Package 'mapycusmaximus'

February 6, 2026

Title Focus-Glue-Context Fisheye Transformations for Spatial Visualization

Version 1.0.7

Description Focus-glue-context (FGC) fisheye transformations to two-dimensional coordinates and spatial vector geometries. Implements a smooth radial distortion that enlarges a focal region, transitions through a glue ring, and preserves outside context. Methods build on generalized fisheye views and focus+context mapping. For more details see Furnas (1986) <doi:10.1145/22339.22342>, Furnas (2006) <doi:10.1145/1124772.1124921> and Yamamoto et al. (2009) <doi:10.1145/1653771.1653788>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports ggplot2, sf

Suggests testthat (>= 3.0.0), dplyr, knitr, rmarkdown, shiny, tidyr, purrr, ggthemes

VignetteBuilder knitr

Config/testthat/edition 3

Depends R (>= 3.6)

LazyData true

LazyDataCompression xz

URL https://alex-nguyen-vn.github.io/mapycusmaximus/

BugReports https://github.com/Alex-Nguyen-VN/mapycusmaximus/issues

NeedsCompilation no

Author Alex Nguyen [aut, cre, cph], Dianne Cook [aut] (ORCID: <https://orcid.org/0000-0002-3813-7155>), Michael Lydeamore [aut] (ORCID: <https://orcid.org/0000-0001-6515-827X>)

Maintainer Alex Nguyen <thanhcuong10091992@gmail.com>

Repository CRAN

Date/Publication 2026-02-06 09:20:02 UTC

Contents

`classify_zones` *Classify Coordinates into Focus, Glue, or Context Zones*

Description

Assigns each point to one of three zones based on its radial distance from a specified center:

- **focus**: inside the inner radius `r_in`
- **glue**: between `r_in` and `r_out`
- **context**: outside `r_out`

This is a helper for visualizing and analyzing fisheye transformations using the Focus–Glue–Context (FGC) model.

Usage

```
classify_zones(coords, cx = 0, cy = 0, r_in = 0.34, r_out = 0.5)
```

Arguments

`coords`	A numeric matrix or data frame with at least two columns representing `(x, y)` coordinates.
`cx`, `cy`	Numeric. The x and y coordinates of the fisheye center (default = 0, 0).
`r_in`	Numeric. Inner radius of the focus zone (default = 0.34).
`r_out`	Numeric. Outer radius of the glue zone (default = 0.5).

Value

A character vector of the same length as `nrow(coords)`, with values `"focus"`, `"glue"`, or `"context"`.

See Also

`fisheye_fgc()`, `plot_fisheye_fgc()`

Examples

```
# Simple example
pts <- matrix(c(0, 0, 0.2, 0.2, 0.6, 0.6), ncol = 2, byrow = TRUE)
classify_zones(pts, r_in = 0.3, r_out = 0.5)
#> "focus"   "glue"    "context"
```

---

conn_fish *Fisheye-Distorted Hospital–RACF Connections (sf)*

Description

An example LINESTRING layer showing hospital–RACF transfer routes after applying a **Focus–Glue–Context (FGC) fisheye warp**. It demonstrates how line geometries can be spatially distorted in sync with polygon layers to visualize flow patterns within the magnified focus zone.

Usage

```
conn_fish
```

Format

An sf object with:

weight Numeric, representing transfer magnitude or connection strength.

geometry LINESTRING geometries in projected CRS (EPSG:3111).

Details

Built from hospital–RACF coordinate pairs in data-raw/transfers_coded.csv using:

1. connection creation via make_connections() to form LINESTRINGs,
2. projection to VicGrid94 (EPSG:3111),
3. distance-based filtering to keep only sources within r_in = 0.34 of the focus point (cx = 145.0, cy = -37.8),
4. fisheye transformation using sf_fisheye() with r_in = 0.428, r_out = 0.429, and zoom_factor = 1.

The resulting object aligns spatially with vic_fish, allowing co-visualization of regional flow intensity within the distorted focus region.

Source

Prepared in data-raw/gen-data.R from transfers_coded.csv and the make_connections() function.

