Package: centerline (via r-universe)

October 23, 2024

Title Extract Centerline from Closed Polygons

Version 0.2

Maintainer Anatoly Tsyplenkov <atsyplenkov@fastmail.com>

Description Generates skeletons of closed 2D polygons using Voronoi diagrams. It provides methods for 'sf', 'terra', and 'geos' objects to compute polygon centerlines based on the generated skeletons. Voronoi, G. (1908) <doi:10.1515/crll.1908.134.198>.

License MIT + file LICENSE

URL https://centerline.anatolii.nz, https://github.com/atsyplenkov/centerline

BugReports https://github.com/atsyplenkov/centerline/issues

Imports wk (>= 0.9), sf (>= 1.0), geos (>= 0.2.4), sfnetworks (>= 0.6), checkmate

Suggests smoothr (>= 1.0.0), testthat (>= 3.0.0), geomtextpath (>= 0.1.0), terra (>= 1.7), igraph (>= 2.0.0), ggplot2 (>= 3.1.0), raybevel (>= 0.1.3)

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Language en-US

Config/Needs/website mapgl, rmapshaper, bench, htmlwidgets

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

RemoteUrl https://github.com/atsyplenkov/centerline

RemoteRef HEAD

RemoteSha 2ef68a114fbfff78d21950d0f5ca7a116dd28670

Contents

`cnt_path` *Find the shortest path between start and end points within a polygon*

Description

Find the shortest path between start and end points within a polygon

Usage

```
cnt_path(skeleton, start_point, end_point)
```

Arguments

`skeleton`	an output from `cnt_skeleton()` function
`start_point`	one or more starting points. It should be of the same class as the `skeleton` parameter
`end_point`	one ending point of the same class as `skeleton` and `start_point` parameters.

Details

The following function uses the `sfnetworks::st_network_paths()` approach to connect `start_point` with `end_point` by using the `skeleton` of a closed polygon as potential routes.

It is important to note that multiple starting points are permissible, but there can only be **one ending point**. Should there be two or more ending points, the algorithm will return an error.

Neither starting nor ending points are required to be located on the edges of a polygon (i.e., snapped to the boundary); they can be positioned wherever possible inside the polygon.

The algorithm identifies the closest nodes of the polygon's skeleton to the starting and ending points and then connects them using the shortest path possible along the skeleton. Therefore, if more precise placement of start and end points is necessary, consider executing the `cnt_skeleton()` function with the `keep = 1` option. In doing so, the resulting skeleton may be more detailed, increasing the likelihood that the starting and ending points are already situated on the skeleton paths.

Value

a list of `sf`, `sfc`, `SpatVector` or `geos_geometry` class objects of a `LINESTRING` geometry

Examples

```
library(sf)
library(geos)
# Load Polygon and points data
polygon <-
  sf::st_read(
    system.file("extdata/example.gpkg", package = "centerline"),
    layer = "polygon",
    quiet = TRUE
  ) |>
  geos::as_geos_geometry()

points <-
  sf::st_read(
    system.file("extdata/example.gpkg", package = "centerline"),
    layer = "polygon_points",
    quiet = TRUE
  ) |>
  geos::as_geos_geometry()

# Find polygon's skeleton
pol_skeleton <- cnt_skeleton(polygon)

# Connect points
pol_path <-
  cnt_path(
    skeleton = pol_skeleton,
    start_point = points[2],
    end_point = points[1]
  )

# Plot
plot(polygon)
plot(pol_skeleton, col = "blue", add = TRUE)
plot(points[1:2], col = "red", add = TRUE)
plot(pol_path, lwd = 3, add = TRUE)
```

`cnt_path_guess` *Guess polygon's centerline*

Description

This function, as follows from the title, tries to guess the polygon centerline by connecting the most distant points from each other. First, it finds the point most distant from the polygon's centroid, then it searches for a second point, which is most distant from the first. The line connecting these two points will be the desired centerline.

Usage

```
cnt_path_guess(input, skeleton = NULL, return_geos = FALSE, ...)
```

Arguments

input	sf, sfc or SpatVector polygons object
skeleton	NULL (default) or cnt_skeleton() output. If NULL then polygon's skeleton would be estimated in the background using specified parameters (see inherit params below).
return_geos	FALSE (default). A logical flag that controls whether the geos_geometry should be returned.
...	Arguments passed on to cnt_skeleton
	keep numeric, proportion of points to retain (0.05-5.0; default 0.5). See Details.
	method character, either "voronoi" (default) or "straight", or just the first letter "v" or "s". See Details.

Value

An sf, sfc or SpatVector class object of a LINESTRING geometry

Examples

```
library(sf)
library(geos)
lake <-
  sf::st_read(
    system.file("extdata/example.gpkg", package = "centerline"),
    layer = "lake",
    quiet = TRUE
  ) |>
  geos::as_geos_geometry()
# Find lake's centerline
lake_centerline <- cnt_path_guess(input = lake, keep = 1)
# Plot
plot(lake)
plot(lake_centerline, col = "firebrick", lwd = 2, add = TRUE)
```

---

cnt_skeleton *Create a skeleton of a closed polygon object*

Description

This function generates skeletons of closed polygon objects.

