

Package ‘hexify’

August 22, 2026

Title Equal-Area Hex Grids on the Snyder ISEA Icosahedron

Version 0.8.2

Description Provides functions to build and use hexagonal discrete global grids using the Icosahedral Snyder Equal Area ('ISEA') projection (Snyder 1992 <[doi:10.3138/27H7-8K88-4882-1752](https://doi.org/10.3138/27H7-8K88-4882-1752)>) and the 'H3' hierarchical hexagonal system (Uber Technologies). Implements the 'ISEA' discrete global grid system (Sahr, White and Kimerling 2003 <[doi:10.1559/152304003100011090](https://doi.org/10.1559/152304003100011090)>). Includes a fast C++ core for 'ISEA' projection and aperture quantization, an included 'H3' v4.4.1 C library for native 'H3' grid operations, and 'sf'/'terra'-compatible R wrappers for grid generation and coordinate assignment. Output is compatible with 'dggridR' for interoperability.

License MIT + file LICENSE

SystemRequirements C99

Language en-US

Encoding UTF-8

Suggests testthat (>= 3.0.0), knitr, rmarkdown, terra, raster, ggplot2, RColorBrewer, rnaturalearth, tibble, gridExtra, leaflet

VignetteBuilder knitr

LinkingTo Rcpp

Imports sf, Rcpp, methods, rlang

URL <https://gillescolling.com/hexify/>

BugReports <https://github.com/gcol33/hexify/issues>

Config/testthat/edition 3

Depends R (>= 3.5)

LazyData true

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Gilles Colling [aut, cre, cph] (ORCID: <<https://orcid.org/0000-0003-3070-6066>>),
Uber Technologies, Inc. [ctb, cph] (H3 C library, src/h3 (Apache-2.0))

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-08-22 16:00:03 UTC

Contents

hexify-package	3
as_dggrid	4
as_sf	4
as_tibble.HexData	5
cells	6
cell_area	6
cell_to_lonlat	7
cell_to_sf	8
dgearthstat	9
dggrid_is_compatible	10
dgverify	10
from_dggrid	11
get_neighbors	12
globe_centers	13
grid_clip	14
grid_global	15
grid_info	16
grid_rect	16
h3_crosswalk	17
HexData-class	18
HexGridInfo-class	19
hexify	20
hexify-conversions	22
hexify-grid	22
hexify-stats	22
hexify_build_icosah	23
hexify_compare_resolutions	24
hexify_face_centers	25
hexify_forward	25
hexify_forward_to_face	26
hexify_get_precision	27
hexify_grid	27
hexify_heatmap	29
hexify_inverse	32
hexify_projection_stats	33
hexify_roundtrip_test	34
hexify_set_precision	35
hexify_set_verbose	35
hexify_which_face	36
hexify_world	37
hex_browse	38

hex_compact	39
hex_distance	40
hex_extract	41
hex_grid	42
hex_summarize	45
hex_uncompact	46
hex_zonal	47
import_h3	48
is_hex_data	49
is_hex_grid	50
is_pentagon	50
lonlat_to_cell	51
n_cells	52
plot,HexData,missing-method	53
plot_globe	55
plot_grid	57
plot_world	58
st_crs.HexGridInfo	59
Index	60

hexify-package	hexify
----------------	--------

Description

Core icosahedron and 'Snyder' projection helpers.

Author(s)

Maintainer: Gilles Colling <gilles.colling051@gmail.com> (**ORCID**) [copyright holder]

Authors:

- Gilles Colling <gilles.colling051@gmail.com> (**ORCID**) [copyright holder]

Other contributors:

- Uber Technologies, Inc. (H3 C library, src/h3 (Apache-2.0)) [contributor, copyright holder]

See Also

Useful links:

- <https://gillescolling.com/hexify/>
- Report bugs at <https://github.com/gcol33/hexify/issues>

as_dggrid	<i>Convert hexify grid to 'dggridR'-compatible grid object</i>
-----------	--

Description

Creates a 'dggridR'-compatible grid specification from a hexify_grid object. The resulting object can be used with 'dggridR' functions that accept a dggs object.

Usage

```
as_dggrid(grid)
```

Arguments

grid	A hexify_grid object from hexify_grid()
------	---

Value

A list with 'dggridR'-compatible fields:

pole_lon_deg	Longitude of grid pole (default 11.25)
pole_lat_deg	Latitude of grid pole (default 58.282525588538995)
azimuth_deg	Grid azimuth rotation (default 0)
aperture	Grid aperture (3, 4, or 7)
res	Resolution level
topology	Grid topology ("HEXAGON")
projection	Map projection ('ISEA')
precision	Output decimal precision (default 7)

See Also

Other 'dggridR' compatibility: [dggrid_43h_sequence\(\)](#), [dggrid_is_compatible\(\)](#), [from_dggrid\(\)](#)

as_sf	<i>Convert HexData to sf Object</i>
-------	-------------------------------------

Description

Converts a HexData object to an sf spatial features object. Can create either point geometries (cell centers) or polygon geometries (cell boundaries).

Usage

```
as_sf(x, geometry = c("point", "polygon"), ...)
```

Arguments

x	A HexData object
geometry	Type of geometry: "point" (default) or "polygon"
...	Additional arguments (ignored)

Details

For point geometry, cell centers (cell_cen_lon, cell_cen_lat) are used. For polygon geometry, cell boundaries are computed using the grid specification.

Value

An sf object

Examples

```
df <- data.frame(lon = c(0, 10, 20), lat = c(45, 50, 55))
result <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000)

# Get sf points
sf_pts <- as_sf(result)

# Get sf polygons
sf_poly <- as_sf(result, geometry = "polygon")
```

as_tibble.HexData	<i>Convert HexData to tibble</i>
-------------------	----------------------------------

Description

Convert HexData to tibble

Usage

```
as_tibble.HexData(x, ...)
```

Arguments

x	A HexData object
...	Additional arguments (ignored)

Value

A tibble

cells	<i>Get Cell IDs</i>
-------	---------------------

Description

Extract the unique cell IDs present in a HexData object.

Usage

```
cells(x)
```

Arguments

x A HexData object

Value

A vector of cell IDs

cell_area	<i>Compute per-cell area in km²</i>
-----------	--

Description

Returns the area of each cell in square kilometers. For ISEA grids, all cells have the same area (equal-area property). For H3 grids, each cell has a different geodesic area depending on its location.

Usage

```
cell_area(cell_id = NULL, grid)
```

Arguments

cell_id Cell IDs to compute area for. For ISEA grids, these are numeric; for H3 grids, character strings. When grid is a HexData object and cell_id is NULL, all cell IDs from the data are used.

grid A HexGridInfo or HexData object.

Details

For ISEA grids the area is constant across all cells and is read directly from the grid specification.

For H3 grids the area varies by latitude. The vendored 'H3' library computes each cell's spherical polygon area as a solid angle, which this function reads on the grid's body, so a grid built with radius_km reports that body's areas.

Value

Named numeric vector of areas in km², one per cell_id.

See Also

[hex_grid](#) for grid specifications, [h3_crosswalk](#) for ISEA/H3 interoperability

Examples

```
# ISEA: constant area
grid <- hex_grid(area_km2 = 1000)
cells <- lonlat_to_cell(c(0, 10, 20), c(45, 50, 55), grid)
cell_area(cells, grid)

# H3: area varies by location

h3 <- hex_grid(resolution = 5, type = "h3")
h3_cells <- lonlat_to_cell(c(0, 0), c(0, 80), h3)
cell_area(h3_cells, h3) # equator vs polar – different areas
```

cell_to_lonlat	<i>Convert cell ID to longitude/latitude</i>
----------------	--

Description

Converts DGGS cell IDs back to geographic coordinates (cell centers).

Usage

```
cell_to_lonlat(cell_id, grid)
```

Arguments

cell_id	Numeric vector of cell IDs
grid	A HexGridInfo or HexData object

Value

Data frame with lon_deg and lat_deg columns

See Also

[lonlat_to_cell](#) for the forward operation

Examples

```
grid <- hex_grid(area_km2 = 1000)
cells <- lonlat_to_cell(c(0, 10), c(45, 50), grid)
coords <- cell_to_lonlat(cells, grid)
```

cell_to_sf	<i>Convert cell IDs to sf polygons</i>
------------	--

Description

Creates sf polygon geometries for hexagonal grid cells.

Usage

```
cell_to_sf(cell_id = NULL, grid, wrap_dateline = TRUE)
```

Arguments

cell_id	Numeric vector of cell IDs. If NULL and x is HexData, uses cells from x.
grid	A HexGridInfo or HexData object. If HexData and cell_id is NULL, polygons are generated for all cells in the data.
wrap_dateline	Logical. If TRUE (default), calls <code>sf::st_wrap_dateline()</code> to split antimeridian-crossing polygons. Set to FALSE for orthographic/globe projections where wrapping creates gaps.

Details

When called with a HexData object and no cell_id argument, this function generates polygons for all unique cells in the data, which is useful for plotting.

Value

sf object with cell_id and geometry columns

See Also

[hex_grid](#) for grid specifications, [as_sf](#) for converting HexData to sf

Examples

```
# From grid specification
grid <- hex_grid(area_km2 = 1000)
cells <- lonlat_to_cell(c(0, 10, 20), c(45, 50, 55), grid)
polys <- cell_to_sf(cells, grid)

# From HexData (all cells)
df <- data.frame(lon = c(0, 10, 20), lat = c(45, 50, 55))
result <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000)
polys <- cell_to_sf(grid = result)
```

dgearthstat

Get grid statistics for whole-body coverage

Description

Calculates statistics about the hexagonal grid at the current resolution, including total number of cells, cell area, and cell spacing. Figures follow the radius the grid carries, so a grid built with `radius_km` reports that body's areas and distances.