See Also

sf_fisheye(), vic_fish

Examples

```
library(sf)
plot(st_geometry(vic_fish), col = "grey95", border = "grey70")
plot(st_geometry(conn_fish), add = TRUE, col = "black", lwd = 1)
```

---

create_test_grid *Create a Regular Test Grid of Coordinates*

Description

Generates a 2D grid of equally spaced points, useful for testing fisheye transformations and other spatial warping functions.

Usage

```
create_test_grid(range = c(-1, 1), spacing = 0.1)
```

Arguments

range	Numeric vector of length 2 giving the x and y limits of the grid (default = c(-1, 1)).
spacing	Numeric. Distance between adjacent grid points along each axis (default = 0.1).

Value

A numeric matrix with two columns (x, y) containing the coordinates of the grid points.

See Also

plot_fisheye_fgc(), fisheye_fgc()

Examples

```
# Create a grid from -1 to 1 with spacing 0.25
grid <- create_test_grid(range = c(-1, 1), spacing = 0.25)
head(grid)
```

`fisheye_fgc` *Apply Focus–Glue–Context Fisheye Transformation*

Description

Transforms 2D coordinates using a **Focus–Glue–Context (FGC) fisheye transformation**. The function expands points inside a focus region, compresses points in a glue region, and leaves the surrounding context unchanged. Optionally, a rotational "revolution" can be added to the glue region to produce a swirling effect.

Usage

```
fisheye_fgc(
  coords,
  cx = 0,
  cy = 0,
  r_in = 0.34,
  r_out = 0.5,
  zoom_factor = 1.5,
  squeeze_factor = 0.3,
  method = "expand",
  revolution = 0
)
```

Arguments

`coords`	A matrix or data frame with at least two columns representing x and y coordinates.
`cx, cy`	Numeric. The x and y coordinates of the fisheye center (default = 0, 0).
`r_in`	Numeric. Radius of the focus zone (default = 0.34).
`r_out`	Numeric. Radius of the glue zone boundary (default = 0.5).
`zoom_factor`	Numeric. Expansion factor applied within the focus zone (default = 1.5).
`squeeze_factor`	Numeric in (0,1]. Compression factor applied within the glue zone (smaller values = stronger compression, default = 0.3).
`method`	Character. "expand" or "outward" (default = "expand").
`revolution`	Numeric. Optional rotation factor applied in the glue zone. Positive values rotate counter-clockwise, negative values clockwise (default = 0.0).

Details

This function operates in three radial zones around a chosen center:

- **Focus zone (r <= r_in)**: expands distances from the center using `zoom_factor`, but does not exceed the `r_in` boundary.

- **Glue zone (r_in < r <= r_out)**: compresses distances using a power-law defined by `squeeze_factor`, then remaps them to smoothly connect focus and context zones.
- **Context zone (r > r_out)**: coordinates remain unchanged.

Optionally, points in the glue zone can be rotated (`revolution`) to emphasize continuity.

Value

A numeric matrix with two columns (`x_new`, `y_new`) of transformed coordinates. Additional attributes:

- `"zones"`: character vector classifying each point as `"focus"`, `"glue"`, or `"context"`.
- `"original_radius"`: numeric vector of original radial distances.
- `"new_radius"`: numeric vector of transformed radial distances.

Examples

```
# Create a set of example coordinates
grid <- create_test_grid(range = c(-1, 1), spacing = 0.1)

# Apply FGC fisheye with expansion and compression
transformed <- fisheye_fgc(grid, r_in = 0.34, r_out = 0.5, zoom_factor = 1.3, squeeze_factor = 0.5)

# Plot original vs transformed
plot_fisheye_fgc(grid, transformed, r_in = 0.34, r_out = 0.5)
```

---

plot_fisheye_fgc *Visualize Focus–Glue–Context (FGC) Fisheye Transformation*

Description

Creates a side-by-side scatterplot comparing the **original** and **transformed** coordinates of a dataset under the Focus–Glue–Context fisheye mapping. Points are colored according to whether they fall in the *focus*, *glue*, or *context* zones, and boundary circles are drawn for clarity.