Usage

```
cnt_skeleton(input, keep = 0.5, method = "voronoi")
```

Arguments

input	sf, sfc, SpatVector, or geos_geometry polygons object
keep	numeric, proportion of points to retain (0.05-5.0; default 0.5). See Details.
method	character, either "voronoi" (default) or "straight", or just the first letter "v" or "s". See Details.

Details

Polygon simplification/densification:

- If keep = 1, no transformation will occur. The function will use the original geometry to find the skeleton.
- If the keep parameter is below 1, then the geos::geos_simplify() function will be used. So the original input geometry would be simplified, and the resulting skeleton will be cleaner but maybe more edgy. The current realisation of simplification is similar (*but not identical*) to rmapshaper::ms_simplify() one with Douglas-Peuker algorithm. However, due to geos superpower, it performs several times faster. If you find that the built-in simplification algorithm performs poorly, try rmapshaper::ms_simplify() first and then find the polygon skeleton with keep = 1, i.e. cnt_skeleton(rmapshaper::ms_simplify(polygon_sf), keep = 1)
- If the keep is above 1, then the densification algorithm is applied using the geos::geos_densify() function. This may produce a very large object if keep is set more than 2. However, the resulting skeleton would potentially be more accurate.

Skeleton method:

- If method = "voronoi" (default), the skeleton will be generated using the geos::geos_voronoi_edges() function. This is application of the Voronoi diagram algorithm (Voronoi, 1908). A Voronoi diagram partitions space into regions based on the distance to the polygon's vertices. The edges of these cells form a network of lines (skeletons) that represent the structure of the polygon while preserving its overall shape.
- If method = "straight", the skeleton will be generated using the raybevel::skeletonize() function. See https://www.tylermw.com/posts/rayverse/raybevel-introduction.html

Value

a sf, sfc, SpatVector or geos_geometry class object of a MULTILINESTRING geometry

References

Voronoi, G. (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Journal für die reine und angewandte Mathematik, 134, 198-287. doi:10.1515/crll.1908.134.198

Examples

```
library(sf)

polygon <-
  sf::st_read(system.file("extdata/example.gpkg", package = "centerline"),
    layer = "polygon",
```

```
    quiet = TRUE
  )

plot(polygon)

pol_skeleton <- cnt_skeleton(polygon)

plot(pol_skeleton)
```

---

geom_cnt *Plot centerline with ggplot2*

Description

Binding for ggplot2::geom_sf(), therefore it supports only sf objects.

Usage

```
geom_cnt(
  mapping = ggplot2::aes(),
  data = NULL,
  stat = "sf",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  keep = 0.5,
  method = c("voronoi", "straight"),
  simplify = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used the override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes.

You can also set this to one of "polygon", "line", and "point" to override the default legend.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`keep` numeric, proportion of points to retain (0.05-5.0; default 0.5). See Details.

`method` character, either `"voronoi"` (default) or `"straight"`, or just the first letter `"v"` or `"s"`. See Details.

`simplify` logical, if `TRUE` (default) then the centerline will be smoothed with `smoothr::smooth_ksmooth()`

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Value

A `Layer` ggproto object that can be added to a plot.

CRS

`coord_sf()` ensures that all layers use a common CRS. You can either specify it using the `crs` param, or `coord_sf()` will take it from the first layer that defines a CRS.

Combining sf layers and regular geoms

Most regular geoms, such as `geom_point()`, `geom_path()`, `geom_text()`, `geom_polygon()` etc. will work fine with `coord_sf()`. However when using these geoms, two problems arise. First, what CRS should be used for the x and y coordinates used by these non-sf geoms? The CRS applied to non-sf geoms is set by the `default_crs` parameter, and it defaults to `NULL`, which means positions for non-sf geoms are interpreted as projected coordinates in the coordinate system set by the `crs` parameter. This setting allows you complete control over where exactly items are placed on the plot canvas, but it may require some understanding of how projections work and how to generate data in projected coordinates. As an alternative, you can set `default_crs = sf::st_crs(4326)`, the World Geodetic System 1984 (WGS84). This means that x and y positions are interpreted as longitude and latitude, respectively. You can also specify any other valid CRS as the default CRS for non-sf geoms.

The second problem that arises for non-sf geoms is how straight lines should be interpreted in projected space when `default_crs` is not set to `NULL`. The approach `coord_sf()` takes is to break straight lines into small pieces (i.e., segmentize them) and then transform the pieces into projected coordinates. For the default setting where x and y are interpreted as longitude and latitude, this approach means that horizontal lines follow the parallels and vertical lines follow the meridians. If you need a different approach to handling straight lines, then you should manually segmentize and project coordinates and generate the plot in projected coordinates.

See Also

`geom_cnt_text()`, `geom_cnt_label()`, `ggplot2::geom_sf()`

Examples

