Package 'hexify'

November 7, 2026

Title Equal-Area Hex Grids on the Snyder ISEA Icosahedron

Version 0.9.1

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>) and the 'H3' hierarchical hexagonal system (Uber Technologies). Implements the 'ISEA' discrete global grid system (Sahr, White and Kimerling 2003 <doi:10.1559/152304003100011090>) and the 'Hex9' octahedral grid (Griffin 2026 <doi:10.48550/arXiv.2608.00022>). 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), chromote, hexglobe, knitr, rmarkdown, terra, raster, ggplot2, RColorBrewer, rnaturalearth, tibble, gridExtra, htmlwidgets, jsonlite, leaflet

VignetteBuilder knitr

LinkingTo Rcpp

Imports sf, Rcpp, methods, rlang, bit64

Additional_repositories https://gcol33.r-universe.dev

URL https://gillescolling.com/hexify/

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

Config/testthat/edition 3

Depends R (>= 4.0)

LazyData true

Config/roxygen2/version 8.1.0

Collate 'RcppExports.R' 'aperture_mixed_hierarchy.R' 'aperture_sequence.R' 'cell_id.R' 'cell_metrics.R' 'constants.R' 'classes.R' 'construction.R' 'constructors.R' 'crosswalk.R' 'data.R' 'dggrid.R' 'dggrid_compat.R' 'dggrs_definition.R' 'distortion.R' 'gosper.R' 'grid_helpers.R' 'h3_compat.R' 'hex9_warp.R' 'hex_aggregate.R' 'hex_browse.R' 'hex_compact.R' 'hex_distance.R' 'hex_extract.R' 'hex_globe.R' 'hex_glyphs.R' 'hex_sample.R' 'hex_smooth.R' 'hex_summarize.R' 'hex_topology.R' 'hex_zonal.R' 'hexify-package.R' 'hexify.R' 'hexify_assign.R' 'hexify_cell_to_sf.R' 'hexify_conversions.R' 'hexify_grid.R' 'hexify_index.R' 'hexify_map.R' 'hexify_neighbors.R' 'hexify_projection.R' 'hexify_stats.R' 'net_cells.R' 'net_layout.R' 'ogc_zirs.R' 'orientation.R' 'plot_methods.R' 'plot_surface.R' 'sphere_geometry.R'

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)), Ben Griffin [ctb, cph] (Hex9 address tables in src/hex9.cpp, from libhex9 (Apache-2.0))

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

Repository CRAN

Date/Publication 2026-11-07 22:50:11 UTC

Contents

hexify-package . . . 3
as_dggrid . . . 4
cells . . . 5
cell_area . . . 6
cell_metrics . . . 7
cell_to_lonlat . . . 8
cell_to_sf . . . 9
dgearthstat . . . 11
dggrid_is_compatible . . . 12
dggrs_definition . . . 13
from_dggrid . . . 14
get_neighbors . . . 15
globe_centers . . . 16
grid_clip . . . 17
grid_global . . . 18
grid_info . . . 19
grid_rect . . . 20
h3_crosswalk . . . 21
hex9_warp_download . . . 22
HexData-class . . . 23

HexGridInfo-class 28
hexify 34
hexify_heatmap 37
hexify_world 40
hex_aggregate 42
hex_browse 44
hex_compact 46
hex_distance 47
hex_extract 48
hex_globe 49
hex_globe_png 53
hex_grid 54
hex_rays 60
hex_sample 62
hex_smooth 63
hex_summarize 64
hex_triangles 65
hex_uncompact 66
hex_zonal 67
import_h3 68
is_hex_data 69
is_hex_grid 70
is_pentagon 71
lonlat_to_cell 72
net_cells 73
net_layout 74
net_project 76
n_cells 77
plot_grid 77
plot_world 79
projection_distortion 80
projection_stages 81
wall_metrics 83

Index **85**

`hexify-package` *hexify: Equal-Area Hex Grids on the Snyder ISEA Icosahedron*

Description

Provides functions to build and use hexagonal discrete global grids using the Icosahedral Snyder Equal Area ('ISEA') projection (Snyder 1992 doi:10.3138/27H78K8848821752) and the 'H3' hierarchical hexagonal system (Uber Technologies). Implements the 'ISEA' discrete global grid system (Sahr, White and Kimerling 2003 doi:10.1559/152304003100011090) and the 'Hex9' octahedral grid (Griffin 2026 doi:10.48550/arXiv.2608.00022). 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.

