Package ‘silicate’

November 13, 2020

Title Common Forms for Complex Hierarchical and Relational Data Structures

Version 0.7.0

Description Generate common data forms for complex data suitable for conversions and transmission by decomposition as paths or primitives. Paths are sequentially-linked records, primitives are basic atomic elements and both can model many forms and be grouped into hierarchical structures. The universal models ‘SCO’ (structural) and ‘SC’ (labelled, relational) are composed of edges and can represent any hierarchical form. Specialist models ‘PATH’, ‘ARC’ and ‘TRI’ provide the most common intermediate forms used for converting from one form to another. The methods are inspired by the simplicial complex <https://en.wikipedia.org/wiki/Simplicial_complex> and provide intermediate forms that relate spatial data structures to this mathematical construct.

Depends R (>= 3.4.0)

License GPL-3

Encoding UTF-8

LazyData true

Suggests covr, knitr, rmarkdown, sp, testthat (>= 2.1.0), trip, vdiffrr

RoxygenNote 7.1.1

Imports dplyr, gibble (>= 0.4.0), methods, purrr, rlang, decido, tibble, unjoin (>= 0.1.0), grDevices, graphics, stats, magrittr, gridBase, crsmeta (>= 0.3.0)

VignetteBuilder knitr

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

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

NeedsCompilation no

Author Michael D. Sumner [aut, cre],
John Corbett [ctb] (the original inspiration),
Simon Wotherspoon [ctb],
Kent Johnson [dtc],
Mark Padgham [aut]
Maintainer  Michael D. Sumner <mdsumner@gmail.com>
Repository  CRAN
Date/Publication  2020-11-13 11:10:03 UTC

R topics documented:

ARC ................................................................. 3
dplyr-methods ..................................................... 4
ear_gc ............................................................. 4
flight_tracks ...................................................... 5
inlandwaters ..................................................... 5
minimal_mesh .................................................... 6
mmesh ............................................................ 6
PATH ............................................................... 7
PATH0 .............................................................. 8
plot.SC ........................................................... 9
polymesh ........................................................ 10
print.sc ........................................................ 10
routes ............................................................ 11
SC ............................................................... 11
SC0 ............................................................... 12
sc_arc ............................................................ 12
sc_colours ....................................................... 13
sc_coord ........................................................ 14
sc_edge .......................................................... 16
sc_node ........................................................ 17
sc_object ......................................................... 18
sc_path .......................................................... 19
sc_segment ....................................................... 22
sc_uid ........................................................... 22
sc_vertex ......................................................... 23
sfzoo .............................................................. 24
silicate .......................................................... 25
TRI ............................................................... 25
TRI0 ............................................................. 26
tri_area .......................................................... 28

Index 29
Description

Arcs are unique paths that connect nodes. In a polygon layer with shared boundaries, the arcs are the linear features that have no branches.

Usage

\[
\text{ARC}(x, \ldots)
\]

## Default S3 method:
\[
\text{ARC}(x, \ldots)
\]

## S3 method for class 'PATH'
\[
\text{ARC}(x, \ldots)
\]

Arguments

- \(x\): input model
- \(\ldots\): arguments passed to methods

Details

Nodes are the vertices where three or more arcs meet. An arc can exist without including any nodes, a path that has no neighbouring relationship with another path.

This is not the same terminology as used by other systems, such as "arc-node". The arc\_link\_vertex mapping is inherently ordered, but we don't consider order of arcs. Duplicated arcs (i.e. complementary turns around neighbouring polygons) are not kept. The object\_link\_arc mapping records which arc belongs to the objects, so feature polygons can in theory be reconstructed within objects by tracing arc\_link\_vertex start and end point identity.

Value

ARC model

Examples

\[
a <- \text{ARC(minimal\_mesh)}
\]
\[
\text{sc\_arc}(a)
\]
\[
\text{sc\_arc(minimal\_mesh)}
\]
**dpolyr-methods**

*Dplyr methods for silicate objects*

**Description**

Filter an SC model, currently only `dplyr::filter` for SC is available.

**Usage**

```r
## S3 method for class 'SC'
filter(.data, ...)
```

**Arguments**

- `.data` data object of class SC
- `...` passed to `dplyr::filter`

**Details**

Apply expressions as if used on the object table. See `sc_object(x)` for that form.

Currently all the vertices are still kept, so the model (and any plots) include the filtered edges as well as undifferentiated points. This is likely to change ...