```
library(sf)
```

```
library(ggplot2)

lake <-
  sf::st_read(
    system.file("extdata/example.gpkg", package = "centerline"),
    layer = "lake",
    quiet = TRUE
  )

ggplot() +
  geom_sf(data = lake) +
  geom_cnt(
    data = lake,
    keep = 1,
    simplify = TRUE
  ) +
  theme_void()
```

---

geom_cnt_text *Plot label or text on centerline with ggplot2*

Description

Binding for geomtextpath::geom_textsf() and geomtextpath::geom_labelsf()

Usage

```
geom_cnt_text(
  mapping = ggplot2::aes(),
  data = NULL,
  stat = "sf",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  keep = 0.5,
  method = c("voronoi", "straight"),
  simplify = TRUE,
  ...
)

geom_cnt_label(
  mapping = ggplot2::aes(),
  data = NULL,
  stat = "sf",
  position = "identity",
  na.rm = FALSE,
```

```
  show.legend = NA,
  inherit.aes = TRUE,
  keep = 0.5,
  method = c("voronoi", "straight"),
  simplify = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer, either as a ggproto Geom subclass or as a string naming the stat stripped of the stat_ prefix (e.g. "count" rather than "stat_count")
position	Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. You can also set this to one of "polygon", "line", and "point" to override the default legend.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().
keep	numeric, proportion of points to retain (0.05-5.0; default 0.5). See Details.
method	character, either "voronoi" (default) or "straight", or just the first letter "v" or "s". See Details.
simplify	logical, if TRUE (default) then the centerline will be smoothed with smoothr::smooth_ksmooth()
...	Arguments passed on to geom_textpath, geom_labelpath text_only A logical(1) indicating whether the path part should be plotted along with the text (FALSE, the default). If TRUE, any parameters or aesthetics relating to the drawing of the path will be ignored.

gap A `logical(1)` which if `TRUE`, breaks the path into two sections with a gap on either side of the label. If `FALSE`, the path is plotted as a whole. Alternatively, if `NA`, the path will be broken if the string has a `vjust` between 0 and 1, and not otherwise. The default for the label variant is `FALSE` and for the text variant is `NA`.

upright A `logical(1)` which if `TRUE` (default), inverts any text where the majority of letters would upside down along the path, to improve legibility. If `FALSE`, the path decides the orientation of text.

halign A `character(1)` describing how multi-line text should be justified. Can either be `"center"` (default), `"left"` or `"right"`.

offset A `unit` object of length 1 to determine the offset of the text from the path. If this is `NULL` (default), the `vjust` parameter decides the offset. If not `NULL`, the `offset` argument overrules the `vjust` setting.

parse A `logical(1)` which if `TRUE`, will coerce the labels into expressions, allowing for plotmath syntax to be used.

straight A `logical(1)` which if `TRUE`, keeps the letters of a label on a straight baseline and if `FALSE` (default), lets individual letters follow the curve. This might be helpful for noisy paths.

padding A `unit` object of length 1 to determine the padding between the text and the path when the `gap` parameter trims the path.

text_smoothing a `numeric(1)` value between 0 and 100 that smooths the text without affecting the line portion of the geom. The default value of `0` means no smoothing is applied.

rich A `logical(1)` whether to interpret the text as html/markdown formatted rich text. Default: `FALSE`. See also the rich text section of the details in `geom_textpath()`.

remove_long if TRUE, labels that are longer than their associated path will be removed.

label.padding Amount of padding around label. Defaults to 0.25 lines.

label.r Radius of rounded corners. Defaults to 0.15 lines.

Details

Aesthetics:

`geom_cnt_text()` understands the following aesthetics:

- `x`
- `y`
- `label`
- `alpha`
- `angle`
- `colour`
- `family`
- `fontface`
- `group`
- `hjust`

- linecolour
- lineheight
- linetype
- linewidth
- size
- spacing
- textcolour
- vjust

In addition to aforementioned aesthetics, geom_cnt_label() also understands:

- boxcolour
- boxlinetype
- boxlinewidth
- fill

See Also

geom_cnt(), geomtextpath::geom_textsf(), geomtextpath::geom_labelsf(), ggplot2::geom_sf()

Examples

```
library(sf)
library(ggplot2)

lake <-
  sf::st_read(
    system.file("extdata/example.gpkg", package = "centerline"),
    layer = "lake",
    quiet = TRUE
  )

# Plot centerline and lake name as text
ggplot() +
  geom_sf(data = lake) +
  geom_cnt_text(
    data = lake,
    aes(label = "Lake Ohau"),
    size = 8,
    simplify = TRUE
  ) +
theme_void()

# Plot lake name as label
ggplot() +
  geom_sf(data = lake) +
  geom_cnt_label(
    data = lake,
    aes(label = "Lake Ohau"),
    linecolor = NA, # disable line drawing
    size = 10,
```

```
    method = "s",
    simplify = TRUE
  ) +
theme_void()
```

Index