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]
- Ben Griffin (Hex9 address tables in src/hex9.cpp, from libhex9 (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` *Convert hexify Grid to 'dggridR'-compatible Grid Object*

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)

## S3 method for class 'dggs'
format(x, ...)

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

Arguments

`grid`	A grid from `hex_grid`, or a legacy hexify_grid
`x`	A dggs object
`...`	Ignored

Details

A dggs carries no body radius, so a grid built on another body cannot be expressed as one: 'dggridR' reads every dggs on Earth's radius, and its areas, spacings and CLS come back on Earth. Converting such a grid warns.

Value

A list with 'dggridR'-compatible fields:

`pole_lon_deg`	Longitude of icosahedron vertex 0 (standard 11.25)
`pole_lat_deg`	Latitude of icosahedron vertex 0 (standard 58.282525588538995)
`azimuth_deg`	Azimuth of vertex 1 seen from vertex 0 (standard 0)
`aperture`	Grid aperture (3, 4, or 7)
`res`	Resolution level
`topology`	Grid topology ("HEXAGON")
`projection`	Face projection ('ISEA' or 'FULLER')
`precision`	Output decimal precision (default 7)

It prints as one line naming the grid in DGGRID's terms.

See Also

Other 'dggridR' compatibility: `dggrid_43h_sequence()`, `dggrid_is_compatible()`, `from_dggrid()`

Examples

```
dggs <- as_dggrid(hex_grid(resolution = 5, aperture = 3))
dggs
dggrid_is_compatible(dggs)
from_dggrid(dggs)
```

`cells` *Get Cell IDs*

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

Examples

```
df <- data.frame(lon = c(0, 0.01, 20), lat = c(45, 45, 55))
cells(hexify(df, lon = "lon", lat = "lat", area_km2 = 1000))
```

---

cell_area *Compute Per-Cell Area in km²*

Description

Returns the area of each cell in square kilometers. On the ISEA and IVEA projections every hexagon of a resolution has the same area, the 12 pentagons of the icosahedron 5/6 of it, the 6 squares of the octahedron 4/6 of it and the 4 triangles of the tetrahedron half of it. On the Fuller projection and for H3 grids, the area varies from cell to cell.

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

On the ISEA and IVEA projections, both equal-area, every hexagon of a resolution has the same area. A vertex cell, centred on a vertex of the solid where k faces meet, covers k of a hexagon's six sixths, so a grid of N cells on a body of area S has hexagons of area $S/(N-2)$, pentagons of $(5/6)S/(N-2)$ on the icosahedron, squares of $(4/6)S/(N-2)$ on the octahedron and triangles of $(3/6)S/(N-2)$ on the tetrahedron. The grid's `area_km2` is the mean, S/N.

On the Fuller projection (`hex_grid(projection = "fuller")`) cells are not equal-area. Each cell's area is its solid angle, summed over its true boundary, times the body's area over 4π, so the cells of a grid add up to the body's area.

For H3 grids the area varies from cell to cell, by about a factor of 2 across the hexagons of a resolution. 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: one area for every hexagon
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_metrics *Shape Metrics of Cells: Area, Perimeter and Compactness*

Description

Measures each cell on the sphere: its area, the length of its boundary and how close its shape is to a circle.

Usage

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

Arguments

cell_id	Cell IDs. When NULL, the cells of a HexData grid, or every cell of a HexGridInfo grid.
grid	A HexGridInfo or HexData object.

Details

The perimeter follows each wall as the grid draws it. An ISEA wall is straight on its face of the solid and curved on the sphere, so it is followed on the sphere until its length is exact to about 1e-8. An H3 wall is a great-circle arc between corners, with an extra corner where it crosses an edge of the icosahedron.

Compactness (White et al. 1998) is the perimeter of a spherical cap with the cell's area over the cell's perimeter, $\sqrt{4\pi a - a^2/r^2}/p$ for area a, perimeter p and radius r. A circle scores 1; a small regular hexagon 0.952, pentagon 0.930, square 0.886 and triangle 0.778.