Usage

```
dgearthstat(dggs)
```

Arguments

`dggs` Grid specification from `hexify_grid()`

Value

List with components:

<code>area_km</code>	Total surface area of the body in km^2
<code>n_cells</code>	Total number of cells at this resolution
<code>cell_area_km2</code>	Average cell area in km^2
<code>cell_spacing_km</code>	Average distance between cell centers in km
<code>resolution</code>	Resolution level
<code>aperture</code>	Grid aperture

See Also

Other grid statistics: [dg_closest_res_to_area\(\)](#), [hexify_area_to_eff_res\(\)](#), [hexify_compare_resolutions\(\)](#), [hexify_eff_res_to_area\(\)](#), [hexify_eff_res_to_resolution\(\)](#), [hexify_resolution_to_eff_res\(\)](#)

Examples

```
grid <- hexify_grid(area = 1000, aperture = 3)
stats <- dgearthstat(grid)

print(sprintf("Resolution %d has %.0f cells",
             stats$resolution, stats$n_cells))
print(sprintf("Average cell area: %.2f km^2",
             stats$cell_area_km2))
print(sprintf("Average cell spacing: %.2f km",
             stats$cell_spacing_km))
```

dggrid_is_compatible	<i>Validate 'dggridR' grid compatibility with hexify</i>
----------------------	--

Description

Checks whether a 'dggridR' grid object is compatible with hexify functions. Returns TRUE if compatible, or throws an error describing incompatibilities.

Usage

```
dggrid_is_compatible(dggs, strict = TRUE)
```

Arguments

dggs	A 'dggridR' grid object
strict	If TRUE (default), throw errors for incompatibilities. If FALSE, return FALSE instead of throwing errors.

Value

TRUE if compatible, FALSE if not compatible (when strict=FALSE)

See Also

Other 'dggridR' compatibility: [as_dggrid\(\)](#), [dggrid_43h_sequence\(\)](#), [from_dggrid\(\)](#)

dgverify	<i>Verify grid object</i>
----------	---------------------------

Description

Validates that a grid object has all required fields and valid values. This function is called internally by most hexify functions to ensure grid integrity.

Usage

```
dgverify(dggs)
```

Arguments

dggs	Grid object to verify (from <code>hexify_grid()</code> or <code>hex_grid()</code>)
------	---

Value

TRUE (invisibly) if valid, otherwise throws an error

Examples

```

grid <- hexify_grid(area = 1000, aperture = 3)
dgverify(grid) # Should pass silently

# Modern HexGridInfo objects are accepted too
dgverify(hex_grid(area_km2 = 1000))

# Invalid grid will throw error
bad_grid <- list(aperture = 5)
try(dgverify(bad_grid)) # Will error

```

from_dggrid

Convert 'dggridR' grid object to hexify_grid

Description

Creates a hexify_grid object from a 'dggridR' dggs object. This allows using hexify functions with grids created by 'dggridR' dgconstruct().

Usage

```
from_dggrid(dggs)
```

Arguments

dggs A 'dggridR' grid object from dgconstruct()

Details

Only 'ISEA' projection with HEXAGON topology is fully supported. Other configurations will generate warnings.

The function validates that the 'dggridR' grid uses compatible settings:

- Projection must be 'ISEA' (FULLER not supported)
- Topology must be "HEXAGON" (DIAMOND, TRIANGLE not supported)
- Aperture must be 3, 4, or 7

Value

A hexify_grid object

See Also

Other 'dggridR' compatibility: [as_dggrid\(\)](#), [dggrid_43h_sequence\(\)](#), [dggrid_is_compatible\(\)](#)

get_neighbors	<i>Get Neighboring Cells</i>
---------------	------------------------------

Description

Returns the k-ring (disk) of cells neighboring the input cells. For k = 1, returns the immediate 6 neighbors (5 for pentagons). For k > 1, returns all cells within k grid hops.

Usage

```
get_neighbors(cell_id, grid, k = 1L, include_self = FALSE, distances = FALSE)
```

Arguments

cell_id	Cell IDs to find neighbors for. Numeric vector for ISEA grids, character vector for H3 grids.
grid	A HexGridInfo or HexData object specifying the grid.
k	Integer. Ring distance (default 1). k = 1 returns immediate neighbors, k = 2 includes neighbors-of-neighbors, etc.
include_self	Logical. If TRUE, include the input cell in the result (default FALSE).
distances	Logical. If TRUE, return a data.frame with cell IDs and their ring distance from the origin (default FALSE).

Details

For **ISEA grids**, neighbors are computed using axial coordinate offsets in the quad IJ space. Cells at quad boundaries are handled by reprojection through lon/lat coordinates.

For **H3 grids**, neighbors use the vendored H3 gridDisk / gridDiskDistances functions.

Pentagon cells (the 12 icosahedron vertices) have only 5 neighbors instead of the usual 6.

Value

If distances = FALSE (default): a list of cell ID vectors, one per input cell. If distances = TRUE: a list of data.frames with columns cell_id and ring_distance.

See Also

[hexify\(\)](#) for creating HexData objects, [hex_distance\(\)](#) for grid distances between cells

Examples

```
# ISEA grid neighbors
g <- hex_grid(area_km2 = 1000)
cell <- lonlat_to_cell(10, 50, g)
nbrs <- get_neighbors(cell, g)
nbrs[[1]]
```

```
# H3 grid neighbors
g_h3 <- hex_grid(resolution = 5, type = "h3")
cell_h3 <- lonlat_to_cell(10, 50, g_h3)
get_neighbors(cell_h3, g_h3, k = 2)

# With distances
get_neighbors(cell, g, k = 2, distances = TRUE)
```

globe_centers

Globe center presets

Description

Named list of lon/lat coordinates for common globe views. Used by [plot_globe](#) when center is specified as a string.

Usage

```
globe_centers
```

Format

Named list with elements:

```
europe c(10, 50) - Western/Central Europe
north_america c(-100, 45) - USA and Canada
south_america c(-60, -15) - Full continent
africa c(20, 5) - Central Africa
asia c(100, 35) - China, SE Asia, Japan
oceania c(135, -25) - Australia, NZ, Indonesia
middle_east c(45, 25) - Arabian Peninsula, Iran, Turkey
south_asia c(80, 20) - India, Pakistan, Bangladesh
pacific c(-160, -10) - Polynesia, Pacific islands
caribbean c(-70, 18) - Caribbean islands
arctic c(0, 90) - North pole view
antarctic c(0, -90) - South pole view
```

Examples

```
globe_centers$europe
globe_centers$oceania
```

grid_clip	<i>Clip hexagon grid to polygon boundary</i>
-----------	--

Description

Creates hexagon polygons clipped to a given polygon boundary. This is useful for generating grids that conform to country borders, study areas, or other irregular boundaries.

Usage

```
grid_clip(boundary, grid, crop = TRUE)
```

Arguments

boundary	An sf/sfc polygon to clip to. Can be a country boundary, study area, or any polygon geometry.
grid	A HexGridInfo object specifying the grid parameters
crop	If TRUE (default), cells are cropped to the boundary. If FALSE, only cells whose centroids fall within the boundary are kept (no cropping).

Details

The function first generates cells covering the boundary polygon, then clips or filters them. For H3 grids, all cells that overlap the boundary are included (not just cells whose center falls inside), ensuring full spatial coverage with no gaps along the boundary edge.

When crop = TRUE, hexagons are geometrically intersected with the boundary, which may produce partial hexagons at the edges. When crop = FALSE, only complete hexagons whose centroids fall within the boundary are returned.

Value

sf object with hexagon polygons clipped to the boundary

See Also

[grid_rect](#) for rectangular grids, [grid_global](#) for global grids

Examples

```
# Get France boundary from built-in world map
france <- hexify_world[hexify_world$name == "France", ]

# Create grid clipped to France
grid <- hex_grid(area_km2 = 2000)
france_grid <- grid_clip(france, grid)

# Plot result
library(ggplot2)
```

```
ggplot() +
  geom_sf(data = france, fill = "gray95") +
  geom_sf(data = france_grid, fill = alpha("steelblue", 0.3),
    color = "steelblue") +
  theme_minimal()

# Keep only complete hexagons (no cropping)
france_grid_complete <- grid_clip(france, grid, crop = FALSE)
```

grid_global	<i>Generate a global hexagon grid</i>
-------------	---------------------------------------

Description

Creates hexagon polygons covering the entire Earth.

Usage

```
grid_global(grid, wrap_dateline = TRUE)
```

Arguments

grid	A HexGridInfo object specifying the grid parameters
wrap_dateline	Logical. If TRUE (default), antimeridian-crossing polygons are split at +/-180 degrees. Set to FALSE for orthographic/globe projections where wrapping creates gaps.

Details

This function generates a complete global grid by sampling points densely across the globe. For large grids (many small cells), consider using `grid_rect()` to generate regional subsets.

Value

sf object with hexagon polygons

See Also

[grid_rect](#) for regional grids

Examples

```
# Coarse global grid
grid <- hex_grid(area_km2 = 100000)
global <- grid_global(grid)
plot(global)
```

grid_info	<i>Get Grid Specification</i>
-----------	-------------------------------

Description

Extract the grid specification from a HexData object.

Usage

```
grid_info(x)
```

Arguments

x	A HexData object
---	------------------

Value

A HexGridInfo object

Examples

```
df <- data.frame(lon = c(0, 10, 20), lat = c(45, 50, 55))
result <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000)
grid_spec <- grid_info(result)
```

grid_rect	<i>Generate a rectangular grid of hexagons</i>
-----------	--

Description

Creates hexagon polygons covering a rectangular geographic region. For H3 grids, all cells that overlap the bounding box are included (not just cells whose center falls inside), ensuring full spatial coverage.

Usage

```
grid_rect(bbox, grid)
```

Arguments

bbox	Bounding box as c(xmin, ymin, xmax, ymax), or an sf/sfc object
grid	A HexGridInfo object specifying the grid parameters

Value

sf object with hexagon polygons

See Also

[grid_global](#) for global grids

Examples

```
grid <- hex_grid(area_km2 = 5000)
europe <- grid_rect(c(-10, 35, 30, 60), grid)
plot(europe)
```

h3_crosswalk

*Crosswalk Between ISEA and H3 Cell IDs***Description**

Maps cell IDs between ISEA (equal-area) and H3 grid systems by looking up each cell's center coordinate in the target grid. This enables workflows where analysis is done in ISEA (exact equal-area) and reporting in H3 (industry-standard).

Usage

```
h3_crosswalk(
  cell_id = NULL,
  grid,
  h3_resolution = NULL,
  isea_grid = NULL,
  direction = c("isea_to_h3", "h3_to_isea")
)
```

Arguments

cell_id	Cell IDs to translate. Numeric for ISEA, character for H3. When grid is a HexData object and cell_id is NULL, all cell IDs from the data are used.
grid	A HexGridInfo or HexData object. For direction = "isea_to_h3", this must be an ISEA grid. For direction = "h3_to_isea", this must be an H3 grid.
h3_resolution	Target H3 resolution for "isea_to_h3". When NULL (default), the closest H3 resolution matching the ISEA cell area is selected automatically. For "h3_to_isea", the source H3 resolution is always inferred from grid; if h3_resolution is supplied for that direction it is validated against grid's resolution rather than used to select one.
isea_grid	A HexGridInfo for the target ISEA grid. Required when direction = "h3_to_isea".
direction	One of "isea_to_h3" (default) or "h3_to_isea".