Usage

```
plot_fisheye_fgc(
  original_coords,
  transformed_coords,
  cx = 0,
  cy = 0,
  r_in = 0.34,
  r_out = 0.5
)
```

Arguments

original_coords	A matrix or data frame with at least two columns representing the original (x, y) coordinates.
transformed_coords	A matrix or data frame with the transformed (x, y) coordinates (same number of rows as original_coords).
cx, cy	Numeric. The x and y coordinates of the fisheye center (default = 0, 0).
r_in	Numeric. Radius of the inner *focus* boundary (default = 0.34).
r_out	Numeric. Radius of the outer *glue* boundary (default = 0.5).

Value

A ggplot2 object showing original vs transformed coordinates, colored by zone, with boundary circles overlaid.

See Also

create_test_grid(), fisheye_fgc()

Examples

```
library(ggplot2)

# Generate test grid and apply fisheye
grid <- create_test_grid(range = c(-1, 1), spacing = 0.1)
warped <- fisheye_fgc(grid, r_in = 0.4, r_out = 0.7)

# Visualize transformation
plot_fisheye_fgc(grid, warped, r_in = 0.4, r_out = 0.7)
```

---

sf_fisheye *Radial fisheye warp for* sf/sfc *objects (auto-CRS + flexible centers)*

Description

sf_fisheye() applies a **focus–glue–context** fisheye to vector data: it (1) ensures a sensible projected working CRS, (2) **normalizes** coordinates around a chosen center, (3) calls fisheye_fgc() to warp radii, (4) **denormalizes** back to map units, and (5) restores the original CRS. Inside the focus ring (r_in) features enlarge; across the glue ring (r_out) they transition smoothly; outside, they stay nearly unchanged.

Usage

```
sf_fisheye(
  sf_obj,
  center = NULL,
  center_crs = NULL,
  normalized_center = FALSE,
  cx = NULL,
  cy = NULL,
  r_in = 0.34,
  r_out = 0.5,
  zoom_factor = 1.5,
  squeeze_factor = 0.35,
  method = "expand",
  revolution = 0,
  target_crs = NULL,
  preserve_aspect = TRUE
)
```

Arguments

`sf_obj`	An `sf` or `sfc` object. Supports `POINT`, `LINESTRING`, `POLYGON`, and `MULTIPOLYGON`. Empty geometries are removed before processing.
`center`	Flexible center specification (see **Center selection**): • numeric length-2 pair interpreted via `center_crs` or by lon/lat heuristic, or as map units if not lon/lat; • any `sf`/`sfc` geometry, from which a centroid is derived; • normalized $[-1, 1]$ pair when `normalized_center = TRUE`.
`center_crs`	Optional CRS for a numeric `center` (e.g., `"EPSG:4326"`). Ignored if `center` is an `sf`/`sfc` object (its own CRS is used).
`normalized_center`	Logical. If `TRUE`, `center` is treated as a normalized $[-1, 1]$ coordinate around the bbox midpoint.
`cx, cy`	Optional center in **working CRS** map units (legacy path, ignored when `center` is provided).
`r_in, r_out`	Numeric radii (in **normalized units**) defining focus and glue boundaries; must satisfy `r_out > r_in`.
`zoom_factor`	Numeric (> 1 to enlarge). Focus magnification passed to `fisheye_fgc()`.
`squeeze_factor`	Numeric in [0, 1]. Glue-zone compression strength passed to `fisheye_fgc()`.
`method`	Character; name understood by `fisheye_fgc()` (default `"expand"`).
`revolution`	Numeric (radians); optional angular twist for glue zone, passed to `fisheye_fgc()`.
`target_crs`	Optional working CRS (anything accepted by `sf::st_crs()` / `sf::st_transform()`). If `NULL`, a projected CRS is auto-selected when the input is lon/lat; otherwise the input CRS is used.
`preserve_aspect`	Logical. If `TRUE` (default), use uniform scaling; if `FALSE`, scale axes independently (may stretch shapes).