The normalized area is the cell's area over the mean area of the grid's cells, the body's surface over the cell count; an equal-area grid of N cells gives $N/(N-2)$ for its hexagons, 5/6 of that for the pentagons of the icosahedron, 4/6 for the squares of the octahedron and 3/6 for the triangles of the

tetrahedron. The isoperimetric quotient `ipq` is $4\pi a/p^2$, for a circle 1, a small regular hexagon 0.907, square 0.785 and triangle 0.605. Kmoch et al. (2022) compare grids by these two measures, reading area and perimeter in an equal-area plane centred on each cell; here both are read on the sphere, which adds $(a/(rp))^2$ to `compactness^2`, a few parts in a million for cells of a few thousand square kilometres.

Value

Data frame with one row per `cell_id`: `cell_id`, `area_km2` (as `cell_area` gives it), `normalized_area`, `perimeter_km`, `compactness` and `ipq`.

References

White, D., Kimerling, A. J., Sahr, K. and Song, L. (1998). Comparing area and shape distortion on polyhedral-based recursive partitions of the sphere. International Journal of Geographical Information Science 12(8), 805-827. doi:10.1080/136588198241518

Kmoch, A., Vasilyev, I., Virro, H. and Uuemaa, E. (2022). Area and shape distortions in open-source discrete global grid systems. Big Earth Data 6(3), 256-275. doi:10.1080/20964471.2022.2094926

See Also

`wall_metrics` for the spacing of neighbouring cells, `cell_area`

Examples

```
g <- hex_grid(resolution = 3, aperture = 3)
m <- cell_metrics(grid = g)
summary(m$compactness)

# Fuller's projection is not equal-area: compare the spread of areas
gf <- hex_grid(resolution = 3, aperture = 3, projection = "fuller")
mf <- cell_metrics(grid = gf)
sd(mf$area_km2) / mean(mf$area_km2)
```

`cell_to_lonlat` *Convert Cell ID to Longitude/Latitude*

Description

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

Usage

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

Arguments

cell_id	Cell IDs: integer64 for ISEA grids (whole numbers below 2^53 and digit strings are accepted), character for H3 grids
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 *Convert Cell IDs to sf Polygons*

Description

Creates sf polygon geometries for hexagonal grid cells.

Usage

```
cell_to_sf(
  cell_id = NULL,
  grid,
  wrap_dateline = TRUE,
  densify = NULL,
  max_km = NULL,
  shape = c("hexagon", "gosper", "descendants"),
  depth = 3L
)
```

Arguments

cell_id	Cell IDs (integer64 for ISEA, character for H3). 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 `sf::st_wrap_dateline()` to split antimeridian-crossing polygons, which flat maps and planar (GEOS) operations need. Set to FALSE for orthographic/globe projections where wrapping creates gaps, and for spherical (s2) operations: each cell then stays one polygon whose corners lie in -180..180, so a cell crossing the antimeridian spans a flat lon/lat map.

`densify` Points added along each cell edge: every edge is halved until the straight line in longitude and latitude between consecutive points stays within `densify` times its length of the true edge. `0` keeps the corners alone. `NULL` uses 0.001 for ISEA cells and 0 for H3 cells.

`max_km` Longest edge piece, in km on the grid's sphere: edges are halved further until no straight piece between consecutive points spans more than `max_km`, on top of `densify`. `NULL` (default) sets no limit. The bound `densify` sets is relative to each piece's length, so a long, gently curved edge can stand as one piece that strays hundreds of metres from the true edge; `max_km` bounds that in distance.

`shape` `"hexagon"` (default) draws each cell's own boundary. `"gosper"` draws it as a Gosper island and `"descendants"` as the outline of its descendants; both need an ISEA grid, and `"descendants"` one of aperture 7. See Details.

`depth` For `shape = "gosper"`, the number of times each edge is replaced, multiplying its segments by 3; for `"descendants"`, the number of resolutions down, multiplying the cells behind one outline by 7.

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.

An ISEA cell edge is straight on its icosahedron face and curved in longitude and latitude, so ISEA polygons are densified by default. An H3 cell edge is a great-circle arc between corners, which sf reads exactly from the corners alone when it uses s2; densify H3 cells for planar work, or to draw them on a flat map.