**Value**

an `SC()` model, with some parts filtered out

**Examples**

```r
library(dplyr)
sc <- SC(inlandwaters)
plot(filter(sc, Province == "Tasmania"))
plot(filter(sc, Province %in% c("Victoria", "South Australia", "New South Wales")))
plot(filter(SC(minimal_mesh), a == 1))
```

---

**ear_gc**

*Geometry collection of triangles*

**Description**

A 'sfc_GEOMETRYCOLLECTION' of four-cornered triangles ('POLYGON') created by ear cutting the North Carolina polygon from sf.

**Examples**

```
TRI0(ear_gc)
```
**flight_tracks**

**Flight tracks**

**Description**

A data set flight tracks in XYZM form, a form of 4D tracks. Primarily to explore the use of silicate as able to represent this topologically, and to experiment with auto-time-based plotting in anglr.

**Details**

Provided by Kent Johnson (kent37) in a github discussion where the data was attached in a zip file. Original form (in extdata/flight_tracks) is a XYZM LINESTRING shapefile containing 144 flight tracks of aircraft departing runway 33L at Boston Logan airport on January 27, 2017. Data is from an ADS-B recorder. Each point includes lat, lon, altitude in feet and time in North American Eastern Standard Time (EST).

Converted via sf into silicate::PATH normal form, see (data-raw/flight_tracks.R).

**inlandwaters**

**Inland waters, for parts of Australia, and New Caledonia.**

**Description**

The inland waters are lakes and inland waters presenting as holes within the bounded regions of Australian (and New Caledonian) provinces.

**Details**

This is an extract from the old Manifold DVD. It is in sf format. The features have variables ID and Province:

- "103841"Australian Capital Territory
- "103842"New Caledonia
- "103843"New South Wales
- "103846"South Australia
- "103847"Tasmania
- "103848"Victoria

There’s no good reason that New Caledonia is included and not Queensland (for example) it’s just what happened doing a quick crop and extract with the mouse. Lord Howe Island and Macquarie Island are both present, as part of New South Wales and Tasmania respectively.
Examples

```r
path <- PATH(inlandwaters)
plot(path)
obj <- split(path$path_link_vertex, path$path_link_vertex$path_)
cl <- grDevices::colors()[-1L]
cols <- sample(cl, length(obj), replace = length(obj) > length(cl))
op <- par(mfrow = grDevices::n2mfrow(length(obj)), mar = rep(0, 4))
funplot <- function(ob, vert, col) {
  vx <- c("x_", "y_")
  plot(dplyr::inner_join(ob, vert, "vertex_")[vx], col = col, type = "l", axes = FALSE)
}
junk <- lapply(seq_along(obj),
  function(a) {
    funplot(obj[[a]], path$vertex, cols[a])
    invisible(NULL)
  })
par(op)
```

---

**minimal_mesh**  
*Minimal mesh.*

**Description**

The simplest pairing of two polygons with one shared edge. One polygon contains a hole and a concavity, the other is a simply convex. This is composed of four "arcs", one around each polygon, one for the shared edge, and one for the isolated hole. There are two nodes, the endpoints of the single shared edge.

**Examples**

```r
arc <- ARC(minimal_mesh)
plot(arc)
sc_arc(arc)
sc_node(arc)
```

---

**mmesh**  
*Deprecated data set.*

**Description**

This data set is in legacy format and will be removed. A couple of polygons with a single shared edge between them, in PRIMITIVE form.
Description

A PATH model is a direct translation of a simple features-alike object to normal form. This is four tables with the three kinds of entities, "objects" (or "features"), "paths" (or "parts") and "vertices", and a table to link the one-to-many relation between paths and vertices.

Usage

PATH(x, ...)

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

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

## Default S3 method:
PATH(x, ...)

Arguments

x input model

... arguments passed to methods

Details

In a data set with no parts touching their neighbours, the only normalization of the vertices will be the removal of the duplicated closing coordinate on any polygon ring, and on any self-intersecting case within a single path.

PATH()$path should always have columns object_ path_ subobject ncoords_

Value

a PATH model, with tables 'object', 'path', 'path_link_vertex' and 'vertex'

See Also

sc_path, sc_coord
Description

Structural form requires only tables 'object' and 'vertex'.
Minimal columns is x,y but can be grouped by path_ for separate paths, then subobject_ and object_ for full polygon support.

Usage

PATH0(x, ...)