Details

CRS handling. If `target_crs` is `NULL` and the input is geographic (lon/lat), a projected **working CRS** is chosen from the layer's centroid:

- Victoria, AU region (approximate 140–150°E, 40–30°S): **EPSG:7855** (GDA2020 / MGA55).
- Otherwise UTM: **EPSG:326##** (north) or **EPSG:327##** (south).

You may override with `target_crs`. The original CRS is restored on return.

Center selection. The fisheye center can be supplied in multiple ways:

- `center = c(lon, lat)`, with `center_crs = "EPSG:4326"` (recommended for WGS84) or another CRS string/object.
- `center = c(x, y)` already in **working CRS** map units (meters).
- `center` as any `sf`/`sfc` geometry (POINT/LINE/POLYGON/etc.): its **centroid of the combined geometry** is used, then transformed to the working CRS.
- `center = c(cx, cy)` as **normalized** coordinates in $[-1, 1]$ when `normalized_center = TRUE` (relative to the bbox midpoint and scale used for normalization).
- Legacy `cx, cy` (map units) are still accepted and used only when `center` is not supplied.

Normalization. Let bbox half-width/height be `sx`, `sy`. With `preserve_aspect = TRUE` (default), a uniform scale `s = max(sx, sy)` maps $(x, y) \mapsto ((x - cx)/s, (y - cy)/s)$, so `r_in`/`r_out` (e.g., 0.34/0.5) are interpreted in a unit-like space. If `preserve_aspect = FALSE`, X and Y are independently scaled by `sx` and `sy`.

Implementation notes. Geometry coordinates are transformed by `st_transform_custom()` which safely re-closes polygon rings and drops Z/M. The radial warp itself is delegated to `fisheye_fgc()` (which is not modified).

The transformation may introduce self-intersections or other topology issues due to geometric warping.

Value

An object of the same top-level class as `sf_obj` (`sf` or `sfc`), with geometry coordinates warped by the fisheye and the **original CRS** restored.

See Also

`sf::st_transform()`, `sf::st_is_longlat()`, `sf::st_crs()`, `sf::st_coordinates()`, `st_transform_custom()`, `fisheye_fgc()`

Examples

```
library(sf)

# Toy polygon in a projected CRS
poly <- st_sfc(st_polygon(list(rbind(
  c(0,0), c(1,0), c(1,1), c(0,1), c(0,0)
))), crs = 3857)
```

```
# Default center (bbox midpoint), gentle magnification
out1 <- sf_fisheye(poly, r_in = 0.3, r_out = 0.6,
                   zoom_factor = 1.5, squeeze_factor = 0.35)

# Explicit map-unit center, stronger focus
out2 <- sf_fisheye(poly, cx = 0.5, cy = 0.5,
                   r_in = 0.25, r_out = 0.55,
                   zoom_factor = 2.0, squeeze_factor = 0.25)

# Lon/lat point (auto-project to UTM/MGA), then fisheye around CBD (WGS84)
pt_ll <- st_sfc(st_point(c(144.9631, -37.8136)), crs = 4326)  # Melbourne CBD
out3  <- sf_fisheye(pt_ll, r_in = 0.2, r_out = 0.5)

# Center supplied as an sf polygon: centroid is used as the warp center
out4 <- sf_fisheye(poly, center = poly)
```

`shiny_fisheye` *Launch Interactive Fisheye Lens Explorer*

Description

Launches an interactive Shiny application for exploring Focus–Glue–Context (FGC) fisheye lens transformations on geographic data. The app provides real-time lens positioning, adjustable distortion parameters, and side-by-side comparison of transformed and original views.

The application demonstrates fisheye effects on Victoria, Australia Local Government Areas (LGAs) with a network of healthcare facilities and aged care connections. Users can drag the lens to any position and adjust parameters without server-side re-rendering for smooth, responsive interaction.