A Gosper island replaces every cell edge by three segments $1/\sqrt{7}$ as long, turned by $\arctan(\sqrt{3}/5)$, and repeats this `depth` times; its outline approaches the flowsnake. The replacement is drawn on the face plane of the projection and moves as much area out of a cell as into it, so on an equal-area grid an island has exactly its cell's area, and neighbouring islands share their boundary, so the islands of a grid tile the sphere. The twelve pentagonal cells give five-sided islands.

The descendant outline of an aperture-7 cell is the boundary of its descendants `depth` resolutions down. It covers exactly the cell's area, the outlines of one resolution tile the sphere, and an outline is the union of its children's outlines, so they nest across resolutions. The child lattice of an aperture-7 ISEA grid turns one way at one resolution and back at the next, so the outline stays close to the hexagon rather than approaching a Gosper island.

Either shape is drawn around the cell's hexagon: a point near the boundary can lie in one cell's shape and be assigned to the neighbouring cell by `lonlat_to_cell`.

Value

sf object with cell_id and geometry columns

See Also

hex_grid for grid specifications, st_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)

# Cells drawn as Gosper islands, and as the outline of their descendants
g7 <- hex_grid(resolution = 3, aperture = 7)
cells <- lonlat_to_cell(c(0, 10), c(45, 50), g7)
islands <- cell_to_sf(cells, g7, shape = "gosper", depth = 3)
outlines <- cell_to_sf(cells, g7, shape = "descendants", depth = 2)
```

---

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:

area_km	Total surface area of the body in km^2
n_cells	Total number of cells at this resolution
cell_area_km2	Average cell area in km^2
cell_spacing_km	Average distance between cell centers in km
resolution	Resolution level
aperture	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 *Validate 'dggridR' Grid Compatibility with hexify*

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()

Examples

```
dggrid_is_compatible(as_dggrid(hex_grid(resolution = 4, aperture = 7)))
dggrid_is_compatible(list(projection = "ISEA", topology = "DIAMOND",
                          aperture = 4, res = 4), strict = FALSE)
```

`dggrs_definition` *Definition of a Grid as a Discrete Global Grid Reference System*

Description

Describes a grid in the DGGRS definition schema of OGC API - Discrete Global Grid Systems: the hierarchy of grids (solid, refinement ratio, refinement strategy, zone shapes, orientation), the zone identifiers and the order of sub-zones.

Usage

```
dggrs_definition(grid)
```

Arguments

`grid`	A HexGridInfo or HexData object.

Details

The definition describes the grid as hexify builds it. The zone identifiers are the strings `cell_to_index` writes, which carry their resolution and are unique across all resolutions, and the cell IDs, which number the cells of one resolution from 1 and are unique only together with it.

The ISEA3H, ISEA7H, IVEA3H and IVEA7H definitions registered with OGC (https://www.opengis.net/def/dggrs/OGC/1.0/ISEA7H) convert WGS84 geodetic latitude to authalic latitude before projecting, and place the icosahedron's first vertex at 11.20 degrees E and authalic latitude $\arctan(\phi)$: `hex_grid(aperture = 3, ellipsoid = "WGS84",orientation = "ogc")`, its aperture-7 twin and the same with `projection = "ivea"`. For such a grid the definition carries OGC's zone identifiers (`cell_to_index(form = "textZIRS")`) and links to the OGC definition, which it matches in its hierarchy and zone identifiers; it carries no OGC URI, since hexify lists sub-zones in ascending order of cell ID rather than in OGC's scanlines. On a grid with an ellipsoid the orientation's latitude is geodetic, as OGC states it.

Value

A named list with the schema's fields `title`, `description`, `dggh`, `zirs` and `subZoneOrder`. `jsonlite::toJSON(x, auto_unbox = TRUE, pretty = TRUE)` writes it as JSON.

References

Open Geospatial Consortium. OGC API - Discrete Global Grid Systems - Part 1: Core. OGC 21-038r1. https://docs.ogc.org/is/21-038r1/21-038r1.html

See Also

`hex_grid`, `cell_to_index`

Examples

```
def <- dggrs_definition(hex_grid(resolution = 5, aperture = 7))
def$dggh$definition$refinementRatio
def$zirs$textZIRS$description
```

`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, radius_km = EARTH_RADIUS_KM)
```

Arguments

`dggs`	A 'dggridR' grid object from dgconstruct()
`radius_km`	Radius the grid is sized on, in km or as a body name. A dggs carries no radius, so the default is Earth's; pass this when the dggs describes a grid on another body.

