# Package 'bigcurve'

November 6, 2026

**Title** Adaptive Densification of Lines for Map Projections

**Version** 0.6.0

**Description** Densify lines adaptively, as required by the curvature a map projection imposes on them. Edges are read as great circle arcs, and vertices are inserted along them only where the projected curve deviates from a straight chord by more than a tolerance measured on the map, following the adaptive resampling method of the 'd3-geo' 'JavaScript' library described in Bostock (2016) <https://github.com/d3/d3-geo>. Projections are computed with the 'PROJ' coordinate transformation library (<https://proj.org/>), so any coordinate reference system it supports can be a target. Inputs can be coordinate matrices, segment meshes or vertex pool topology from 'wkpool', and edges shared by neighbouring polygons receive identical new vertices.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Imports** stats

**URL** https://github.com/hypertidy/bigcurve, https://hypertidy.github.io/bigcurve/

**BugReports** https://github.com/hypertidy/bigcurve/issues

**Suggests** geographiclib, knitr, maps, PROJ, rmarkdown, s2, testthat (>= 3.0.0), textures, wk, wkpool (>= 0.4.0)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**LinkingTo** cpp11

**SystemRequirements** PROJ (>= 8.0.0): libproj-dev (deb), proj-devel (rpm), proj (brew)

**Config/roxygen2/version** 8.1.0

**NeedsCompilation** yes

**Author** Michael D. Sumner [aut, cre, cph],
Simon Wotherspoon [ctb] (developed the spherical track midpoint method that the great circle midpoint used here is based on),
Mike Bostock [ctb, cph] (author of the 'd3-geo' adaptive resampling method ported here; see inst/COPYRIGHTS)

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

**Repository** CRAN

**Date/Publication** 2026-11-06 12:40:09 UTC

# Contents

densify . . . . . 2
laea . . . . . 4
rproj_xy . . . . . 5
segment . . . . . 6

**Index** **7**

---

`densify` *Adaptively densify lines for a projection*

---

## Description

Insert vertices along great circle arcs exactly where a target projection bends them, so that straight chords drawn between output vertices stay within `tolerance` of the true projected curve. This is the adaptive resampling scheme of d3-geo (Bostock), generalized to paths and to segment meshes.

## Usage

```
densify(
  x,
  target,
  source = NULL,
  tolerance = NULL,
  pixels = 2048L,
  max_depth = 16L,
  ellipsoid = FALSE
)
```

## Arguments

| | |
|---|---|
| `x` | a two-column matrix of lon,lat (a path), a list of such matrices, a segment mesh (a list with `vb`, a 4 x n vertex matrix with rows x, y, z, h, and `is`, a 2 x m segment index matrix, 1-based, as used by rgl's mesh3d), or a `wkpool` vertex-pool topology from the wkpool package (any wk-handleable geometry via `wkpool::establish_topology()`); pools carrying z or m vertex values are refused, as densification cannot invent them at new vertices |

| | |
|---|---|
| target | projection for which to densify (proj string, WKT, authority code - anything PROJ accepts) |
| source | coordinate system of the input. If NULL (the default) it is taken from a wkpool input's own crs when it has one, and otherwise is 'EPSG:4326' (longitude, latitude); authority codes are axis-normalized so 'EPSG:4326' input is still given as lon,lat |
| tolerance | maximum allowed deviation, in units of target (usually metres); if NULL (the default) it is derived from the larger side of the projected extent of the input as max(diff(range))/pixels |
| pixels | notional output width/height used to derive the default tolerance, i.e. "keep error under about one part in pixels of the plot"; ignored when tolerance is supplied |
| max_depth | maximum bisection depth per input segment (each level at most doubles the vertex count of a segment) |
| ellipsoid | if TRUE, new vertices are placed on the WGS84 geodesic instead of the great circle, using the 'geographiclib' package (which must be installed). The sphere still decides how many vertices each edge gets and at what fraction along it; each one is then moved to the same fraction of the geodesic between the same two endpoints. Input vertices are unchanged, shared edges still get identical vertices, and the input must be longitude, latitude. On long edges the two differ by tens of kilometres (about 23 km at most from Hobart to London) |

## Details

The error metric is evaluated in the plane of the target projection: the great circle midpoint of each candidate segment is forward-projected and compared to the segment's planar chord (perpendicular distance, plus a parametric check that catches strong shear). Segments are split recursively until they pass or max_depth is reached. No inverse transforms are used and each point is projected exactly once.

Coordinates are returned in the coordinates of source (geographic by default): project the result yourself for plotting, e.g. with rproj_xy() for a matrix or, for wk geometry (such as wkpool::segments_to_wkb() of a refined pool), with wk::wk_transform() and a PROJ::proj_trans_create() transform.

