle, and conformance to the Delaunay criterion. If you are interested in a less restrictive license for high-quality meshing in R please get involved with the laridae package which aims to provide access to CGAL.

See Also

DEL0 TRI TRI0

Examples

```
plot3d(DEL(simpleworld))
```

DEL0 *Convert object to a constrained-Delaunay triangulation*

Description

This *structural-form* Delaunay-based triangulation model is analogous to the TRI() model in the silicate package and formally extends the class of that model. A primitives-based shape-constrained triangulation. The Delaunay model is the *mostly Delaunay* scheme used by the provable-quality meshers.

Usage

```
DEL0(x, ..., max_area = NULL)

## S3 method for class 'DEL'
DEL0(x, ..., max_area = NULL)

## Default S3 method:
DEL0(x, ..., max_area = NULL)

## S3 method for class 'SC'
DEL0(x, ..., max_area = NULL)

## S3 method for class 'SC0'
DEL0(x, ..., max_area = NULL)

## S3 method for class 'TRI'
DEL0(x, ..., max_area = NULL)

## S3 method for class 'TRI0'
DEL0(x, ..., max_area = NULL)

## S3 method for class 'ARC'
DEL0(x, ..., max_area = NULL)

## S3 method for class 'PATH'
DEL0(x, ..., max_area = NULL)

## S3 method for class 'PATH0'
DEL0(x, ..., max_area = NULL)

## S3 method for class 'BasicRaster'
DEL0(x, ..., max_triangles = NULL)
```

Arguments

x	object of class PATH0 or understood by PATH0()

...	ignored
max_area	the maximum area of a triangle
max_triangles	limit on triangles to create, passed to terrainmeshr

Details

This is a more compact form of the *relational-form* DEL() model.

Value

DEL0 class

Topology

Note that for explicitly linear features, these still use a post-meshing identification for which triangles belong in which feature. This can't make sense for many line layers, but we leave it for now.

For point features, the mesher unproblematically creates a triangulation in the convex hull of the points, any attributes names z_, m_, or t_ are automatically interpolated and include in the output. See the help for RTriangle::triangulate() for how this works via the $PA element.

Note that for a raster input the terrainmeshr package is used to determine a sensible number of triangles based on local curvature. To avoid creating this adative mesh and use as.mesh3d(QUAD(raster)) to get quad primitives or as.mesh3d(QUAD(raster), triangles = TRUE) to get triangle primitives directly from raster cells.

Licensing

The anglr package is released with license CC BY-NC-SA 4.0 to match the one dependency RTriangle. Please note and respect the license of the RTriangle package used by the DEL() or DEL0() functions in anglr, and invoked within 3D plot methods. These return high-quality constrained Delaunay triangulations of polygonal regions, with the ability to control mesh characteristics including maximum triangle area, minimum internal angle, and conformance to the Delaunay criterion. If you are interested in a less restrictive license for high-quality meshing in R please get involved with the laridae package which aims to provide access to CGAL.

See Also

DEL

Examples

```
a <- DEL0(cad_tas)
plot(a)

## ---- intepolate via triangulation, sample points from volcano
rgl::clear3d()
n <- 150
max_area <- .005 ## we working in x 0,1 y 0,1
library(anglr)
```

```
library(dplyr)
d <-
  data.frame(x = runif(n), y = runif(n), multipoint_id = 1) %>%
  dplyr::mutate(
    z = raster::extract(raster::raster(volcano), cbind(x, y)),
    multipoint_id = 1
  )

mesh <- DEL0(
  sfheaders::sf_multipoint(d, x = "x", y = "y", z = "z",
    multipoint_id = "multipoint_id"), max_area = max_area)

plot3d(mesh , color = "darkgrey", specular = "darkgrey") #sample(grey.colors(5)))
```

---

dot3d *Draw a mesh as points in 3D*

Description

Draw points with rgl from any mesh-alike or shape3d classed object. Produces a 3D scatterplot like that of `rgl::points3d()`, but from a mesh-alike object.