Details

The 'ISEA' and 'FULLER' projections with HEXAGON topology are supported. Other configurations will generate warnings.

A dggs has no field for the body it is sized on, so the resolution is all that carries over and the result is an Earth grid unless `radius_km` says otherwise. Cell area follows from the radius, so it is read from the radius given rather than from the dggs.

The orientation carries over: `pole_lon_deg`, `pole_lat_deg` and `azimuth_deg` place the icosahedron as DGGRID's `dggs_vert0_lon`, `dggs_vert0_lat` and `dggs_vert0_azimuth` do, and a field left out takes its standard ISEA value.

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

- Projection must be 'ISEA' or 'FULLER'
- 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()`

Examples

```
dggs <- as_dggrid(hex_grid(resolution = 4, aperture = 4))
from_dggrid(dggs)
```

`get_neighbors` *Get Neighboring Cells*

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,
  as_sf = FALSE,
  ring = FALSE
)
```

Arguments

`cell_id`	Cell IDs to find neighbors for: integer64 (or whole numbers) for ISEA grids, character 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`).
`as_sf`	Logical. If `TRUE`, return the links from each cell to its neighbours as great-circle lines between cell centres.
`ring`	Logical. If `TRUE`, return only the cells exactly `k` hops away, the hollow ring, rather than the whole disk (default `FALSE`).

Details

For **ISEA grids**, neighbors are computed using axial coordinate offsets in the quad IJ space. On aperture-7 grids of the icosahedron, a cell at a quad boundary takes its neighbors from digit arithmetic on its IGEO7 Z7 index, which is exact and needs no projection; on other ISEA grids such a cell's neighbors are found 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`. If `as_sf = TRUE`: an sf object of lines (MULTILINESTRINGs when one is split at the antimeridian), one row per cell and neighbour, with columns `cell_id`, `neighbor_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)

# Only the cells two hops away
get_neighbors(cell, g, k = 2, ring = TRUE)

# Links to the neighbours, drawn over the cells
links <- get_neighbors(cell, g, k = 2, as_sf = TRUE)
plot(sf::st_geometry(cell_to_sf(c(cell, links$neighbor_id), g)))
plot(sf::st_geometry(links), col = "red", add = TRUE)
```

---

globe_centers *Globe Center Presets*

Description

Named list of lon/lat coordinates for common globe views, accepted by the `center` argument of the grid `plot` method.

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 *Clip Hexagon Grid to Polygon Boundary*

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

```
# Metropolitan France: the world map also carries the overseas departements,
# and grid_clip() covers the bounding box of whatever it is given.
france <- sf::st_crop(
  sf::st_geometry(hexify_world[hexify_world$name == "France", ]),
  c(xmin = -5, ymin = 41, xmax = 10, ymax = 52)
)

# 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 *Generate a Global Hexagon Grid*

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), calls sf::st_wrap_dateline() to split antimeridian-crossing polygons, which flat maps and planar (GEOS) operations need. Set to FALSE for orthographic/globe projections where wrapping creates gaps, and for spherical (s2) operations: each cell then stays one polygon whose corners lie in -180..180, so a cell crossing the antimeridian spans a flat lon/lat map.

Details

Every cell of the grid, one row each, in cell ID order for ISEA grids. 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 *Get Grid Specification*

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 *Generate a Rectangular Grid of Hexagons*

Description

Creates hexagon polygons covering a rectangular geographic region. All cells that overlap the bounding box are included (not just cells whose center falls inside), ensuring full spatial coverage on both backends.

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. integer64 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 (integer64)

h3_cell_id H3 cell ID (character)

isea_area_km2 Area of the ISEA cell in km2, as `cell_area` returns it

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")
```

`hex9_warp_download` *Download Hex9's Warp Field*

Description

Hex9's own projection (`hex_grid(aperture = 9, projection = "akw")`) is Kaseorg's octahedral projection with an area-correcting warp that Griffin trained by optimal transport and publishes with 'libhex9', his reference implementation (Griffin 2026, Sec. 11b). The field is 19 MB of stored displacements and gradients, which hexify reads but does not ship; this function downloads it once from the 'libhex9' repository, checks it, and saves it where hexify looks for it.

