

Package: gris (via r-universe)

October 15, 2026

Type Package

Title A Flexible Geometry and Topology Model for Spatial Data in R

Version 0.1.0

Author Michael D. Sumner

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

Description A flexible geometry data model for R.

License GPL-3

ByteCompile Yes

LazyData TRUE

Imports dplyr, geometry, RTriangle, raster, rgl, sp, spbabel, tibble

Suggests knitr, maptools, rworldmap, testthat, rmarkdown, covr

VignetteBuilder knitr

RoxygenNote 5.0.1

Config/pak/sysreqs cmake libfreetype6-dev libgdal-dev gdal-bin libgeos-dev libglu1-mesa-dev make texlive libpng-dev libuv1-dev libgl1-mesa-dev libproj-dev libsqlite3-dev zlib1g-dev

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

Date/Publication 2026-02-11 03:39:21 UTC

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

RemoteRef HEAD

RemoteSha 4a5cc1fac3b9d9afdaa58bb39a6b38d1ff15050c

Contents

gris-package	2
as.gris.triangulation	4
as.SpatialPolygonsDataFrame	4
brick2col	5
geocentric	5
grasp	6
gris	6

gris.Spatial	7
grisFromFortify	8
grisFromRasterGeom	8
pointInTriangle	9
q3d	9
t3d	10
triangulate	10
Index	12

gris-package	<i>gris: A flexible geometry data model for R.</i>
--------------	--

Description

The gris package provides a relational table model for **spatial vector** data in R. *Spatial vector data* are topology-based discrete spatial units such as areas (including polygons), lines and points.

Details

These kinds of data are defined in many different ways in various R packages, but the sp package has provided the most comprehensive and standard model to match the traditional GIS layer forms.

Gris features

- provide a clear decoupling of topology from geometry (see section "Gris topology" below)
- transcend the X/Y-only limitation for vertex attributes
- allow multiple topological types in individual layers (i.e. Manifold or MapInfo drawings)
- provide a flexible basis for conversion between other formats and domain-specific structures

I. Creation

<code>gris.Spatial</code>	create from <code>Spatial*DataFrame</code>
<code>grisFromRasterGeom</code>	create from <code>geom</code> format
<code>grisFromFortify</code>	create from <code>fortify</code> format
<code>as.gris.triangulation</code>	create from <code>triangulation</code>

II. Plotting

<code>plot.gris</code>	plot a gris object as polypath, lines, points, or triangles
<code>plot3d.gris</code>	plot a gris object in 3d

III. Manipulation

<code>[.gris</code>	extract a subset of a gris object
<code>pointInTriangle</code>	identify which triangle points fall in

IV. Conversion to other types

<code>as.SpatialPolygonsDataFrame.gris</code>	coerce to SpatialPolygonsDataFrame, SpatialLinesDataFrame, SpatialPointsDataFrame
<code>triangulate.gris</code>	add triangle primitives to a gris object

V. Miscellaneous

<code>brick2col</code>	create hex value colours from an RGB <code>RasterBrick</code>
------------------------	---

Gris topology

Topology defines the way that space is filled by 0D (points), 1D (lines), 2D (areas - or polygons) objects. (Higher dimensional topologies are possible but not explored yet.)

Geometry defines an actual instantiation of an object in a given coordinate system.

Examples of the difference are

- polygons defining continental outlines on the Earth, these shapes are 2D topologically, but geometrically can be plotted in the plane (in long-lat, or projected coordinates) or "draped" around a curve, such as the Earth as a globe or across a topographic landscape
- lines defining animal tracks in space time, these 1D shapes can have **geometric** vertex attributes such as X, Y, Z, time, temperature, etc.

Topology dimension may be 0D, 1D, 2D, or higher and geometry dimension is not necessarily coupled to this, or restricted to 2D.

Note that certain degeneracies are possible and may be important. A time series of point estimates in space may not have any sensible geometric values at all, but the topological description provided by that time series is a real structure.

```
as.gris.triangulation as.gris
```

Description

as.gris

Usage

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

Arguments

x	triangulation (package RTriangle)
type	mesh, triangle polygons, or triangle edges

```
as.SpatialPolygonsDataFrame
```

Return a list of Polygon, Line or matrix from gris

Description

Return a list of Polygon, Line or matrix from gris

Spatial classes are defined in the [sp](#) package.

Usage

```
as.SpatialPolygonsDataFrame(x, ...)
```

```
## S3 method for class 'gris'
as.SpatialPolygonsDataFrame(x, ...)
```

Arguments

x	gris object
...	ignored
type	return type: pp-oly, l-line, p-oints
x	gris object

brick2col	<i>Title</i>
-----------	--------------

Description

Title

Usage

brick2col(x)

Arguments

x RasterBrick

Value

hex colour character vector

geocentric	<i>Geocentric coordinates.</i>
------------	--------------------------------

Description

Transform to geocentric coordinates.