Usage

```
## S3 method for class 'sf'
dot3d(x, ...)

## S3 method for class 'sfc'
dot3d(x, ...)

## S3 method for class 'Spatial'
dot3d(x, ...)

## S3 method for class 'matrix'
dot3d(x, ...)

## S3 method for class 'BasicRaster'
dot3d(x, ...)

## S3 method for class 'sc'
dot3d(x, ...)

## S3 method for class 'SC'
dot3d(x, ...)

## S3 method for class 'SC0'
dot3d(x, ...)
```

```
## S3 method for class 'triangulation'
dot3d(x, ...)
```

Arguments

x	sc, sp, sf, raster, trip, or any other model understood by anglr/silicate
...	pass material3d properties to rgl

Details

The class mesh3d extends 'shape3d' and allows methods to plot non-surface properties. Note that `dot3d()` will always add to an existing scene.

It is not currently *technically defined or clear* how colour properties are mapped to dots by default ... there is a problem of what property to use from features that share the same vertex, and we have put that aside and erred on the side of inaccuracy in favour of getting a pretty plot (hopefully). (Properties that come later - lower rows - win, I think.

(For some reason `size` is not vectorized like `col` is, but this is not explored in detail from an anglr view).

See Also

as.mesh3d persp3d wire3d plot3d shade3d

Examples

```
dot3d(cad_tas)
dot3d(volcano, size = 10)
auto_3d()


rgl::open3d()
## from ?persp
y <- x <- seq(-10, 10, length= 80)
z <- outer(x, y,
     function(x, y) {
        r <- sqrt(x^2+y^2); 10 * sin(r)/r
        })
dot3d(z)
dot3d(raster::raster(volcano), size = 10)
auto_3d()

dot3d(silicate::SC(cad_tas))


rgl::open3d()
dot3d(as.mesh3d(copy_down(DEL(cad_tas, max_area = 1e3), "CID")))
auto_3d()
```

gebco *world elevation raster*

Description

A simple raster map of world topography, elevation relative to sea level in metres. Source data is Gebco 2014, converted to a much reduced 1 degree resolution global map.

Details

Data downloaded from GEBCO 2014 (0.0083 degrees = 30sec arcmin resolution) and set at resolution 1 degrees. GEBCO 2014.

Examples

```
data("gebco", package = "anglr")
library(silicate)
laea <- "+proj=laea +lon_0=147 +lat_0=-42"
longlat <- "+proj=longlat +datum=WGS84"
x <- SC(simpleworld) %>% copy_down(gebco + 500)
plot3d(x); rgl::aspect3d(1, 1, 0.07)
```

globe *Geocentric (XYZ) coordinates*

Description

Convert longitude/latitude coordinates to geocentric coordinates.

Usage

```
globe(x, ...)

## S3 method for class 'mesh3d'
globe(x, gproj = NULL, ...)

## Default S3 method:
globe(x, gproj = "+proj=geocent +datum=WGS84", ...)
```

Arguments

x	a silicate model or mesh3d
...	arguments to methods (none used)
gproj	Geocentric PROJ.4 string, defaults to WGS84

Details

With silicate data checks are made for the projection in use, but not for mesh3d. In that case data are assumed to be 'longitude,latitude,elevation'.

Value

object with vertices table modified

Examples

```
data(simpleworld)
g <- globe(silicate::PATH(as(simpleworld, "SpatialLinesDataFrame")))
if (interactive()) {
 plot(g, lwd = 3)
 plot3d(g)
 rgl::open3d()
 wire3d(globe(DEL0(gebco*50)), col = "grey");
 rgl::spheres3d(0, 0, 0, rad = 6000000,col = "aliceblue")
}
```

---

mesh_plot *Plot a mesh surface in 2D*

Description

Draw a 2D interpretation of a mesh object, or a mesh-alike object. This is very fast and can be used to created *approximately* continuously varying surface plots.

Usage

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

## S3 method for class 'mesh3d'
mesh_plot(
  x,
  col = NULL,
  add = FALSE,
  zlim = NULL,
  ...,
  coords = NULL,
```

```
  crs = NULL
)

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

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

## Default S3 method:
mesh_plot(
  x,
  col = NULL,
  add = FALSE,
  zlim = NULL,
  ...,
  coords = NULL,
  crs = NULL
)

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

Arguments

x	object to convert to mesh and plot
col	colours to use, defaults to that used by graphics::image()
add	add to existing plot or start a new one
zlim	unimplemented (was used in 'quadmesh::mesh_plot')
...	passed through to base::plot
coords	optional input raster of coordinates of each cell, see details
crs	target map projection

Details

The input is treated as a mesh and plotted in vectorized form using 'grid'.