Details

The crosswalk works by computing the center coordinate of each source cell, then finding which cell in the target grid contains that center. This is a many-to-one mapping: multiple ISEA cells may map to the same H3 cell (or vice versa) depending on the relative resolutions.

When `h3_resolution` is `NULL` and `direction = "isea_to_h3"`, the H3 resolution whose average cell area is closest to the ISEA cell area is chosen automatically. This gives the best 1:1 correspondence.

Value

A data frame with columns:

isea_cell_id ISEA cell ID (numeric)

h3_cell_id H3 cell ID (character)

isea_area_km2 Area of the ISEA cell in km2

h3_area_km2 Geodesic area of the H3 cell in km2

area_ratio Ratio of ISEA area to H3 area

See Also

[cell_area](#) for per-cell area computation, [hex_grid](#) for creating grids

Examples

```
# ISEA -> H3
grid <- hex_grid(area_km2 = 1000)
cells <- lonlat_to_cell(c(0, 10, 20), c(45, 50, 55), grid)
xwalk <- h3_crosswalk(cells, grid)
head(xwalk)

# H3 -> ISEA
h3 <- hex_grid(resolution = 5, type = "h3")
h3_cells <- lonlat_to_cell(c(0, 10), c(45, 50), h3)
xwalk2 <- h3_crosswalk(h3_cells, h3, isea_grid = grid, direction = "h3_to_isea")
```

HexData-class

HexData Class

Description

An S4 class representing hexified data. Contains the original user data plus cell assignments from the hexification process.

Details

HexData objects are created by [hexify](#). The original data is preserved in the data slot, while cell assignments are stored separately in `cell_id` and `cell_center`.

Use `as.data.frame()` to get a combined data frame with cell columns.

Slots

`data` Data frame or sf object. The original user data (untouched).

`grid` HexGridInfo object. The grid specification used.

`cell_id` Cell IDs for each row of data. Numeric for ISEA grids, character for H3 grids.

`cell_center` Matrix. Two-column matrix (lon, lat) of cell centers.

See Also

[hexify](#) for creating HexData objects, [HexGridInfo-class](#) for grid specifications

HexGridInfo-class	<i>HexGridInfo Class</i>
-------------------	--------------------------

Description

An S4 class representing a hexagonal grid specification. Stores all parameters needed for grid operations.

Details

Create HexGridInfo objects using the [hex_grid](#) constructor function. Do not use `new("HexGridInfo", ...)` directly.

The aperture can be "3", "4", "7" for grids that refine by one aperture at every level; a family name such as "4/3" or "4/7", which refines by the first aperture for the first floor(resolution / 2) levels and by the second for the rest; or one aperture per level, "4/4/7/3".

For H3 grids, the aperture is fixed at "7" and resolution ranges from 0 to 15.

Slots

`aperture` Character. Grid aperture: "3", "4", "7", a mixed family such as "4/3" or "4/7", or one aperture per resolution level ("4/4/7/3").

`resolution` Integer. Grid resolution level (0-30 for ISEA, 0-15 for H3).

`area_km2` Numeric. Cell area in square kilometers.

`diagonal_km` Numeric. Cell diagonal (long diagonal) in kilometers.

`crs` Integer or character. Coordinate reference system: an EPSG code, or a 'PROJ' or 'WKT' string. Defaults to 'WGS84' on Earth, and to a longlat CRS on the sphere of `radius_km` on any other body.

`grid_type` Character. Grid system: "isea" (default) or "h3".

`radius_km` Numeric. Radius of the body the grid covers, in kilometers. NA reads as Earth's mean radius.

See Also

[hex_grid](#) for the constructor function, [HexData-class](#) for hexified data objects

hexify

Assign hexagonal DGGS cell IDs to geographic points

Description

Takes a data.frame or sf object with geographic coordinates and returns a HexData object that stores the original data plus cell assignments. The original data is preserved unchanged; cell IDs and centers are stored in separate slots.

Usage

```
hexify(
  data,
  grid = NULL,
  lon = "lon",
  lat = "lat",
  area_km2 = NULL,
  diagonal = NULL,
  resolution = NULL,
  aperture = 3,
  resround = "nearest",
  radius_km = EARTH_RADIUS_KM
)
```

Arguments

data	A data.frame or sf object containing coordinates
grid	A HexGridInfo object from hex_grid() . If provided, overrides area_km2, resolution, aperture and radius_km parameters.
lon	Column name for longitude (ignored if data is sf)
lat	Column name for latitude (ignored if data is sf)
area_km2	Target cell area in km ² (mutually exclusive with diagonal).
diagonal	Target cell diagonal (long diagonal) in km
resolution	Grid resolution (0-30). Alternative to area_km2.
aperture	Grid aperture: 3, 4, 7, a mixed family such as "4/3" or "4/7", or one aperture per resolution level, e.g. c(4, 4, 7, 3) (default 3)
resround	How to round resolution: "nearest", "up", or "down"
radius_km	Radius of the body the grid covers, in kilometers, or a body name such as "mars" (default Earth). See hex_grid .

Details

For sf objects, coordinates are automatically extracted and transformed to 'WGS84' (EPSG:4326) if needed. The geometry column is preserved.

Either `area_km2`, `diagonal`, or `resolution` must be provided unless a grid object is supplied.

The `HexData` return type (default) stores the grid specification so downstream functions like `plot()`, `hexify_cell_to_sf()`, etc. don't need grid parameters repeated.

Value

A `HexData` object containing:

- `data`: The original input data (unchanged)
- `grid`: The `HexGridInfo` specification
- `cell_id`: Numeric vector of cell IDs for each row
- `cell_center`: Matrix of cell center coordinates (lon, lat)

Use `as.data.frame(result)` to extract the original data. Use `cells(result)` to get unique cell IDs. Use `result@cell_id` to get all cell IDs. Use `result@cell_center` to get cell center coordinates.

Grid Specification

You can create a grid specification once and reuse it:

```
grid <- hex_grid(area_km2 = 1000)
result1 <- hexify(df1, grid = grid)
result2 <- hexify(df2, grid = grid)
```

See Also

[hex_grid](#) for grid specification, [HexData-class](#) for return object details, [as_sf](#) for converting to sf

Other hexify main: [hexify_grid\(\)](#)

Examples

```
# Simple data.frame
df <- data.frame(
  site = c("Vienna", "Paris", "Madrid"),
  lon = c(16.37, 2.35, -3.70),
  lat = c(48.21, 48.86, 40.42)
)

# New recommended workflow: use grid object
grid <- hex_grid(area_km2 = 1000)
result <- hexify(df, grid = grid, lon = "lon", lat = "lat")
print(result) # Shows grid info
plot(result)  # Plot with default styling
```

```
# Direct area specification (grid created internally)
result <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000)

# Extract plain data.frame
df_result <- as.data.frame(result)

# With sf object (any CRS)
library(sf)
pts <- st_as_sf(df, coords = c("lon", "lat"), crs = 4326)
result_sf <- hexify(pts, area_km2 = 1000)

# Different apertures
result_ap4 <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000, aperture = 4)

# Mixed aperture (ISEA43H)
result_mixed <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000, aperture = "4/3")
```

hexify-conversions	<i>Coordinate Conversions</i>
--------------------	-------------------------------

Description

Functions for converting between coordinate systems

hexify-grid	<i>Core Grid Construction</i>
-------------	-------------------------------

Description

Core functions for hexify grid construction and validation

hexify-stats	<i>Grid Statistics</i>
--------------	------------------------

Description

Functions for calculating grid statistics and utilities

hexify_build_icosah	<i>Initialize icosahedron geometry</i>
---------------------	--

Description

Sets up the icosahedron state for ISEA projection. Uses standard ISEA3H orientation by default (vertex 0 at 11.25E, 58.28N).

Usage

```
hexify_build_icosah(  
    vert0_lon = ISEA_VERT0_LON_DEG,  
    vert0_lat = ISEA_VERT0_LAT_DEG,  
    azimuth = ISEA_AZIMUTH_DEG  
)
```

Arguments

vert0_lon	Vertex 0 longitude in degrees (default ISEA_VERT0_LON_DEG)
vert0_lat	Vertex 0 latitude in degrees (default ISEA_VERT0_LAT_DEG)
azimuth	Azimuth rotation in degrees (default ISEA_AZIMUTH_DEG)

Details

The icosahedron is initialized lazily at the C++ level when first needed. Manual call is only required for non-standard orientations.

Value

Invisible NULL. Called for side effect.

See Also

Other projection: [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

Examples

```
# Use standard ISEA3H orientation  
hexify_build_icosah()  
  
# Custom orientation  
hexify_build_icosah(vert0_lon = 0, vert0_lat = 90, azimuth = 0)
```

hexify_compare_resolutions

Compare grid resolutions

Description

Generates a table comparing different resolution levels for a given grid configuration. Useful for choosing appropriate resolution.

Usage

```
hexify_compare_resolutions(
  aperture = 3,
  res_range = 0:15,
  type = c("isea", "h3"),
  print = FALSE,
  radius_km = EARTH_RADIUS_KM
)
```

Arguments

aperture	Grid aperture (3, 4, or 7). Ignored for H3 grids.
res_range	Range of resolutions to compare (e.g., 1:10)
type	Grid type: "isea" (default) or "h3".
print	If TRUE, prints a formatted table to console. If FALSE (default), returns a data frame.
radius_km	Radius of the body, in kilometers, or a body name such as "mars" (default Earth). See hex_grid .

Value

If print=FALSE: data frame with columns resolution, n_cells, cell_area_km2, cell_spacing_km, cls_km. If print=TRUE: invisibly returns the data frame after printing.

See Also

Other grid statistics: [dg_closest_res_to_area\(\)](#), [dgearthstat\(\)](#), [hexify_area_to_eff_res\(\)](#), [hexify_eff_res_to_area\(\)](#), [hexify_eff_res_to_resolution\(\)](#), [hexify_resolution_to_eff_res\(\)](#)

Examples

```
# Get data frame of resolutions 0-10 for aperture 3
comparison <- hexify_compare_resolutions(aperture = 3, res_range = 0:10)
print(comparison)

# Print formatted table directly
hexify_compare_resolutions(aperture = 3, res_range = 0:10, print = TRUE)
```



```
# Find resolution with cells ~1000 km^2
subset(comparison, cell_area_km2 > 900 & cell_area_km2 < 1100)

# Resolutions on another body
hexify_compare_resolutions(aperture = 3, res_range = 0:6, radius_km = "mars")
```

hexify_face_centers *Get icosahedron face centers*

Description

Returns the center coordinates of all 20 icosahedral faces.