## Default S3 method:
PATH0(x, ...)

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

PATH0_from_df(
  x,
  ..., 
  path_ = "path_", 
  object_ = "object_", 
  subobject_ = "subobject_", 
  x_ = "x", 
  y_ = "y"
)

Arguments

x  data frame with at least x, y columns
...
path_ path identifier, these should identify individual paths
object_ object identifier (like group in ggplot)
subobject_ subobject identifier (like polygon_id with multipolygons in sfheaders)
x_ optional name for x column (assumed to be x)
y_ optional name for x column (assumed to be y)

Details

This function exists as a special-case for non-format input for PATH0(). It’s expected there are columns x, y, and optionally object_, subobject_, and path_. These correspond to names in sfheaders, multipolygon_id, polygon_id, and linestring_id. (subobject is optional if not multipolygon).
Value

PATH0 model with tables 'object' and 'vertex'

Examples

(p <- PATH0(minimal_mesh))

p$object$topology
PATH0_from_df(data.frame(x = runif(10), y = runif(10)))

plot.SC

Plot silicate

Description

Basic edge plot, all the standard base graphics facilities for line segments are available.

Usage

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

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

Arguments

x sc object

... arguments passed to lower level plotting functions

add if TRUE add to current plot

Details

The 'col' argument is passed directly to segments() or polypath() as needed, in the usual one-to-one or recycling way.

Graphical parameters are not able to be passed to the initial plot setup, but a plot can be set up and then added to with this method.
**polymesh**  
*Polygonal mesh*

**Description**

A simple set of sf neighbouring polygons, with redundant vertices created from polygonizing a raster.

**Examples**

```r
carc <- ARC(polymesh)  
plot(carc)  
scc <- SC(polymesh)  
plot(scc)
```

**print.sc**  
*Methods for silicate*

**Description**

Print a silicate model.

**Usage**

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

**Arguments**

- `x`  
  object inheriting from 'sc' class

- `...`  
  ignore currently

**Details**

Simple summary of type and content of a silicate model.

**Examples**

```r
print(TRI(minimal_mesh))  
print(SC(minimal_mesh))  
print(PATH(minimal_mesh))  
print(SC(TRI(minimal_mesh)))  
print(ARC(minimal_mesh))  
print(SC0(minimal_mesh))
```
Description
Routing data set stolen from stplanr see data-raw/routes.R

SC
The universal model

Description
The universal model SC is coordinates and binary relations between pairs of coordinates. This is purely an edge (or segment) model, with all higher level structures recorded as groupings of edges.

Usage
SC(x, ...)

## Default S3 method:
SC(x, ...)

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

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

Arguments
x input model
...
arguments passed to methods

Value
SC model with tables 'object', 'object_link_edge', 'edge', and 'vertex'

Examples
## we can produce a high quality triangulation from a low quality one
## see how the TRI edges are maintained (we can't yet filter out holes from DEL)
tri <- TRI(minimal_mesh)
plot(tri)
plot(SC(tri))
**SC0**

*Pure edge model, structural form*

**Description**

SC0 is the simplest and most general of all silicate models. Composed of an object and vertex table linked by nested vertex-index pairs.

**Usage**

```r
SC0(x, ...)
```

## Default S3 method:

```r
SC0(x, ...)
```

## S3 method for class 'pslg'

```r
SC0(x, ...)
```

**Arguments**

- `x`: an object understood by silicate
- `...`: reserved for methods

**Value**

SC0 model with tables 'object' and 'vertex'

**Examples**

```r
SC0(minimal_mesh)
```

**sc_arc**

*Arc-node topology.*

**Description**

Return a label and vertex count of each arc.

**Usage**

```r
sc_arc(x, ...)
```

## Default S3 method:

```r
sc_arc(x, ...)
```

## S3 method for class 'ARC'

```r
sc_arc(x, ...)
```
Arguments

x  input object
...
arguments for methods

Details

Arcs are unbranched paths within the line segment graph. Nodes are the vertices where three or more arcs meet.

As with the PATH and SC models the arc id values will only be relevant when those entities are identified and labelled. Running sc_arc on a simple features model (for example) will identify them and return a summary, but without having any record of what they refer to. Use ARC(x) first to work with models that don’t included labels.

Value

a data frame with only the identities of the shared boundaries (arcs)

Examples

sc_arc(minimal_mesh)
ARC(minimal_mesh)[[“arc”]]
arc <- ARC(minimal_mesh)
plot(arc)
points(arc$vertex[match(sc_node(arc)$vertex_, arc$vertex$vertex_), c(“x_”, “y_”)])

arc <- ARC(polymesh)
plot(arc)
title(“arcs and nodes”)
points(arc$vertex[match(sc_node(arc)$vertex_, arc$vertex$vertex_), c(“x_”, “y_”)])

sc_colours

Silicate colours

Description

Simple set of colours for discrete palette.