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 inputdata as in graphics::image().

The coords argument only applies to a raster object. The crs argument only applies to a spatial object that has a crs projection metadata string understood by anglr (works, but still work in progress). There is a change from the previous quadmesh::mesh_plot() function that requires both crs and coords to be named. In quadmesh, crs was the second argument to the mesh_plot() function and so in usage was normally not named.

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.

If the input is a mesh3d and has a material texture image this is *approximated* by averaging the RGB values of each primitive's corner into a constant colour for that face. If you would like to avoid this texture colour, set the 'mesh3d$material$color' property to NULL.

Value

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

---

persp3d *persp3d*

Description

Plot surface

Usage

```
## S3 method for class 'TRI'
persp3d(x, ..., add = FALSE)

## S3 method for class 'TRI0'
persp3d(x, ..., add = FALSE)

## S3 method for class 'BasicRaster'
persp3d(x, ..., add = FALSE)

## S3 method for class 'DEL'
persp3d(x, ..., add = FALSE)

## S3 method for class 'DEL0'
persp3d(x, ..., add = FALSE)

## S3 method for class 'QUAD'
persp3d(x, ..., add = FALSE)

## S3 method for class 'matrix'
persp3d(x, ..., add = FALSE)

## S3 method for class 'sf'
persp3d(x, ..., add = FALSE)

## S3 method for class 'sfc'
persp3d(x, ..., add = FALSE)

## S3 method for class 'Spatial'
persp3d(x, ..., add = FALSE)

## S3 method for class 'triangulation'
persp3d(x, ..., add = FALSE)
```

Arguments

x	sp, sf, raster, trip, or any other model understood by anglr/silicate
...	pass material3d properties to rgl
add	add to existing plot or start a new one (the default)

See Also

plot3d as.mesh3d wire3d dot3d shade3d

plot3d *3D object plot*

Description

This is the workhorse function for anglr, the idea is that just about anything can be plotted in a 3D scene, polygons, lines, rasters, matrix. These objects from sp, sf, raster, trip, and silicate should all work.

Usage

```
## S3 method for class 'TRI'
plot3d(x, ...)

## S3 method for class 'TRI0'
plot3d(x, ...)

## S3 method for class 'DEL'
plot3d(x, ...)

## S3 method for class 'DEL0'
plot3d(x, ...)

## S3 method for class 'QUAD'
plot3d(x, ...)

## S3 method for class 'matrix'
plot3d(x, ...)

## S3 method for class 'BasicRaster'
plot3d(x, ...)

## S3 method for class 'sc'
plot3d(x, ...)

## S3 method for class 'SC'
plot3d(x, ..., add = FALSE)

## S3 method for class 'SC0'
plot3d(x, ..., add = FALSE)

## S3 method for class 'PATH'
plot3d(x, ..., add = FALSE)

## S3 method for class 'sf'
plot3d(x, ..., add = FALSE)
```

```
## S3 method for class 'sfc'
plot3d(x, ..., add = FALSE)

## S3 method for class 'triangulation'
plot3d(x, ..., add = FALSE)

## S3 method for class 'Spatial'
plot3d(x, ..., add = FALSE)

## S3 method for class 'trip'
plot3d(x, ..., add = FALSE)

## S3 method for class 'ARC'
plot3d(x, ..., add = FALSE)
```

Arguments

x	silicate model, SC, TRI, ARC, or PATH
...	passed to material properties
add	add to plot or not

Details

The function `plot3d()` covers the full suite of plotting functions from `rgl::plot3d()` for meshes, points, and lines. This main function includes the family of `dot3d()`, `wire3d()`, and `persp3d()` and each works with matrix, raster, sf, sp, trip, RTriangle, and silicate models. Each of the mesh-surface forms rely on `as.mesh3d()` conversion behind the scenes, whereas `plot3d()` for the linear types (sf, sp, trip, and from silicate SC, SC0, PATH, PATH0, and ARC0) all are plotted using rgl segments without going through a triangulated surface form. This reflects their underlying topology when it comes to 3D visualization and analysis.