Usage

```
hexify_face_centers()
```

Value

Data frame with 20 rows and columns lon, lat (degrees)

See Also

Other projection: [hexify_build_icosahedron\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

Examples

```
centers <- hexify_face_centers()
plot(centers$lon, centers$lat)
```

hexify_forward *Forward Snyder projection*

Description

Projects geographic coordinates onto the icosahedron, returning face index and planar coordinates (tx, ty).

Usage

```
hexify_forward(lon, lat)
```

Arguments

lon	Longitude in degrees
lat	Latitude in degrees

Details

tx and ty are normalized coordinates within the triangular face, typically in range [0, 1].

Value

Named numeric vector: c(face, tx, ty)

See Also

Other projection: [hexify_build_icosahedron\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

Examples

```
result <- hexify_forward(16.37, 48.21)
# result["face"], result["icosa_triangle_x"], result["icosa_triangle_y"]
```

hexify_forward_to_face

Forward projection to specific face

Description

Projects to a known face (skips face detection).

Usage

```
hexify_forward_to_face(face, lon, lat)
```

Arguments

face	Face index (0-19)
lon	Longitude in degrees
lat	Latitude in degrees

Value

Named numeric vector: c(icosa_triangle_x, icosa_triangle_y)

See Also

Other projection: [hexify_build_icosa\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

hexify_get_precision *Get current precision settings*

Description

Get current precision settings

Usage

```
hexify_get_precision()
```

Value

List with tol and max_iters

See Also

Other projection: [hexify_build_icosa\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

hexify_grid *Create a hexagonal grid specification*

Description

Creates a discrete global grid system (DGGS) object with hexagonal cells at a specified resolution. This is the main constructor for hexify grids.

Usage

```
hexify_grid(
  area,
  topology = "HEXAGON",
  metric = TRUE,
  resround = "nearest",
  aperture = 3,
  projection = "ISEA",
  radius_km = EARTH_RADIUS_KM
)
```

Arguments

area	Target cell area in km ² (if metric=TRUE) or area code
topology	Grid topology (only "HEXAGON" supported)
metric	Whether area is in metric units (km ²)
resround	How to round resolution ("nearest", "up", "down")
aperture	Aperture sequence (3, 4, or 7)
projection	Projection type (only 'ISEA' supported currently)
radius_km	Radius of the body the grid covers, in kilometers, or a body name such as "mars" (default Earth). See hex_grid .

Value

A hexify_grid object containing:

area	Target cell area
resolution	Calculated resolution level
aperture	Grid aperture (3, 4, or 7)
topology	Grid topology ("HEXAGON")
projection	Map projection ("ISEA")
radius_km	Radius of the body, in kilometers
index_type	Index encoding type ("z3", "z7", or "zorder")

See Also

[hexify](#) for the main user function, [hexify_grid_to_cell](#) for coordinate conversion

Other hexify main: [hexify\(\)](#)

Examples

```
# Create a grid with ~1000 km^2 cells
grid <- hexify_grid(area = 1000, aperture = 3)
print(grid)

# Create a finer resolution grid (~100 km^2 cells)
fine_grid <- hexify_grid(area = 100, aperture = 3, resround = "up")
```

hexify_heatmap	Create a ggplot2 visualization of hexagonal grid cells
----------------	--

Description

Creates a ggplot2-based visualization of hexagonal grid cells, optionally colored by a value column. Supports continuous and discrete color scales, projection transformation, and customizable styling.

Usage

```
hexify_heatmap(
  data,
  value = NULL,
  basemap = NULL,
  crs = NULL,
  colors = NULL,
  breaks = NULL,
  labels = NULL,
  hex_border = "#5D4E37",
  hex_lwd = 0.3,
  hex_alpha = 0.7,
  basemap_fill = "gray90",
  basemap_border = "gray50",
  basemap_lwd = 0.5,
  mask_outside = FALSE,
  aperture = 3L,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  legend_title = NULL,
  na_color = "gray90",
  theme_void = TRUE
)
```

Arguments

data	A HexData object from <code>hexify()</code> , a data frame with <code>cell_id</code> and <code>cell_area</code> columns, or an <code>sf</code> object with hexagon polygons.
value	Column name (as string) to use for fill color. If <code>NULL</code> , cells are drawn with a uniform fill color. If not specified but data has a 'count' or 'n' column, that will be used automatically.
basemap	Optional basemap. Can be: • <code>NULL</code>: No basemap (default) • <code>"world"</code>: Use built-in <code>hexify_world</code> map (low resolution) • <code>"world_hires"</code>: Use high-resolution map from <code>rnaturalearth</code> (requires package)

	• An sf object: User-supplied vector map, from any format <code>sf::st_read()</code> reads, such as a shapefile, 'GeoJSON' or 'GeoPackage' • A SpatRaster: User-supplied raster, from any format <code>terra::rast()</code> reads, such as a 'GeoTIFF', drawn in grayscale
<code>crs</code>	Target CRS for the map projection. Can be: • A numeric EPSG code (e.g., 4326 for 'WGS84', 3035 for LAEA Europe) • A proj4 string • An sf crs object • NULL to use 'WGS84' (EPSG:4326)
<code>colors</code>	Color palette for the heatmap. Can be: • A character vector of colors (for manual scale) • A single RColorBrewer palette name (e.g., "YlOrRd", "Greens") • NULL to use viridis
<code>breaks</code>	Numeric vector of break points for binning continuous values, or NULL for continuous scale. Use Inf and -Inf for open-ended bins.
<code>labels</code>	Labels for the breaks (length should be one less than breaks). If NULL, labels are auto-generated.
<code>hex_border</code>	Border color for hexagons
<code>hex_lwd</code>	Line width for hexagon borders
<code>hex_alpha</code>	Transparency for hexagon fill (0-1)
<code>basemap_fill</code>	Fill color for basemap polygons
<code>basemap_border</code>	Border color for basemap polygons
<code>basemap_lwd</code>	Line width for basemap borders
<code>mask_outside</code>	Logical. If TRUE and basemap is provided, mask hexagon portions that fall outside the basemap polygons.
<code>aperture</code>	Grid aperture (default 3), used if data is from hexify()
<code>xlim</code>	Optional x-axis limits (in target CRS units) as c(min, max)
<code>ylim</code>	Optional y-axis limits (in target CRS units) as c(min, max)
<code>title</code>	Plot title
<code>legend_title</code>	Title for the color legend
<code>na_color</code>	Color for NA values
<code>theme_void</code>	Logical. If TRUE (default), use a minimal theme without axes, gridlines, or background.

Details

This function provides publication-quality heatmap visualizations of hexagonal grids using ggplot2. It returns a ggplot object that can be further customized with standard ggplot2 functions.

Value

A ggplot2 object that can be further customized or saved.

Color Scales

The function supports three types of color scales:

Continuous Set breaks = NULL for a continuous gradient

Binned Provide breaks vector to bin values into categories

Discrete If value column is a factor, discrete colors are used

Projections

Common projections:

4326 'WGS84' (unprojected lat/lon)

3035 LAEA Europe

3857 Web Mercator

"`+proj=robin`" Robinson (world maps)

"`+proj=moll`" Mollweide (equal-area world maps)

See Also

[plot_grid](#) for base R plotting, [cell_to_sf](#) to generate polygons manually

Other visualization: [plot_world\(\)](#)

Examples

```
library(hexify)

# Sample data with counts
cities <- data.frame(
  lon = c(16.37, 2.35, -3.70, 12.5, 4.9),
  lat = c(48.21, 48.86, 40.42, 41.9, 52.4),
  count = c(100, 250, 75, 180, 300)
)
result <- hexify(cities, lon = "lon", lat = "lat", area_km2 = 5000)

# Simple plot (uniform fill, no value mapping)
hexify_heatmap(result)

library(ggplot2)

# With world basemap
hexify_heatmap(result, basemap = "world")

# Heatmap with value mapping
hexify_heatmap(result, value = "count")

# With world basemap and custom colors
hexify_heatmap(result, value = "count",
  basemap = "world",
```

```

        colors = "YlOrRd",
        title = "City Density")

# Binned values with custom breaks
hexify_heatmap(result, value = "count",
               basemap = "world",
               breaks = c(-Inf, 100, 200, Inf),
               labels = c("Low", "Medium", "High"),
               colors = c("#fee8c8", "#fdbb84", "#e34a33"))

# Different projection (LAEA Europe)
hexify_heatmap(result, value = "count",
               basemap = "world",
               crs = 3035,
               xlim = c(2500000, 6500000),
               ylim = c(1500000, 5500000))

# Customize further with ggplot2
hexify_heatmap(result, value = "count", basemap = "world") +
  labs(caption = "Data source: Example") +
  theme(legend.position = "bottom")

```

hexify_inverse	<i>Inverse Snyder projection</i>
----------------	----------------------------------

Description

Converts face plane coordinates back to geographic coordinates.

Usage

```
hexify_inverse(x, y, face, tol = NULL, max_iters = NULL)
```

Arguments

x	X coordinate on face plane
y	Y coordinate on face plane
face	Face index (0-19)
tol	Convergence tolerance (NULL for default)
max_iters	Maximum iterations (NULL for default)

Value

Named numeric vector: c(lon_deg, lat_deg)

See Also

Other projection: [hexify_build_icosahedron\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

Examples

```
coords <- hexify_inverse(0.5, 0.3, face = 2)
```

`hexify_projection_stats`*Get inverse projection statistics*

Description

Returns and optionally resets convergence statistics.

Usage

```
hexify_projection_stats(reset = TRUE)
```

Arguments

<code>reset</code>	Whether to reset statistics after retrieval (default TRUE)
--------------------	--

Value

List with statistics (iterations, convergence info, etc.)

See Also

Other projection: [hexify_build_icosahedron\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

hexify_roundtrip_test	<i>Round-trip accuracy test</i>
-----------------------	---------------------------------

Description

Tests the accuracy of the coordinate conversion functions by converting coordinates to cells and back, measuring the distance between original and reconstructed coordinates.