Usage

sc_colours(x = 16, ... , viridis = FALSE)

Arguments

x  number of colours to generate
...
currently ignored
viridis  use viridis, TRUE or FALSE
Value

vector of colours

Examples

sc_colours(10)

Description

Collect all instances of all coordinates in one table. This complementary to the sc_path of an object, since the number of coordinates per path gives a structural mapping into this set.

Collect all coordinates in one table, the path_link_vertex table has the information about the original grouping.

Usage

sc_coord(x, ...)

## S3 method for class 'list'
sc_coord(x, ...)

## Default S3 method:
sc_coord(x, ...)

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

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

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

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

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

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

## S3 method for class 'SC'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'sf'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'sfc'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'pslg'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'MULTIPOLYGON'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'POLYGON'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'MULTILINESTRING'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'LINESTRING'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'MULTIPOINT'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'POINT'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'Spatial'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'Polygons'
\texttt{sc\_coord(x, \ldots)}

## S3 method for class 'Lines'
\texttt{sc\_coord(x, \ldots)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{2cm} input model
  \item \texttt{\ldots} \hspace{2cm} arguments passed to methods
\end{itemize}

\textbf{Value}

data frame of all the coordinates in the order they occur
sc_edge

See Also

sc_path for the central part of the model, sc_object for the features, and PATH for the full model.

Examples

sc_coord(minimal_mesh)
sc_coord(SC(minimal_mesh))

sc_edge

Edges.

Description

Simple binary relationships, a primitive composed of two vertices.

Usage

sc_edge(x, ...)

## Default S3 method:
sc_edge(x, ...)

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

sc_start(x, ...)

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

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

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

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

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

## S3 method for class 'PATH0'
sc_end(x, ...)
sc_node

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

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

sc_end(x, ...)

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

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

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

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

### Arguments

- **x**: input object
- **...**: arguments for methods

### Details

Edges are unique, undirected line segments. Compare to sc_segment which refers to all instances of edges.

sc_start and sc_end are convenience functions that provide the obvious start and end coordinates by joining on the appropriate edge vertex label, .vx0 or .vx1. Currently this returns the ordered segments, along with their unique (unordered) edge_. as well as unique segment_ labels.

### Value

data frame of edge identity, or start/end coordinates

---

**sc_node**

*Nodes for arc-node topology.*

### Description

Nodes are the vertices in the graph that are shared by "arcs".
Usage

sc_node(x, ...)

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

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

## Default S3 method:
sc_node(x, ...)

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

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

Arguments

x input object
...
arguments for methods

Value
data frame of the nodes

Examples

sc_node(ARC(minimal_mesh))
sc <- SC(routes)
library(dplyr)
plot(sc)
sc_node(sc) %>% inner_join(sc$vertex) %>% select(x_, y_) %>% points()

sc_object

Objects, features

Description

The objects are the front end entities, the usual "GIS contract" objects, or features.
The objects are the front end entities, the usual "GIS contract" objects, the features.
Usage

```r
sc_object(x, 

## Default S3 method:
sc_object(x, 

## S3 method for class 'sf'
sc_object(x, 

## S3 method for class 'sfc'
sc_object(x, 

## S3 method for class 'TRI'
sc_object(x, 
```

Arguments

- `x` input object
- `...` arguments passed to methods

Value
data frame of the object values

See Also

`sc_coord` for the coordinates part of the model, `sc_path` for the central part of the model, and `PATH` for the full model.

Examples

```r
sc_object(minimal_mesh)
sc_object(SC0(minimal_mesh))
```

---

### sc_path

**Path decomposition**

**Description**

Start in the middle, and build the 'path-link-vertex' table.

Paths.
Usage

sc_path(x, ...)

## S3 method for class 'list'
sc_path(x, ids = NULL, ...)

## Default S3 method:
sc_path(x, ...)

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

## S3 method for class 'sfc_TIN'
sc_path(x, ...)

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

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

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

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

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

## S3 method for class 'sf'
sc_path(x, ids = NULL, ...)

## S3 method for class 'sfc'
sc_path(x, ids = NULL, ...)

## S3 method for class 'MULTIPOLYGON'
sc_path(x, ...)

## S3 method for class 'POLYGON'
sc_path(x, ...)

## S3 method for class 'LINESTRING'
sc_path(x, ...)

## S3 method for class 'MULTILINESTRING'
sc_path(x, ...)
sc_path

```r
## S3 method for class 'POINT'
sc_path(x, ...) # S3 method for class 'MULTIPOINT'
sc_path(x, ...) # S3 method for class 'GEOMETRYCOLLECTION'
sc_path(x, ...) # S3 method for class 'Spatial'
sc_path(x, ids = NULL, ...)
```

**Arguments**

- `x`: input object
- `...`: arguments passed to methods
- `ids`: object id, one for each object in the sfc

**Details**

Paths have properties of their type, their number of vertices, their geometric dimension and which object they occur in.