Usage

geocentric(x, crs = "+proj=geocent +ellps=WGS84")

Arguments

x table of long, lat, and optionally "height"
crs
exag exaggeration to apply to height values (added to radius)

Value

tibble

grasp	<i>convert from spbabel sptable to gris</i>
-------	---

Description

convert from spbabel sptable to gris

Usage

```
grasp(dat1, map1)
```

gris	<i>gris</i>
------	-------------

Description

gris

Usage

```
gris(x, ...)

## S3 method for class 'gris'
x[i, j, drop = FALSE]

## S3 method for class 'gris'
print(x, ..., n = NULL, width = NULL)

## S3 method for class 'gris'
plot(x, y, triangles = FALSE, ...)

as.gris(x, ...)
```

Arguments

x	Spatial* object
...	unused
i	indices for extract
j	indices for extract
drop	ignored
n	n
width	width
y	y

Value

gris

Methods

gris implements these base methods:

plot Plots the object according to its topology type, or by a chosen type. This method follows base plotting with types "p" points, "l" lines, and adds "pp" for polygons

print Prints the object and vertices tables with a summary of the topology linkages

[Subset as an object table on attributes, propagates down through the geometry links to return a subsetting gris object

gris.Spatial	<i>Convert Spatial*DataFrame objects to gris</i>
--------------	--

Description

All available Spatial*DataFrame types are supported, and are interpreted via the Branch model. The gris function can ingest [SpatialPolygonsDataFrame](#), [SpatialLinesDataFrame](#), [SpatialPointsDataFrame](#), and [SpatialMultiPointsDataFrame](#) objects.

Usage

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

Arguments

x	Spatial* object
...	not used

Details

See vignette("branch-vs-primitives")

Value

gris

grisFromFortify *Not yet implemented*

Description

Not yet implemented

Usage

```
grisFromFortify(x)
```

Arguments

x

Value

gris

Examples

```
## Not run:  
data(cmu)  
  
sp <- as.SpatialPolygonsDataFrame(cmu)  
gm <- raster::geom(sp)  
  
## End(Not run)
```

grisFromRasterGeom *Not yet implemented*

Description

Not yet implemented

Usage

```
grisFromRasterGeom(x)
```

Arguments

x

Value

gris

pointInTriangle	<i>Point in triangle test.</i>
-----------------	--------------------------------

Description

Point in triangle test.

Usage

pointInTriangle(x, pt)

pointBary(x, pt)

Arguments

x	gris object with triangulation
pt	matrix of xy coordinates

Value

for PointInTriangle, the index of triangle for each point, in vertex_ form for PointBary, the table of barycentric coordinates and vertex_index

q3d	<i>Template quad 3d object</i>
-----	--------------------------------

Description

Template quad 3d object

Usage

q3d

Format

see rgl oh3d

t3d	<i>Template triangle 3d object</i>
-----	------------------------------------

Description

Template triangle 3d object

Usage

t3d

Format

see rgl tetrahedron3d

triangulate	<i>Add triangle primitives</i>
-------------	--------------------------------

Description

Add triangle primitives

Usage

triangulate(x, ...)

Arguments

x	gris object
...	passed on to triangulate , as arguments after the input pslg

Details

only a has been extensively explored, this is the minimum triangle area in the units of the coordinates

Examples

```
## Not run:
library(gris)
library(maptools)
data(wrld_simpl)
gall <- gris(wrld_simpl)
g <- gall[gall$NAME %in% c("Australia", "New Zealand"), ]
g$v <- g$v %>% mutate(x = ifelse(x < 0, x + 360, x))
```

```
gT <- gris::triangulate(g)
plot(g)
plot(gT, triangles = TRUE, add = TRUE)

## End(Not run)
```

Index

- * **datasets**
 - q3d, [9](#)
 - t3d, [10](#)
 - [.gris, [3](#)
 - [.gris (gris), [6](#)
- as.gris (gris), [6](#)
- as.gris.triangulation, [2](#), [4](#)
- as.SpatialPolygonsDataFrame, [4](#)
- as.SpatialPolygonsDataFrame.gris, [3](#)
- brick2col, [3](#), [5](#)
- fortify, [2](#)
- geocentric, [5](#)
- geom, [2](#)
- grasp, [6](#)
- gris, [6](#)
- gris-package, [2](#)
- gris.Spatial, [2](#), [7](#)
- grisFromFortify, [2](#), [8](#)
- grisFromRasterGeom, [2](#), [8](#)
- plot.gris, [2](#)
- plot.gris (gris), [6](#)
- plot3d.gris, [2](#)
- pointBary (pointInTriangle), [9](#)
- pointInTriangle, [3](#), [9](#)
- print.gris (gris), [6](#)
- pslg, [10](#)
- q3d, [9](#)
- RasterBrick, [3](#)
- sp, [4](#)
- SpatialLinesDataFrame, [7](#)
- SpatialMultiPointsDataFrame, [7](#)
- SpatialPointsDataFrame, [7](#)
- SpatialPolygonsDataFrame, [7](#)
- t3d, [10](#)
- triangulate, [10](#), [10](#)
- triangulate.gris, [3](#)
- triangulation, [2](#)