Usage

```
hexify_roundtrip_test(grid, lon, lat, units = "km")
```

Arguments

grid	Grid specification
lon	Longitude to test
lat	Latitude to test
units	Distance units ("km" or "degrees")

Value

List with:

original	Original coordinates
cell	Cell index
reconstructed	Reconstructed coordinates
error	Distance between original and reconstructed

See Also

Other coordinate conversion: [hexify_cell_id_to_quad_ij\(\)](#), [hexify_cell_to_icosa_tri\(\)](#), [hexify_cell_to_lonlat\(\)](#), [hexify_cell_to_plane\(\)](#), [hexify_cell_to_quad_ij\(\)](#), [hexify_cell_to_quad_xy\(\)](#), [hexify_grid_cell_to_lonlat\(\)](#), [hexify_grid_to_cell\(\)](#), [hexify_icosa_tri_to_plane\(\)](#), [hexify_icosa_tri_to_quad_ij\(\)](#), [hexify_icosa_tri_to_quad_xy\(\)](#), [hexify_lonlat_to_cell\(\)](#), [hexify_lonlat_to_plane\(\)](#), [hexify_lonlat_to_quad_ij\(\)](#), [hexify_quad_ij_to_cell\(\)](#), [hexify_quad_ij_to_icosa_tri\(\)](#), [hexify_quad_ij_to_xy\(\)](#), [hexify_quad_xy_to_cell\(\)](#), [hexify_quad_xy_to_icosa_tri\(\)](#)

hexify_set_precision *Set inverse projection precision*

Description

Controls the accuracy/speed tradeoff for inverse Snyder projection.

Usage

```
hexify_set_precision(
  mode = c("fast", "default", "high", "ultra"),
  tol = NULL,
  max_iters = NULL
)
```

Arguments

mode	Preset mode: "fast", "default", "high", or "ultra"
tol	Custom tolerance (overrides mode if provided)
max_iters	Custom max iterations (overrides mode if provided)

Value

Invisible NULL

See Also

Other projection: [hexify_build_icosa\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_verbose\(\)](#), [hexify_which_face\(\)](#)

Examples

```
hexify_set_precision("high")
hexify_set_precision(tol = 1e-12, max_iters = 100)
```

hexify_set_verbose *Set verbose mode for inverse projection*

Description

When enabled, prints convergence information.

Usage

```
hexify_set_verbose(verbose = TRUE)
```

Arguments

verbose Logical

Value

Invisible NULL

See Also

Other projection: [hexify_build_icosah\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_which_face\(\)](#)

hexify_which_face	<i>Determine which face contains a point</i>
-------------------	--

Description

Returns the icosahedral face index (0-19) containing the given coordinates.

Usage

hexify_which_face(lon, lat)

Arguments

lon Longitude in degrees
lat Latitude in degrees

Value

Integer face index (0-19)

See Also

Other projection: [hexify_build_icosah\(\)](#), [hexify_face_centers\(\)](#), [hexify_forward\(\)](#), [hexify_forward_to_face\(\)](#), [hexify_get_precision\(\)](#), [hexify_inverse\(\)](#), [hexify_projection_stats\(\)](#), [hexify_set_precision\(\)](#), [hexify_set_verbose\(\)](#)

Examples

```
face <- hexify_which_face(16.37, 48.21)
```

hexify_world	<i>Simplified World Map</i>
--------------	-----------------------------

Description

A lightweight sf object containing simplified world country borders, suitable for use as a basemap when visualizing hexagonal grids.

Usage

```
hexify_world
```

Format

An sf object with 177 features and 15 fields:

name Country short name
name_long Country full name
admin Administrative name
sovereignty Sovereignty
iso_a2 ISO 3166-1 alpha-2 country code
iso_a3 ISO 3166-1 alpha-3 country code
iso_n3 ISO 3166-1 numeric code
continent Continent name
region_un UN region
subregion UN subregion
region_wb World Bank region
pop_est Population estimate
gdp_md GDP in millions USD
income_grp Income group classification
economy Economy type
geometry MULTIPOLYGON geometry in 'WGS84' (EPSG:4326)

Source

Simplified from Natural Earth 1:110m Cultural Vectors (<https://www.naturalearthdata.com/>)

Examples

```
library(sf)

# Plot the built-in world map
plot(st_geometry(hexify_world), col = "lightgray", border = "white")

# Filter by continent
europe <- hexify_world[hexify_world$continent == "Europe", ]
plot(st_geometry(europe))
```

hex_browse

Interactive Hex Map

Description

Opens an interactive leaflet map with hexagonal cell polygons. Cells can be colored by a data column, with popups showing cell information on click.

Usage

```
hex_browse(
  hex_data,
  grid = NULL,
  value = NULL,
  palette = "viridis",
  opacity = 0.7
)
```

Arguments

hex_data	A HexData object, or a data.frame with a cell_id column.
grid	A HexGridInfo object. Required if hex_data is a data.frame.
value	Optional column name (character) to color cells by. If NULL, cells are colored uniformly.
palette	Color palette name for continuous values. Default "viridis".
opacity	Fill opacity (0-1). Default 0.7.

Details

Requires the leaflet package (in Suggests). The map is built using `as_sf()` to generate polygon geometries, then rendered as a leaflet choropleth.

Value

A leaflet map object (can be printed or embedded in Shiny).

See Also

[hexify_heatmap\(\)](#) for static ggplot2 maps, [as_sf\(\)](#) for sf conversion

Examples

```
if (requireNamespace("leaflet", quietly = TRUE)) {
  df <- data.frame(
    lon = runif(50, -5, 5),
    lat = runif(50, 45, 55),
    value = rnorm(50)
  )
  hd <- hexify(df, lon = "lon", lat = "lat", area_km2 = 500)
  hex_browse(hd, value = "value")
}
```

hex_compact

Compact Hex Cells

Description

Merges child cells into their parent when all children are present. This is a lossless compression — no spatial information is lost. The compact representation uses fewer cells to cover the same area.

Usage

```
hex_compact(cell_ids, grid)
```

Arguments

cell_ids	Cell IDs to compact. For H3 grids, a character vector. For ISEA grids, a character vector of hierarchical index strings.
grid	A HexGridInfo object specifying the grid.

Details

H3 backend: Uses the vendored H3 compactCells function.

ISEA backend (aperture 7, Z7 index): Groups cells by parent index (dropping the last digit). If all 7 children are present, replaces them with the parent. Iterates until no further compaction is possible.

Value

A character vector of compacted cell IDs. Cells that could be merged into parents appear as parent IDs at coarser resolution.

See Also

[hex_uncompact\(\)](#) for the inverse operation, [get_parent\(\)](#), [get_children\(\)](#) for hierarchical operations

Examples

```
# H3 compaction
g <- hex_grid(resolution = 3, type = "h3")
parent <- "832830fffffffff"
children <- get_children(parent, g)[[1]]
compact <- hex_compact(children, g)
compact # Should return the parent
```

hex_distance

Grid Distance Between Cells

Description

Computes the grid distance (minimum number of hops) between pairs of hexagonal cells. This is a discrete distance measured in cell steps, not a geodesic distance.

Usage

```
hex_distance(cell_a, cell_b, grid)
```

Arguments

cell_a, cell_b Cell IDs. Must be the same length (pairs) or one of them length 1 (broadcast). For H3 grids, character vectors. For ISEA grids, numeric vectors.

grid A HexGridInfo or HexData object specifying the grid.

Details

H3 backend: Uses the vendored H3 `gridDistance` function. Returns the exact shortest path length in cell hops.

ISEA backend: For cells on the same quad, computes the cube-coordinate distance: $\max(|di|, |dj|, |di + dj|)$. Cross-quad distances use BFS expansion and may return NA for very distant cells.

For geodesic (geographic) distances between cell centers, convert to lon/lat with [cell_to_lonlat\(\)](#) and use `sf::st_distance()`.

Value

An integer vector of grid distances. NA where the distance cannot be computed (e.g., pentagon path issues in H3).

See Also

[get_neighbors\(\)](#) for finding cells within a given distance, [cell_to_lonlat\(\)](#) for geographic coordinates

Examples

```
# H3 grid distance
g <- hex_grid(resolution = 5, type = "h3")
a <- lonlat_to_cell(10, 50, g)
b <- lonlat_to_cell(10.1, 50.1, g)
hex_distance(a, b, g)
```

hex_extract	<i>Extract Raster Values at Hex Cell Centers</i>
-------------	--

Description

Samples raster values at hexagonal cell centers. Faster than [hex_zonal\(\)](#) because it only queries cell center points, not full polygons.

Usage

```
hex_extract(raster, grid, cells = NULL, boundary = NULL)
```

Arguments

raster	A <code>terra::SpatRaster</code> object.
grid	A <code>HexGridInfo</code> or <code>HexData</code> object specifying the grid.
cells	Optional cell IDs to extract. If <code>NULL</code> (default), extracts at all cell centers from grid (only works for <code>HexData</code> objects or if a boundary is provided).
boundary	Optional <code>sf</code> polygon to limit extraction extent.

Details

Requires the `terra` package (in Suggests). The function:

1. Generates cell centers for the raster extent
2. Calls `terra::extract(raster, cell_center_matrix)`
3. Attaches cell IDs

For full zonal statistics (aggregating all pixels within each hex polygon), use [hex_zonal\(\)](#) instead.

Value

A `data.frame` with columns `cell_id`, plus one column per raster layer.