Usage

```
shiny_fisheye(...)
```

Arguments

`...`	Additional arguments passed to `shiny::runApp()`. Common options include: • `launch.browser`: Logical or function to handle app launch (default = `TRUE`) • `port`: Integer port number (default = random available port) • `host`: Character host IP address (default = `"127.0.0.1"`) • `display.mode`: Character display mode, `"auto"`, `"normal"`, or `"showcase"`

Details

Features:

- **Interactive lens dragging**: Click and drag anywhere on the map to reposition the fisheye lens in real-time

- **Parameter controls**: Adjust inner radius (focus), outer radius (glue), zoom factor, and squeeze factor
- **Facility sampling**: Randomly sample healthcare facilities and residential aged care facilities (RACFs) with adjustable sample size
- **Transfer visualization**: Toggle display of patient transfer connections between facilities
- **Side-by-side comparison**: Compare fisheye-transformed and original static views

Requirements:

The Shiny app requires the following suggested packages:

- `shiny`
- `tidyr`
- `dplyr`
- `purrr`
- `ggthemes`

If any are missing, install with:

```
install.packages(c("shiny", "tidyr", "dplyr", "purrr", "ggthemes"))
```

Implementation Notes:

The app uses client-side JavaScript for smooth lens dragging without server round-trips. Fisheye transformations match the mathematical implementation in `fisheye_fgc()` and `sf_fisheye()`, applied to polygons, lines, and points in real-time using SVG rendering.

Value

Invisible `NULL`. The function is called for its side effect of launching the Shiny application. The R session will be blocked until the app is stopped (press Escape or close the browser window).

See Also

- `fisheye_fgc()` for the core transformation function
- `sf_fisheye()` for transforming spatial geometries
- `plot_fisheye_fgc()` for static visualizations
- `shiny::runApp()` for additional launch options

Examples

```
## Not run:
# Launch the app with default settings
shiny_fisheye()

# Launch in browser on specific port
shiny_fisheye(launch.browser = TRUE, port = 8080)

# Launch in RStudio Viewer pane
shiny_fisheye(launch.browser = rstudioapi::viewer)

## End(Not run)
```

st_transform_custom *Apply a custom coordinate transform to an sf/sfc object (POINT/LINESTRING/POLYGON/MULTIPOLYGON)*

Description

st_transform_custom() walks through each geometry in an sf/sfc object, extracts its XY coordinates, applies a user-supplied transformation function to those coordinates, and rebuilds the geometry. It preserves the input CRS on the resulting sfc column. Polygon rings are re-closed after transformation so the first and last vertex match.

Usage

```
st_transform_custom(sf_obj, transform_fun, args)
```

Arguments

sf_obj	An object of class sf or sfc. Supported geometry types: POINT, LINESTRING, POLYGON, and MULTIPOLYGON.
transform_fun	A function that accepts a numeric matrix of coordinates with two columns (X, Y) and returns a transformed numeric matrix with the same number of rows and two columns. For example: function(coords, ...) cbind(f(coords[,1], ...), g(coords[,2], ...)).
args	A named list of additional arguments to pass to transform_fun. These are appended after the coords matrix via do.call(), i.e. do.call(transform_fun, c(list(coords), args)).

Details

For POLYGON/MULTIPOLYGON, the function uses the ring indices returned by sf::st_coordinates() (L1 for rings and L2 for parts) to transform each ring independently, and then ensures each ring is explicitly closed (last vertex equals first vertex).

Error handling is per-geometry: if a geometry fails to transform, a warning is emitted and an empty geometry of the same "polygonal family" is returned to keep list lengths consistent.

The function **does not** modify or interpret the CRS numerically; it simply preserves the CRS attribute on the output sfc. If your transformation assumes metres (e.g., radial warps), ensure the input is in an appropriate projected CRS before calling this function.

Value

An object of the same top-level class as sf_obj (sf or sfc), with the same column structure (if sf) and the same CRS as the input. Geometry coordinates are replaced by the coordinates returned by transform_fun.