**Value**

data frame of path identity and properties

**See Also**

`sc_coord` for the coordinates part of the model, `sc_object` for the features, and `PATH` for the full model.

**Examples**

```r
sc_path(minimal_mesh)
sfzoo$preliminary_mesh
sc_path(sfzoo$multipolygon)
sfzoo$polyline
sc_path(sfzoo$linestring)
sfzoo$multilinestring
sc_path(sfzoo$point)
sfzoo$multipoint
sc_path(sfzoo$multipoint)
```
Given a ‘PATH’ model decompose to 1-dimensional primitives (or 0-dimensional).

**Usage**

```r
sc_segment(x, ...)
```

## Default S3 method:
```r
sc_segment(x, ...)
```

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

**Arguments**

- `x` input object
- `...` arguments passed to methods

**Value**

data frame of the segments, each occurrence of an edge and its order

**Examples**

```r
sc_segment(SC(minimal_mesh))
```

---

**sc_uid**

Unique labels

**Description**

Find unique labels for entities, or create them if not present.

**Usage**

```r
sc_uid(x, ..., uid_nchar = NULL)
```
Arguments

- **x**: number of unique IDs to generate
- **...**: reserved for future use
- **uid_nchar**: number of raw characters to paste as a uuid, default is 6 (only if silicate.uid.type is "uuid", see Details)

Details

If 'integers' default we generate sequential integers, it’s assumed that all IDs are created at one time, we are not adding to an existing set. Code that adds IDs should find the largest existing ID and offset these by that value.

Using 'silicate.uid.type="uuid"' is the default. Using 'silicate.uid.type="integer"' is considered experimental. By default UIDs are a mix of letters, LETTERS and digits of length `getOption("silicate.uid.size")` which defaults to 6.

See ids package for `random_id` used if option 'silicate.uid.type="uuid"'.

Value

vector of unique id values for elements in the input

Examples

```
sc_uid(data.frame(1:10))
```

## Description

Extract unique vertices

## Usage

```
sc_vertex(x, ...)
```

## Default S3 method:
```
sc_vertex(x, ...)
```

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

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

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

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

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

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

## S3 method for class 'pslg'
sc_vertex(x, ...)

### Arguments

- **x** model
- **...** passed to methods

### Value

data frame of only the unique coordinates

### Examples

```r
sc_vertex(minimal_mesh)
sfzoo(sc_vertex(SC0(minimal_mesh)))
```

---

**sfzoo**

*Simple features zoo.*

---

### Description

Basic examples of each type of simple feature geometry. *sfzoo* is a list with each of *point*, *multipoint*, *linestring*, *multilinestring*, *polygon* and *multipolygon*. *sfgc* is a *GEOMETRYCOLLECTION* of all the types in *sfzoo*.

### Examples

```r
lapply(sfzoo, sc_coord)
lapply(sfzoo, sc_path)
```

## unsure how useful this is ...
```
sfzoo(sc_path(sfgc))
```
Description

Decomposes spatial data (of various formats) into simpler forms, including paths, triangles or segments. A development tool for exploring the underlying structures of spatial data, and for converting it to something else. The models `PATH()`, `TRI()`, `SC()` and `ARC()` provide relational tables of all underlying entities, and more specialist versions `PATH0()`, `TRI0()` and `SC0()` provide more efficient topological representations of polygons or lines.

TRI

*TRI model, triangulations*

Description

TRI creates a constrained triangulation using 'ear-cutting', or 'ear-clipping' of polygons. The model is a 'relational' form in that the underlying tables are linked implicitly by unique identifiers. Ear-cutting is inherently path-based, so this model is only available for path-based structures, like simple features, `PATH()`, `PATH0()` and `ARC()`.