See Also

[hex_zonal\(\)](#) for polygon-based zonal statistics, [hexify\(\)](#) for creating HexData objects

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  # Create a small synthetic raster
  r <- terra::rast(nrows = 10, ncols = 10,
                  xmin = -10, xmax = 10, ymin = 40, ymax = 55)
  terra::values(r) <- runif(100)
  names(r) <- "temperature"

  # Extract at hex cell centers
  g <- hex_grid(area_km2 = 500)
  df <- data.frame(lon = c(0, 5), lat = c(45, 50))
  hd <- hexify(df, lon = "lon", lat = "lat", grid = g)
  hex_extract(r, hd)
}
```

hex_grid

Create a Hexagonal Grid Specification

Description

Creates a HexGridInfo object that stores all parameters needed for hexagonal grid operations. Use this to define the grid once and pass it to all downstream functions.

Usage

```
hex_grid(
  area_km2 = NULL,
  resolution = NULL,
  aperture = 3,
  type = c("isea", "h3"),
  resround = "nearest",
  crs = NULL,
  radius_km = EARTH_RADIUS_KM
)
```

Arguments

area_km2	Target cell area in square kilometers. Mutually exclusive with resolution.
resolution	Grid resolution level (0-30 for ISEA, 0-15 for H3). Mutually exclusive with area_km2. For H3, typical use cases by resolution: • 0-3: continental/country scale • 4-7: regional/city scale

	• 8-10: neighborhood/block scale (FCC uses 8-9) • 11-15: building/sub-meter scale
aperture	Grid aperture: 3 (default), 4, 7, a mixed family such as "4/3", "4/7" or "7/4", or one aperture per resolution level as a vector, e.g. c(4, 4, 7, 3). A family name refines by the first aperture for the first floor(resolution / 2) levels and by the second for the rest, which is how DGGRID arranges ISEA43H. A per-level vector needs resolution rather than area_km2. Ignored for H3 grids (fixed at 7).
type	Grid type: "isea" (default) or "h3".
resround	Resolution rounding when using area_km2: "nearest" (default), "up", or "down".
crs	Coordinate reference system: an EPSG code, or a 'PROJ' or 'WKT' string. Defaults to 'WGS84' on Earth, and to a longlat CRS on the sphere of radius_km on any other body, which has no EPSG code to name it.
radius_km	Radius of the body the grid covers, in kilometers, or the name of a body: "mercury", "venus", "earth" (default), "moon", "mars", "ceres", "jupiter", "io", "europa", "ganymede", "callisto", "saturn", "enceladus", "titan", "uranus", "neptune", "pluto".

Details

Exactly one of area_km2 or resolution must be provided.

When area_km2 is provided, the resolution is calculated automatically using the cell count formula: $N = 10 * aperture^{res} + 2$ (ISEA) or by matching the closest H3 resolution.

H3 grids use the Uber H3 hierarchical hexagonal system. Unlike ISEA grids, H3 cells are NOT exactly equal-area (area varies by ~3-5\ location).

Value

A HexGridInfo object containing the grid specification.

Other Bodies

A grid is a partition of the sphere, and radius_km sets the sphere it is measured on. Cell geometry – which cell a coordinate lands in, where cell centres and corners sit, the hierarchy, the neighbours – is angular and identical on every body; the radius sets the kilometer figures: cell area, diagonal, spacing, and the resolution that area_km2 picks. Earth's area comes from the 'WGS84' ellipsoid, every other radius gives the sphere area $4\pi r^2$.

```
mars <- hex_grid(area_km2 = 1000, radius_km = "mars")
hex_grid(resolution = 8, radius_km = 3389.5) # the same grid
```

Both backends take a radius. 'H3' reports a cell's area as its solid angle times Earth's radius squared, so another radius scales those areas by the square of the radius ratio, exactly. One caveat carries: an 'H3' cell ID names a position in 'H3's topology, which 'Uber's 'H3' reads on Earth, so the IDs of a grid on another body are that topology on that body and are not interchangeable with Earth 'H3' data.

One Grid, Many Datasets

A HexGridInfo acts as a shared spatial reference system - like a CRS, but discrete and equal-area. Define the grid once, then attach multiple datasets without repeating parameters:

```
# Step 1: Define the grid once
grid <- hex_grid(area_km2 = 1000)

# Step 2: Attach multiple datasets to the same grid
birds <- hexify(bird_obs, lon = "longitude", lat = "latitude", grid = grid)
mammals <- hexify(mammal_obs, lon = "lon", lat = "lat", grid = grid)
climate <- hexify(weather_stations, lon = "x", lat = "y", grid = grid)

# No aperture, resolution, or area needed after step 1 - the grid
# travels with the data.

# Step 3: Work at the cell level
# Once hexified, lon/lat no longer matter - cell_id is the shared key
bird_counts <- aggregate(species ~ cell_id, data = as.data.frame(birds), length)
mammal_richness <- aggregate(species ~ cell_id, data = as.data.frame(mammals),
                             function(x) length(unique(x)))

# Join datasets by cell_id - guaranteed to align because same grid
combined <- merge(bird_counts, mammal_richness, by = "cell_id")

# Step 4: Visual confirmation
# All datasets produce identical grid overlays
plot(birds) # See the grid
plot(mammals) # Same grid, different data
```

See Also

[hexify](#) for assigning points to cells, [HexGridInfo-class](#) for class documentation

Examples

```
# Create grid by target area
grid <- hex_grid(area_km2 = 1000)
print(grid)

# Create grid by resolution
grid <- hex_grid(resolution = 8, aperture = 3)

# Create grid with different aperture
grid4 <- hex_grid(area_km2 = 500, aperture = 4)

# Create mixed aperture grid
grid43 <- hex_grid(area_km2 = 1000, aperture = "4/3")

# Mix in aperture 7, either as a family or level by level
grid47 <- hex_grid(area_km2 = 1000, aperture = "4/7")
```

```

grid_seq <- hex_grid(resolution = 4, aperture = c(4, 4, 7, 3))

# Grid on another body, by name or by radius
mars <- hex_grid(area_km2 = 1000, radius_km = "mars")
titan <- hex_grid(resolution = 6, radius_km = 2574.76)

# Use grid in hexify
df <- data.frame(lon = c(0, 10, 20), lat = c(45, 50, 55))
result <- hexify(df, lon = "lon", lat = "lat", grid = grid)

```

hex_summarize

Summarize Data by Hex Cell

Description

Aggregates data within each hexagonal cell, similar to `dplyr::group_by(cell_id) |> summarize(...)`. Returns a data.frame with one row per unique cell, including cell center coordinates and area.

Usage

```
hex_summarize(hex_data, ..., .fns = NULL, geometry = FALSE)
```

Arguments

<code>hex_data</code>	A HexData object (from <code>hexify()</code>).
<code>...</code>	Named summary expressions (tidyeval). Each expression is evaluated per cell group with the group's columns available by name. Examples: <code>mean_temp = mean(temperature)</code> , <code>n_species = length(unique(species))</code> , <code>max_elev = max(elevation, na.rm = TRUE)</code> . Provide either <code>...</code> or <code>.fns</code> , not both.
<code>.fns</code>	Optional named list of functions for formula-style aggregation. Example: <code>.fns = list(mean_temp = ~mean(temperature))</code> . Provide either <code>...</code> or <code>.fns</code> , not both.
<code>geometry</code>	Logical. If TRUE, attach cell center points as an sf geometry column (requires sf). Default FALSE.

Details

The function works entirely in R (no C++ needed). It groups by `cell_id` and evaluates the summary expressions within each group.

If no summary expressions are provided, returns cell counts only.

Value

A data.frame with columns:

cell_id Unique cell identifier

cell_cen_lon, cell_cen_lat Cell center coordinates

cell_area_km2 Cell area in km²

n_points Number of data points in this cell

... User-defined summary columns

If geometry = TRUE, returns an sf object with POINT geometry.

See Also

[hexify\(\)](#) for creating HexData objects, [get_neighbors\(\)](#) for finding neighboring cells

Examples

```
df <- data.frame(
  lon = runif(100, -10, 10),
  lat = runif(100, 40, 55),
  temperature = rnorm(100, 15, 5),
  species = sample(letters[1:5], 100, replace = TRUE)
)
hd <- hexify(df, lon = "lon", lat = "lat", area_km2 = 500)

# Count points per cell
hex_summarize(hd)

# Custom summaries
hex_summarize(hd, mean_temp = mean(temperature),
              n_species = length(unique(species)))
```

hex_uncompact

Uncompact Hex Cells

Description

Expands compacted cells to a target resolution. All cells in the output share the same resolution.

Usage

```
hex_uncompact(cell_ids, grid, target_resolution)
```

Arguments

cell_ids Character vector of (possibly mixed-resolution) cell IDs.

grid A HexGridInfo object specifying the grid.

target_resolution Integer. The resolution to expand all cells to.

Details

H3 backend: Uses the vendored H3 uncompactCells function.

ISEA backend (aperture 7, Z7 index): Appends digits 0-6 to expand each cell to its 7 children, repeating until the target resolution is reached.

Value

A character vector of cell IDs, all at target_resolution.

See Also

[hex_compact\(\)](#) for the inverse operation

Examples

```
g <- hex_grid(resolution = 3, type = "h3")
parent <- "832830fffffffff"
hex_uncompact(parent, g, target_resolution = 4L)
```

hex_zonal	<i>Zonal Statistics for Hex Cells</i>
-----------	---------------------------------------

Description

Computes zonal statistics by aggregating all raster pixels falling within each hexagonal cell polygon. More accurate than [hex_extract\(\)](#) but slower because it requires polygon geometries.

Usage

```
hex_zonal(raster, grid, fun = "mean", boundary = NULL, cells = NULL)
```

Arguments

- raster A terra::SpatRaster object.
- grid A HexGridInfo or HexData object specifying the grid.
- fun Summary function name: "mean" (default), "sum", "min", "max", "sd", or "count".
- boundary Optional sf polygon to limit the analysis extent.
- cells Optional cell IDs. If provided, only these cells are included.

Details

Requires the terra package (in Suggests). The function:

1. Generates hex polygon geometries via `cell_to_sf()`
2. Calls `terra::extract(raster, polygons, fun = fun)`
3. Joins results back to cell IDs

For point-based extraction (faster, at cell centers only), use `hex_extract()` instead.

Value

A data.frame with columns `cell_id`, plus one column per raster layer containing the aggregated values.

See Also

`hex_extract()` for cell-center extraction, `cell_to_sf()` for hex polygon generation

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  r <- terra::rast(nrows = 100, ncols = 100,
                  xmin = -10, xmax = 10, ymin = 40, ymax = 55)
  terra::values(r) <- runif(10000)
  names(r) <- "temperature"

  g <- hex_grid(area_km2 = 500)
  df <- data.frame(lon = c(0, 5), lat = c(45, 50))
  hd <- hexify(df, lon = "lon", lat = "lat", grid = g)
  hex_zonal(r, hd, fun = "mean")
}
```

import_h3

Import External H3 Cell IDs into hexify

Description

Ingests H3 cell IDs from an external source (another H3 library, a database, or a CSV file) into hexify. Validates cell IDs, infers the H3 resolution, and optionally attaches data to build a HexData object.