Points that cannot be projected (for example beyond the horizon of an orthographic projection) do not error: segments touching them are left un-densified, and refinement stops at unprojectable midpoints. Clipping is out of scope here.

Inserted vertices are emitted in the longitude frame of their input edge: an edge along longitude 180 refines at 180, its mirror at -180 refines at -180 (the two seams of a global layer remain distinct), and an edge written as crossing the antimeridian yields a locally continuous chain, possibly outside [-180, 180). Input vertices are always returned with their written values, so a crossing edge's path carries one numeric 360-degree step at its far endpoint (the same point on the sphere; PROJ is indifferent). Longitudes are never canonicalized; normalize afterwards if a canonical range is needed. Edges that cross the *target* projection's own discontinuity (which depends on lon_0) cannot form a valid planar chain and should be cut beforehand; refinement of such edges is bounded, not exploded, because arcs spanning less ground than tolerance are never split.

Every edge is a great circle arc on a sphere, the same reading of an edge that 's2' uses. Data whose edges are meant as straight lines in longitude and latitude describe a different shape, and densifying them here changes it.

The source must be geographic (longitude, latitude): reading projected coordinates (eastings, northings in metres) as great circle arcs is meaningless, so input whose coordinates fall outside lon `[-360, 360]` or lat `[-90, 90]` is rejected with an error. Reproject to lon,lat first. (Treating a projected source's edges as straight lines in its own plane is a separate model, not yet implemented.)

Shared edges stay shared. An edge refines to the same vertices whichever way it is written, bit for bit, so two neighbouring polygons get identical new vertices along their common boundary (and `wkpool::merge_coincident()` fuses them). An edge that appears twice with the same vertices, as in a merged pool or a segment mesh, is refined once and its new vertices are reused.

## Value

the same kind of object as `x`, with vertices added: a matrix for a path, a list of matrices for a list, a mesh with appended vertices and refined `is` for a mesh, a `wkpool` with appended vertices (freshly minted `.vx` ids), refined directed segments, and `.feature` provenance carried onto every refined segment

## References

https://bost.ocks.org/mike/example/

## Examples

```
s <- segment(c(140, -164), c(-89, 80))
d <- densify(s, "+proj=laea +lon_0=147 +lat_0=-42")
nrow(d)
plot(rproj_xy(d, "+proj=laea +lon_0=147 +lat_0=-42"), type = "l", asp = 1)

## on the WGS84 ellipsoid rather than the sphere
if (requireNamespace("geographiclib", quietly = TRUE)) {
  e <- densify(d[c(1, nrow(d)), ], "+proj=laea +lon_0=147 +lat_0=-42",
               ellipsoid = TRUE)
  lines(rproj_xy(e, "+proj=laea +lon_0=147 +lat_0=-42"), col = "firebrick")
}
```

---

# laea *Projection string*

---

## Description

Lambert azimuthal equal area projection string, in old-style PROJ format.

## Usage

```
laea(lon_0 = 0, lat_0 = 0, x_0 = 0, y_0 = 0)
```

### Arguments

| | |
|---|---|
| lon_0 | centre longitude |
| lat_0 | centre latitude |
| x_0 | false easting (default 0 is fine) |
| y_0 | false northing (default 0 is fine) |

### Details

Note that easting/northing are poor names for false X and Y offsets, but that is what they are called in PROJ.

### Value

character string, projection

### Examples

```
laea()
laea(147, -42)
```

---

## rproj_xy *Transform coordinates*

---

### Description

Forward-transform a two-column matrix of coordinates with PROJ. Authority codes are axis-normalized, so input is always x,y (lon,lat) order and output is easting,northing - even for 'EPSG:4326'.

### Usage

```
rproj_xy(x, target, ..., source = NULL)
```

### Arguments

| | |
|---|---|
| x | two-column matrix of coordinates in source |
| target | output coordinate system (anything PROJ accepts) |
| ... | ignored |
| source | input coordinate system, default 'EPSG:4326' (axis-normalized, so input is lon,lat regardless of authority order) |

### Details

Points outside the domain of target are returned as NA rather than raising an error.

### Value

two-column matrix of transformed coordinates

## Examples

```
rproj_xy(cbind(147, -42), "+proj=laea +lon_0=147 +lat_0=-42")
```

---

`segment` *Segment*

---

## Description

A two-point path as a two-column matrix, by default a random one.

## Usage

```
segment(x = runif(2L, -180, 180), y = runif(2L, -90, 90))
```

## Arguments

| | |
|---|---|
| x | longitude (2, start point, end point) |
| y | latitude (2, same as x) |

## Value

two-column matrix

## Examples

```
segment()
segment(c(0, 147), c(0, -42))
```

# Index

densify, 2

laea, 4

rproj_xy, 5
rproj_xy(), *3*

segment, 6