If the scene looks funny the aspect ratio might be poor, we've decided not to automatically update this with normal plots, but running `auto_3d()` will attempt to set a reasonable aspect ratio. It can also be used to set exaggerations in different axes.

For SC edges are matched to their object/s. One object's properties is applied as colour. If `color_` column is present on the data object table it is used.

If the argument 'color' is used, this is passed down to the rgl plot function - and will be applied per primitive, not per silicate object. This provides flexibility but does require knowledge of the underlying structures in use.

Value

rgl shape3d types (note that "segment3d" is currently an imaginary shape3d type)

See Also

wire3d as.mesh3d persp3d dot3d shade3d

Examples

```
library(silicate)
cad_tas$color_ <- rainbow(nrow(cad_tas))
x <- SC(cad_tas)
plot3d(x)

## plot3d anything
plot3d(volcano)
wire3d(volcano)
dot3d(volcano)


plot3d(cad_tas)
persp3d(cad_tas)
wire3d(cad_tas)
dot3d(cad_tas)

## add Z elevation to an sf polygon in a mesh
plot3d(copy_down(as.mesh3d(silicate::minimal_mesh), raster::raster(volcano)))

## but make it much more interesting

plot3d(copy_down(as.mesh3d(DEL(silicate::minimal_mesh, max_area = 0.0001)),
  raster::raster(-volcano)), col = c("black", "orange")); auto_3d()
wire3d(silicate::minimal_mesh)
```

QUAD *QUAD model*

Description

The QUAD model is a silicate-like model for raster data, with implicit geometry.

Usage

```
QUAD(x, ...)

## S3 method for class 'matrix'
QUAD(x, ...)

## S3 method for class 'BasicRaster'
QUAD(x, ...)
```

Arguments

x	raster alike, or a matrix
...	ignored

Details

The object table only stores the raster extent, and the pixel values are on the 'quad' table. This is only supported for single-layer 2D regular rasters.

The 'color_' idiom works, but must be put on the '$quad' table. Very much still in-development.

Value

QUAD model

Examples

```
qq <-  QUAD(raster::raster(volcano))
mesh_plot(qq)
qq$quad$color_ <- rep(c("black", "white"), length.out = nrow(qq$quad))
mesh_plot(qq)
qq$quad$color_ <- palr::image_pal(qq$quad$value, col = grey.colors(10))
mesh_plot(qq)
```

---

reproj *Coordinate reprojection*

Description

The `reproj()` function is imported from the `reproj::reproj()` package and re-exported.

---

sf_data_zoo *sf data frame zoo.*

Description

Each kind of geometry in an sf data frame, in a list.

sf_extent *Extent of simple features*

Description

Get Extent of sf objects

Usage

```
sf_extent(x)
```

Arguments

x	sf object

Details

This function exists to avoid raster needing the sf package. We don't need GDAL, GEOS, and all the other fun stuff to find four numbers that describe coordinate range, so this is a workaround.

Value

a raster Extent

Examples

```
sf_extent(silicate::inlandwaters)
```

shade3d *Draw a mesh as surfaces in 3D*

Description

Draw surfaces with rgl from any shape3d classed object. Produces a 3D surface plot from a mesh-alike object.

Usage

```
## S3 method for class 'TRI'
shade3d(x, ...)

## S3 method for class 'TRI0'
shade3d(x, ...)

## S3 method for class 'PATH'
shade3d(x, ...)
```

```
## S3 method for class 'PATH0'
shade3d(x, ...)

## S3 method for class 'DEL'
shade3d(x, ...)

## S3 method for class 'DEL0'
shade3d(x, ...)

## S3 method for class 'QUAD'
shade3d(x, ...)

## S3 method for class 'BasicRaster'
shade3d(x, ...)

## S3 method for class 'matrix'
shade3d(x, ...)

## S3 method for class 'sfc'
shade3d(x, ...)

## S3 method for class 'sf'
shade3d(x, ...)

## S3 method for class 'SC'
shade3d(x, ...)

## S3 method for class 'SC0'
shade3d(x, ...)

## S3 method for class 'Spatial'
shade3d(x, ...)

## S3 method for class 'triangulation'
shade3d(x, ...)
```

Arguments

x	sp, sf, raster, or any other surface model understood by anglr/silicate
...	pass material3d properties to rgl

Details

Objects that are not explicitly surfaces will be triangulated in order to produce the mesh. Whether this is a good idea or not is an open question, and some conversions will fail due to "extra" attributes like z or time stored on vertices. Polygons are only implicit surfaces but these are usually unproblematic to triangulate so this is done.