Usage

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

Arguments

- `x`: object understood by silicate (sf, sp, a silicate model, etc.)
- `...`: current unused
- `add`: logical create new plot (default), or add to existing

Value

TRI model with tables 'object', 'triangle', 'vertex'

Examples

```r
tri <- TRI(minimal_mesh)
plot(tri)
```
TRIO

TRIO model, structural triangulations

Description

TRIO creates a constrained triangulation using 'ear-cutting', or 'ear-clipping' of polygons. It is a 'structural' form, a denser storage mode than 'relational' as used by TRI(), we trade some generality for size and speed.

Usage

TRIO(x, ...)

## Default S3 method:
TRIO(x, ...)

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

## S3 method for class 'TRIO'
TRIO(x, ...)

## S3 method for class 'sfc_TIN'
TRIO(x, ...)

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

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

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

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

## S3 method for class 'sfc_GEOMETRYCOLLECTION'
TRIO(x, ...)

Arguments

x object understood by silicate (sf, sp, a silicate model, etc.)

... currently unused
TRI0

Details

TRI0 is suitable for simple conversion to other mesh forms. See the examples for plotting and (in commented code) conversion to rgl’s ’mesh3d’.

’Structural’ means that the model does not store relational IDs between tables, the vertex indexing is stored as a nested list of data frames in the ’object’ table. Unlike TRI() we cannot arbitrarily rearrange the order or remove content of the underlying tables, without updating the vertex indexes stored for each object.

Ear-cutting is inherently path-based, so this model is only available for path-based structures, like simple features, PATH(), PATH0() and ARC().

There is limited support for simple features GEOMETRYCOLLECTION, in short if the GC is composed purely of POLYGON type with 4 coordinates each this is assumed to be a collection of triangles and is converted directly without any triangulation performed. GEOMETRYCOLLECTION of any other form is not supported.

Value

TRI0 model with tables ’object’, ’vertex’

See Also

TRI

Examples

tri <- TRI0(minimal_mesh)
print(tri)
plot(tri)

# obtain the vertices and indices in raw form

## idx is the triplets of row numbers in tri$vertex
idx <- do.call(rbind, sc_object(tri)$topology_)
idx <- as.matrix(idx[c(".vx0", ".vx1", ".vx2")])

## vert is the vertices x_, y_, ... 
vert <- as.matrix(sc_vertex(tri))

## now we can plot with generic tools
plot(vert)
polygon(vert[t(cbind(idx, NA)), ])

## or create other structures like rgl’s mesh3d
## (see hypertidy/anglr for in-dev helpers)
## rgl::tmesh3d(t(cbind(vert, 1, 1)), t(idx),
## material = list(color = c("firebrick", "black", "grey", "blue")),
## meshColor = "faces")
Area of triangles

Description

Input is x,y matrix in triplets, with 3 rows per triangle vertex.

Usage

tri_area(x)

Arguments

x  
matrix of triangle coordinates

Value

numeric, area of triangles

Examples

pts <- structure(c(5L, 3L, 1L, 4L, 4L, 8L, 6L, 9L), .Dim = c(4L, 2L))
tri <- c(2, 1, 3, 2, 4, 1)
(a <- tri_area(pts[tri, ]))
plot(pts)
polygon(pts[head(as.vector(rbind(matrix(tri, nrow = 3), NA)), -1), ])
text(tapply(pts[tri, 1], rep(1:2, each = 3), mean),
     tapply(pts[tri, 2], rep(1:2, each = 3), mean), labels = sprintf("area: %0.1f", a))
Index

ARC, 3
ARC(), 25, 27
dplyr-methods, 4
ear_gc, 4
filter (dplyr-methods), 4
flight_tracks, 5
inlandwaters, 5
minimal_mesh, 6
mmesh, 6
PATH, 7
PATH(), 25, 27
PATH0, 8
PATH0(), 8, 25, 27
PATH0_from_df (PATH0), 8
plot.SC, 9
plot.SC0 (plot.SC), 9
plot.TRI (TRI), 25
polymesh, 10
polypath(), 9
print.sc, 10
routes, 11
SC, 11
SC(), 4, 25
SC0, 12
SC0(), 25
sc_arc, 12
sc_colours, 13
sc_coord, 14
sc_edge, 16
sc_end (sc_edge), 16
sc_node, 17
sc_object, 18
sc_path, 19
sc_segment, 22
sc_start (sc_edge), 16
sc_uid, 22
sc_vertex, 23
segments(), 9
sfgc (sfzoo), 24
sfzoo, 24
silicate, 25
silicate-package (silicate), 25
TRI, 25
TRI(), 25–27
TRI0, 26
TRI0(), 25
tri_area, 28