Expected signature of `transform_fun`

```
transform_fun <- function(coords, ...) {  ## coords: n x 2 matrix (X, Y)
  ## return an n x 2 matrix with transformed (X, Y)}
```

See Also

sf::st_coordinates(), sf::st_geometry_type(), sf::st_sfc(), sf::st_crs()

Examples

```
library(sf)

# A simple coordinate transformer: scale and shift
scale_shift <- function(coords, sx = 1, sy = 1, dx = 0, dy = 0) {
 X <- coords[, 1] * sx + dx
 Y <- coords[, 2] * sy + dy
  cbind(X, Y)
}

# POINT example
pt <- st_sfc(st_point(c(0, 0)), crs = 3857)
st_transform_custom(pt, transform_fun = scale_shift,
                    args = list(sx = 2, sy = 2, dx = 1000, dy = -500))

# LINESTRING example
ln <- st_sfc(st_linestring(rbind(c(0, 0), c(1, 0), c(1, 1))), crs = 3857)
st_transform_custom(ln, transform_fun = scale_shift,
                    args = list(sx = 10, sy = 10))

# POLYGON example (unit square)
poly <- st_sfc(st_polygon(list(rbind(c(0,0), c(1,0), c(1,1),
                                     c(0,1), c(0,0)))), crs = 3857)
st_transform_custom(poly, transform_fun = scale_shift,
                    args = list(sx = 2, sy = 0.5, dx = 5))

# MULTIPOLYGON example (two disjoint squares)
mp <- st_sfc(st_multipolygon(list(
  list(rbind(c(0,0), c(1,0), c(1,1), c(0,1), c(0,0))),
  list(rbind(c(2,2), c(3,2), c(3,3), c(2,3), c(2,2)))
)), crs = 3857)
st_transform_custom(mp, transform_fun = scale_shift,
                    args = list(dx = 100, dy = 100))

# In an sf data frame
sf_df <- st_sf(id = 1:2, geometry = st_sfc(
  st_point(c(10, 10)),
  st_linestring(rbind(c(0,0), c(2,0), c(2,2)))
), crs = 3857)

st_transform_custom(sf_df, transform_fun = scale_shift,
                    args = list(sx = 3, sy = 3))
```

vic *Victoria Local Government Areas (sf)*

Description

An example polygon layer of Victoria's LGAs for demos and tests. Built from data-raw/map/LGA_POLYGON.shp, Z/M dropped, transformed to a projected CRS, simplified, validated, and reduced to LGA_NAME + geometry.

Usage

```
vic
```

Format

An sf object with:

LGA_NAME Character, LGA name (upper case).

geometry MULTIPOLYGON / POLYGON in a projected CRS.

Details

The CRS stored in the object is whatever st_crs(vic) reports at build time. In data-raw/gen-data.R we:

1. drop Z/M (st_zm()),
2. transform to a projected CRS (st_transform()),
3. simplify (st_simplify(dTolerance = 100)),
4. repair geometries (st_make_valid()),
5. upper-case names and select columns.

Source

Prepared in data-raw/gen-data.R. Update this if you include an external data source.

Examples

```
library(sf)
plot(sf::st_geometry(vic), col = "grey90", border = "grey50")
```

vic_fish *Fisheye-Distorted Victoria LGAs (sf)*

Description

An example polygon layer of Victoria's Local Government Areas (LGAs) after applying a **Focus–Glue–Context (FGC) fisheye transformation**. This dataset illustrates how local detail can be magnified around a chosen focus point while maintaining geographic context across the state.

Usage

```
vic_fish
```

Format

An sf object with:

LGA_NAME Character, name of the LGA (upper case).

geometry MULTIPOLYGON / POLYGON geometries in projected CRS (EPSG:3111).

Details

Built from the base layer vic using:

1. projection to VicGrid94 (st_transform(vic, 3111)),
2. defining a focus center near Melbourne (cx = 145.0, cy = -37.8),
3. applying sf_fisheye() with r_in = 0.34, r_out = 0.5, and zoom_factor = 1,
4. preserving topology with st_make_valid() where needed.

The result is a smoothly warped map emphasizing the metropolitan focus zone.

Source

Prepared in data-raw/gen-data.R using the original vic polygon layer.

See Also

sf_fisheye(), conn_fish

Examples

```
library(sf)
plot(st_geometry(vic_fish), col = "grey90", border = "grey50")
```

Index