See Also

plot3d as.mesh3d persp3d dot3d wire3d mesh_plot

Examples

```
rgl::open3d()
shade3d(volcano)


## create a globe plot of land areas with elevation
rgl::open3d()
world <- copy_down(DEL(simpleworld, max_area = 0.5), gebco * 50)
shade3d(globe(world), specular = "black", color = "white")
rgl::spheres3d(0, 0, 0, radius = 6378000, col = "dodgerblue", alpha = 0.75)
rgl::bg3d("black")
```

---

silicate-models *silicate models*

Description

The anglr functions DEL(), DEL0() and QUAD() extend the models of the silicate package. In particular DEL() and DEL0() are high quality Delaunay-constrained triangulation meshers analogous to the ear-cutting algorithms used by silicate::TRI() and silicate::TRI0().

Arguments

x	any data format understood by the model
...	unused

Details

All models in silicate are imported by anglr and re-exported.

See Also

DEL() DEL0() QUAD() silicate::PATH() silicate::TRI() silicate::ARC() silicate::SC() silicate::PATH0() silicate::TRI0() silicate::SC0()

simpleworld *simple world*

Description

A simple polygon map of world sovereign countries, a modified copy of the rnaturalearth countries110 (see data-raw/simpleworld.R for details).

Examples

```
DEL(simpleworld[1:10, ])
```

TRI.QUAD *TRI model extensions*

Description

TRI model from silicate is extended by methods for QUAD.

Usage

```
## S3 method for class 'QUAD'
TRI(x, ...)
```

Arguments

x	QUAD
...	reserved

Value

a TRI model, as per silicate package

Examples

```
library(anglr)
library(raster)
v <- volcano[1:10, 1:6]
r <- setExtent(raster(v), extent(0, nrow(v), 0, ncol(v)))
a <- QUAD(r)
x <- copy_down(TRI(a), r)
nrow(x$vertex)
sc <- silicate::SC(x)
#mesh <- DEL(sc, max_area = .2)
#mesh <- copy_down(mesh, r)
#nrow(mesh$vertex)
```

wire3d *Draw a mesh as line segments in 3D*

Description

Draw line segments with rgl from any shape3d classed object. Produces a 3D scatterplot like that produced by `rgl::plot3d()`, but from a mesh-alike object.

Usage

```
## S3 method for class 'sc'
wire3d(x, ...)

## S3 method for class 'TRI'
wire3d(x, ...)

## S3 method for class 'TRI0'
wire3d(x, ...)

## S3 method for class 'DEL'
wire3d(x, ...)

## S3 method for class 'DEL0'
wire3d(x, ...)

## S3 method for class 'QUAD'
wire3d(x, ...)

## S3 method for class 'matrix'
wire3d(x, ...)

## S3 method for class 'sf'
wire3d(x, ...)

## S3 method for class 'Spatial'
wire3d(x, ...)

## S3 method for class 'triangulation'
wire3d(x, ...)

## S3 method for class 'trip'
wire3d(x, ...)

## S3 method for class 'BasicRaster'
wire3d(x, ...)
```

Arguments

x	sp, sf, raster, trip, or any other model understood by anglr/silicate
...	pass material3d properties to rgl

Details

Objects that are not explicitly surfaces will be triangulated in order to produce the mesh. Whether this is a good idea or not is an open question.

It is not currently *technically defined or clear* how colour properties are mapped to line segments by default ... there is a problem of what property to use from features that share the same vertex or edge, and we have put that aside and erred on the side of inaccuracy in favour of getting a pretty plot (hopefully). (Properties that come later - lower rows - win, I think.

See Also

plot3d as.mesh3d persp3d dot3d shade3d

Index