Usage

```
import_h3(cell_ids, data = NULL, validate = TRUE, radius_km = EARTH_RADIUS_KM)
```


Arguments

cell_ids	Character vector of H3 cell ID strings
data	Optional data frame to attach. Must have the same number of rows as cell_ids. If NULL, returns a HexGridInfo object.
validate	If TRUE (default), checks that all cell IDs are valid H3 cells at the same resolution before proceeding. Set to FALSE to skip validation when cell IDs are known to be correct.
radius_km	Radius of the body the cells cover, in kilometers, or a body name (default Earth). See hex_grid .

Details

For converting between grid specs, use [hex_grid](#)(type = "h3") directly. For cell-level ISEA/H3 mapping, use [h3_crosswalk](#).

H3 cell IDs encode their resolution in the index itself, so no resolution argument is needed. The resolution is inferred automatically. All cell IDs must share the same resolution; mixed resolutions produce an error.

Value

If data = NULL, a HexGridInfo object for the inferred H3 resolution. If data is provided, a HexData object with data attached at the specified cells.

See Also

[hex_grid](#) for creating grids directly, [h3_crosswalk](#) for cell-level ISEA/H3 mapping

Examples

```
# Import external H3 cell IDs (grid spec only)
h3_ids <- c("8528342bffffffff", "85283473ffffffff", "85283447ffffffff")
grid <- import_h3(h3_ids)
grid

# Import with data attached
df <- data.frame(species = c("oak", "pine", "birch"), count = c(10, 5, 3))
hd <- import_h3(h3_ids, data = df)
hd
```

is_hex_data	<i>Check if object is HexData</i>
-------------	-----------------------------------

Description

Check if object is HexData

Usage

is_hex_data(x)

Arguments

x Object to check

Value

Logical

is_hex_grid	<i>Check if object is HexGridInfo</i>
-------------	---------------------------------------

Description

Check if object is HexGridInfo

Usage

is_hex_grid(x)

Arguments

x Object to check

Value

Logical

is_pentagon	<i>Detect Pentagon Cells</i>
-------------	------------------------------

Description

Identifies which cells are pentagons. Any hexagonal tiling of the sphere must contain exactly 12 pentagons (at the icosahedron vertices). Pentagon cells have 5 neighbors instead of 6.

Usage

is_pentagon(cell_id, grid)

Arguments

cell_id Cell IDs to check. Numeric for ISEA, character for H3.
grid A HexGridInfo or HexData object specifying the grid.

Details

H3 backend: Uses the vendored H3 `isPentagon` function.

ISEA backend: The 12 pentagons are located at icosahedron vertices, which are always the (i, j) = (0, 0) cell of their quad. Pentagon status is checked by decoding each input cell ID's own (quad, i, j) coordinates via `cell_to_lonlat()`'s underlying grid math and testing whether i and j are both zero, rather than by re-deriving each vertex's cell ID (the forward direction has aperture-specific quirks – e.g. aperture 7's substrate/surrogate coordinate distinction – that make a single fixed formula for "the (0,0) cell ID of quad Q" unreliable across apertures).

Value

A logical vector. TRUE for pentagon cells, FALSE for hexagons.

See Also

`get_neighbors()` for neighbor finding (pentagons have 5 neighbors)

Examples

```
# H3 pentagon detection
g <- hex_grid(resolution = 1, type = "h3")
cells <- grid_global(g)
pent <- is_pentagon(cells$cell_id, g)
sum(pent) # Should be 12
```

lonlat_to_cell	<i>Convert longitude/latitude to cell ID</i>
----------------	--

Description

Converts geographic coordinates to DGGS cell IDs using a grid specification.

Usage

```
lonlat_to_cell(lon, lat, grid)
```

Arguments

lon	Numeric vector of longitudes in degrees
lat	Numeric vector of latitudes in degrees
grid	A HexGridInfo or HexData object, or legacy hexify_grid

Details

This function accepts either a HexGridInfo object from `hex_grid()` or a HexData object from `hexify()`. If a HexData object is provided, its grid specification is extracted automatically.

Value

Numeric vector of cell IDs

See Also

[cell_to_lonlat](#) for the inverse operation, [hex_grid](#) for creating grid specifications

Examples

```
grid <- hex_grid(area_km2 = 1000)
cells <- lonlat_to_cell(lon = c(0, 10), lat = c(45, 50), grid = grid)

# Or use HexData object
df <- data.frame(lon = c(0, 10, 20), lat = c(45, 50, 55))
result <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000)
cells <- lonlat_to_cell(lon = 5, lat = 48, grid = result)
```

n_cells

Get Number of Cells

Description

Get the number of unique cells in a HexData object.

Usage

```
n_cells(x)
```

Arguments

x A HexData object

Value

Integer count of unique cells

```
plot, HexData, missing-method
```

Plot HexData objects

Description

Default plot method for HexData objects. Draws hexagonal grid cells with an optional basemap.

Usage

```
## S4 method for signature 'HexData,missing'
plot(
  x,
  y,
  basemap = TRUE,
  clip_basemap = TRUE,
  basemap_fill = "gray90",
  basemap_border = "gray50",
  basemap_lwd = 0.5,
  grid_fill = "#E69F00",
  grid_border = "#5D4E37",
  grid_lwd = 0.8,
  grid_alpha = 0.7,
  fill = NULL,
  show_points = FALSE,
  point_size = "auto",
  point_color = "red",
  point_alpha = 1,
  jitter = TRUE,
  crop = TRUE,
  crop_expand = 0.1,
  main = NULL,
  ...
)
```

Arguments

<code>x</code>	A HexData object from <code>hexify()</code>
<code>y</code>	Ignored (for S4 method compatibility)
<code>basemap</code>	Basemap specification: • TRUE or "world": Use built-in world map • FALSE or NULL: No basemap • sf object: Custom basemap
<code>clip_basemap</code>	Clip basemap to data extent (default TRUE). Clipping temporarily disables S2 spherical geometry to avoid edge-crossing errors.

<code>basemap_fill</code>	Fill color for basemap (default "gray90")
<code>basemap_border</code>	Border color for basemap (default "gray50")
<code>basemap_lwd</code>	Line width for basemap borders (default 0.5)
<code>grid_fill</code>	Fill color for grid cells (default "#E69F00" - amber/orange)
<code>grid_border</code>	Border color for grid cells (default "#5D4E37" - dark brown)
<code>grid_lwd</code>	Line width for cell borders (default 0.8)
<code>grid_alpha</code>	Transparency for cell fill (0-1, default 0.7)
<code>fill</code>	Column name for fill mapping (optional)
<code>show_points</code>	Show original points on top of cells (default FALSE). Points are jittered within their assigned hexagon.
<code>point_size</code>	Size of points. Can be: • A number (direct cex value) • A preset defining what fraction of a hex cell one point covers: "tiny" (~2\
<code>point_color</code>	Color of points (default "red")
<code>point_alpha</code>	Transparency for points (0-1, default 1)
<code>jitter</code>	If TRUE (default), points are randomly scattered within their assigned hexagon. If FALSE, all points in a cell are plotted at the cell centroid instead.
<code>crop</code>	Crop to data extent (default TRUE)
<code>crop_expand</code>	Expansion factor for crop (default 0.1)
<code>main</code>	Plot title
<code>...</code>	Additional arguments passed to base plot()

Details

This function generates polygon geometries for the cells present in the data and plots them. Polygons are computed on demand, not stored, to minimize memory usage.

Value

Invisibly returns the HexData object

See Also

[hexify_heatmap](#) for ggplot2 plotting

Examples

```
df <- data.frame(lon = runif(50, -5, 5), lat = runif(50, 45, 50))
result <- hexify(df, lon = "lon", lat = "lat", area_km2 = 2000)

# Basic plot
plot(result, basemap = FALSE)

# With basemap and custom styling
plot(result, grid_fill = "lightblue", grid_border = "darkblue")
```

plot_globe

*Plot hexagonized globe***Description**

Renders a global hexagonal grid on an orthographic projection with customizable rotation, land clipping, and styling options.

Usage

```
plot_globe(
  area = 50000,
  center = "europe",
  clip_to_land = FALSE,
  land_data = NULL,
  exclude_antarctica = TRUE,
  fill = "#D4B896",
  border = "grey30",
  border_width = 0.2,
  ocean_fill = "white",
  ocean_border = "grey50",
  show_land = clip_to_land,
  land_fill = NA,
  land_border = "grey40",
  land_width = 0.3,
  use_ggplot = NULL,
  return_data = FALSE,
  aperture = 3L
)
```

Arguments

area	Cell area in km ² (passed to hex_grid)
center	Globe center: either a preset name (e.g., "europe") or numeric vector c(lon, lat). See globe_centers for presets.
clip_to_land	If TRUE, clip hexagons to land boundaries
land_data	Optional sf object for land boundaries. If NULL and clip_to_land is TRUE, uses <code>rnaturalearth::ne_countries()</code>
exclude_antarctica	If TRUE, exclude Antarctica from land clipping
fill	Fill color for hexagons (default "#D4B896")
border	Border color for hexagons (default "grey30")
border_width	Border width for hexagons (default 0.2)
ocean_fill	Fill color for ocean/globe background (default "white")

ocean_border	Border color for globe circle (default "grey50")
show_land	If TRUE, show land boundaries (default TRUE when clipping)
land_fill	Fill color for land (default NA, transparent)
land_border	Border color for land boundaries (default "grey40")
land_width	Border width for land boundaries (default 0.3)
use_ggplot	NULL = auto-detect, TRUE = force ggplot2, FALSE = force base
return_data	If TRUE, return sf objects instead of plotting
aperture	Grid aperture (default 3L)

Details

The function handles several technical challenges:

- Hexagons on the back side of the globe fail to transform - these are filtered out gracefully
- Invalid geometries after projection are repaired with `st_buffer(0)`
- Clipping is done in orthographic CRS to avoid topology errors

Value

If `use_ggplot = TRUE`: ggplot2 object (can add layers with `+`) If `use_ggplot = FALSE`: NULL invisibly (plots directly) If `return_data = TRUE`: list of sf objects (hexagons, land, ocean_circle, crs)

See Also

[globe_centers](#) for available presets, [grid_global](#) for generating global grids without plotting

Examples

```
# Get data for custom plotting (no rendering)
data <- plot_globe(area = 100000, center = "europe", return_data = TRUE)
nrow(data$hexagons)
class(data$ocean_circle)

# Basic usage - Europe-centered globe
plot_globe(area = 80000, center = "europe")
```


plot_grid

*Plot hexagonal grid clipped to a polygon boundary***Description**

A convenience function that creates a grid, clips it to a boundary polygon, and plots the result in a single call.

Usage

```
plot_grid(
  boundary,
  grid,
  crop = TRUE,
  boundary_fill = "gray95",
  boundary_border = "gray40",
  boundary_lwd = 0.5,
  grid_fill = "steelblue",
  grid_border = "steelblue",
  grid_lwd = 0.3,
  grid_alpha = 0.3,
  title = NULL,
  expand = 0.05
)
```

Arguments

boundary	An sf/sfc polygon to clip to (e.g., country boundary)
grid	A HexGridInfo object from hex_grid()
crop	If TRUE (default), cells are cropped to boundary. If FALSE, only complete hexagons within boundary are shown.
boundary_fill	Fill color for the boundary polygon (default "gray95")
boundary_border	Border color for boundary (default "gray40")
boundary_lwd	Line width for boundary (default 0.5)
grid_fill	Fill color for grid cells (default "steelblue")
grid_border	Border color for grid cells (default "steelblue")
grid_lwd	Line width for cell borders (default 0.3)
grid_alpha	Transparency for cell fill (0-1, default 0.3)
title	Plot title. If NULL (default), auto-generates title with cell area.
expand	Expansion factor for plot limits (default 0.05)

Details

This is a convenience wrapper around `grid_clip()` that handles the common use case of visualizing a hexagonal grid over a geographic region.

Value

A ggplot object that can be further customized

See Also

[grid_clip](#) for the underlying clipping function, [hex_grid](#) for grid specification

Examples

```
# Plot grid over France
france <- hexify_world[hexify_world$name == "France", ]
grid <- hex_grid(area_km2 = 2000)
plot_grid(france, grid)

# Customize colors
plot_grid(france, grid,
          grid_fill = "coral", grid_alpha = 0.5,
          boundary_fill = "lightyellow")

# Keep only complete hexagons
plot_grid(france, grid, crop = FALSE)

# Add ggplot2 customizations
library(ggplot2)
plot_grid(france, grid) +
  labs(subtitle = "ISEA3H Discrete Global Grid") +
  theme_void()
```

plot_world

Quick world map plot

Description

Simple wrapper to plot the built-in world map.

Usage

```
plot_world(fill = "gray90", border = "gray50", ...)
```

Arguments

fill	Fill color for countries
border	Border color for countries
...	Additional arguments passed to plot()

Value

NULL invisibly. Creates a plot as side effect.

See Also

Other visualization: [hexify_heatmap\(\)](#)

Examples

```
# Quick world map
plot_world()

# Custom colors
plot_world(fill = "lightblue", border = "darkblue")
```

st_crs.HexGridInfo	<i>Coordinate reference system of a grid</i>
--------------------	--

Description

Reads the CRS a grid's coordinates are in, as `sf::st_crs()` does for any spatial object. An Earth grid returns 'WGS84'; a grid built on another body returns a longlat CRS on the sphere of its radius.

Usage

```
## S3 method for class 'HexGridInfo'
st_crs(x, ...)

## S3 method for class 'HexData'
st_crs(x, ...)
```

Arguments

x	A HexGridInfo or HexData object
...	Passed on to <code>sf::st_crs()</code>

Value

An object of class `crs`, as `sf::st_crs()` returns: a list carrying the reference system in 'PROJ' and 'WKT' form. It says how to read the coordinates that the grid's cells, centres and `sf` exports come back in.

Examples

```
st_crs(hex_grid(resolution = 5))
st_crs(hex_grid(resolution = 5, radius_km = "mars"))
```

Index

- * **'dggridR' compatibility**
 - as_dggrid, [4](#)
 - dggrid_is_compatible, [10](#)
 - from_dggrid, [11](#)
- * **coordinate conversion**
 - hexify_roundtrip_test, [34](#)
- * **datasets**
 - hexify_world, [37](#)
- * **grid statistics**
 - dgearthstat, [9](#)
 - hexify_compare_resolutions, [24](#)
- * **hexify main**
 - hexify, [20](#)
 - hexify_grid, [27](#)
- * **projection**
 - hexify_build_icosah, [23](#)
 - hexify_face_centers, [25](#)
 - hexify_forward, [25](#)
 - hexify_forward_to_face, [26](#)
 - hexify_get_precision, [27](#)
 - hexify_inverse, [32](#)
 - hexify_projection_stats, [33](#)
 - hexify_set_precision, [35](#)
 - hexify_set_verbose, [35](#)
 - hexify_which_face, [36](#)
- * **visualization**
 - hexify_heatmap, [29](#)
 - plot_world, [58](#)
- as_dggrid, [4](#)
- as_dggrid(), [10](#), [11](#)
- as_sf, [4](#), [8](#), [21](#)
- as_sf(), [38](#), [39](#)
- as_tibble.HexData, [5](#)
- cell_area, [6](#), [18](#)
- cell_to_lonlat, [7](#), [52](#)
- cell_to_lonlat(), [40](#), [41](#), [51](#)
- cell_to_sf, [8](#), [31](#)
- cell_to_sf(), [48](#)
- cells, [6](#)
- dg_closest_res_to_area(), [9](#), [24](#)
- dgearthstat, [9](#)
- dgearthstat(), [24](#)
- dggrid_43h_sequence(), [4](#), [10](#), [11](#)
- dggrid_is_compatible, [10](#)
- dggrid_is_compatible(), [4](#), [11](#)
- dgverify, [10](#)
- from_dggrid, [11](#)
- from_dggrid(), [4](#), [10](#)
- get_children(), [40](#)
- get_neighbors, [12](#)
- get_neighbors(), [41](#), [46](#), [51](#)
- get_parent(), [40](#)
- globe_centers, [13](#), [55](#), [56](#)
- grid_clip, [14](#), [58](#)
- grid_global, [14](#), [15](#), [17](#), [56](#)
- grid_info, [16](#)
- grid_rect, [14](#), [15](#), [16](#)
- h3_crosswalk, [7](#), [17](#), [49](#)
- hex_browse, [38](#)
- hex_compact, [39](#)
- hex_compact(), [47](#)
- hex_distance, [40](#)
- hex_distance(), [12](#)
- hex_extract, [41](#)
- hex_extract(), [47](#), [48](#)
- hex_grid, [7](#), [8](#), [18–21](#), [24](#), [28](#), [42](#), [49](#), [52](#), [55](#), [58](#)
- hex_summarize, [45](#)
- hex_uncompact, [46](#)
- hex_uncompact(), [40](#)
- hex_zonal, [47](#)
- hex_zonal(), [41](#), [42](#)
- HexData-class, [18](#)
- HexGridInfo-class, [19](#)

hexify, [19](#), [20](#), [28](#), [44](#)
 hexify(), [12](#), [28](#), [42](#), [45](#), [46](#)
 hexify-conversions, [22](#)
 hexify-grid, [22](#)
 hexify-package, [3](#)
 hexify-stats, [22](#)
 hexify_area_to_eff_res(), [9](#), [24](#)
 hexify_build_icosah, [23](#)
 hexify_build_icosah(), [25–27](#), [33](#), [35](#), [36](#)
 hexify_cell_id_to_quad_ij(), [34](#)
 hexify_cell_to_icosah_tri(), [34](#)
 hexify_cell_to_lonlat(), [34](#)
 hexify_cell_to_plane(), [34](#)
 hexify_cell_to_quad_ij(), [34](#)
 hexify_cell_to_quad_xy(), [34](#)
 hexify_compare_resolutions, [24](#)
 hexify_compare_resolutions(), [9](#)
 hexify_eff_res_to_area(), [9](#), [24](#)
 hexify_eff_res_to_resolution(), [9](#), [24](#)
 hexify_face_centers, [25](#)
 hexify_face_centers(), [23](#), [26](#), [27](#), [33](#), [35](#),
 [36](#)
 hexify_forward, [25](#)
 hexify_forward(), [23](#), [25](#), [27](#), [33](#), [35](#), [36](#)
 hexify_forward_to_face, [26](#)
 hexify_forward_to_face(), [23](#), [25–27](#), [33](#),
 [35](#), [36](#)
 hexify_get_precision, [27](#)
 hexify_get_precision(), [23](#), [25–27](#), [33](#), [35](#),
 [36](#)
 hexify_grid, [27](#)
 hexify_grid(), [21](#)
 hexify_grid_cell_to_lonlat(), [34](#)
 hexify_grid_to_cell, [28](#)
 hexify_grid_to_cell(), [34](#)
 hexify_heatmap, [29](#), [54](#)
 hexify_heatmap(), [39](#), [59](#)
 hexify_icosah_tri_to_plane(), [34](#)
 hexify_icosah_tri_to_quad_ij(), [34](#)
 hexify_icosah_tri_to_quad_xy(), [34](#)
 hexify_inverse, [32](#)
 hexify_inverse(), [23](#), [25–27](#), [33](#), [35](#), [36](#)
 hexify_lonlat_to_cell(), [34](#)
 hexify_lonlat_to_plane(), [34](#)
 hexify_lonlat_to_quad_ij(), [34](#)
 hexify_projection_stats, [33](#)
 hexify_projection_stats(), [23](#), [25–27](#), [33](#),
 [35](#), [36](#)
 hexify_quad_ij_to_cell(), [34](#)
 hexify_quad_ij_to_icosah_tri(), [34](#)
 hexify_quad_ij_to_xy(), [34](#)
 hexify_quad_xy_to_cell(), [34](#)
 hexify_quad_xy_to_icosah_tri(), [34](#)
 hexify_resolution_to_eff_res(), [9](#), [24](#)
 hexify_roundtrip_test, [34](#)
 hexify_set_precision, [35](#)
 hexify_set_precision(), [23](#), [25–27](#), [33](#), [36](#)
 hexify_set_verbose, [35](#)
 hexify_set_verbose(), [23](#), [25–27](#), [33](#), [35](#),
 [36](#)
 hexify_which_face, [36](#)
 hexify_which_face(), [23](#), [25–27](#), [33](#), [35](#), [36](#)
 hexify_world, [37](#)

 import_h3, [48](#)
 is_hex_data, [49](#)
 is_hex_grid, [50](#)
 is_pentagon, [50](#)

 lonlat_to_cell, [7](#), [51](#)

 n_cells, [52](#)

 plot, HexData, missing-method, [53](#)
 plot_globe, [13](#), [55](#)
 plot_grid, [31](#), [57](#)
 plot_world, [58](#)
 plot_world(), [31](#)

 st_crs.HexData(st_crs.HexGridInfo), [59](#)
 st_crs.HexGridInfo, [59](#)