Usage

```
hex9_warp_download(path = hex9_warp_path(), quiet = FALSE)
```

Arguments

`path`	Where to save the field. The default is hexify's user data directory (`tools::R_user_dir("hexify", "data")`), or the option `hexify.hex9_warp` when set; hexify reads it from the same place.
`quiet`	Passed to `utils::download.file()`.

Details

The file is 'libhex9"s `core/Sphere_16_fund.f64g.h9warp` at the commit of version 2.3.1, which the Hex9 agreement checks in hexify were run against. 'libhex9' is Apache-2.0 licensed. Its field is defined on the unit sphere: 'libhex9"s WGS84 functions first turn geodetic latitude into authalic latitude, while hexify reads latitude as spherical latitude, as 'libhex9"s `_sphere` functions do.

Value

The path of the saved file, invisibly.

References

Griffin, B. (2026). Hex9: A Quasi-Authalic, Quasi-Continuous Hexagonal DGGS on the Reference Ellipsoid. arXiv:2608.00022.

Examples

```
if (interactive()) {
  hex9_warp_download()
  g <- hex_grid(resolution = 5, aperture = 9, projection = "akw")
  lonlat_to_cell(16.37, 48.21, g)
}
```

HexData-class *HexData Class*

Description

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

Usage

```
## S4 method for signature 'HexData'
grid_info(x)

## S4 method for signature 'HexData'
cells(x)

## S4 method for signature 'HexData'
n_cells(x)

## S4 method for signature 'HexData'
nrow(x)

## S4 method for signature 'HexData'
ncol(x)
```

```
## S4 method for signature 'HexData'
dim(x)

## S4 method for signature 'HexData'
names(x)

## S4 method for signature 'HexData'
x$name

## S4 replacement method for signature 'HexData'
x$name <- value

## S4 method for signature 'HexData'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'HexData'
x[[i]]

## S4 replacement method for signature 'HexData,ANY,missing'
x[[i, j]] <- value

## S4 method for signature 'HexData'
show(object)

## S4 method for signature 'HexData'
summary(object, ...)

## S3 method for class 'hexify_data_summary'
print(x, ...)

## S4 method for signature 'HexData'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S4 method for signature 'HexData'
as.list(x, ...)

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

## S3 method for class 'HexData'
st_as_sf(
  x,
  ...,
  geometry = c("point", "polygon"),
  shape = c("hexagon", "gosper", "descendants"),
  depth = 3L
)
```

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

## 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,
  shape = c("hexagon", "gosper", "descendants"),
  depth = 3L,
  ...
)
```

Arguments

x	A HexData object; for `print()`, a data summary as `summary()` returns one
name	Column name
value	Replacement value
i, j	Row/column indices
...	For `plot()`, passed on to base `plot()`; ignored by the other methods
drop	Logical, whether to drop dimensions
object	A HexData object
row.names	Optional row names
optional	Logical (ignored)
geometry	For `st_as_sf()`: "point" (default) gives each row the centre of its cell, "polygon" the cell's boundary
shape, depth	Cell shape for polygon geometry and for `plot()`, as for `cell_to_sf`
y	Ignored (for S4 method compatibility)

basemap	Basemap specification: • TRUE or "world": Use built-in world map • FALSE or NULL: No basemap • "world_hires": High-resolution map from 'rnaturalearth' (requires the package) • An sf object: User-supplied vector map • A SpatRaster: User-supplied raster, drawn in grayscale
clip_basemap	Clip basemap to data extent (default TRUE). Clipping temporarily disables S2 spherical geometry to avoid edge-crossing errors.
basemap_fill	Fill color for basemap (default "gray90")
basemap_border	Border color for basemap (default "gray50")
basemap_lwd	Line width for basemap borders (default 0.5)
grid_fill	Fill color for grid cells (default "#E69F00" - amber/orange)
grid_border	Border color for grid cells (default "#5D4E37" - dark brown)
grid_lwd	Line width for cell borders (default 0.8)
grid_alpha	Transparency for cell fill (0-1, default 0.7)
fill	Column name for fill mapping (optional)
show_points	Show original points on top of cells (default FALSE). Points are jittered within their assigned hexagon.
point_size	Size of points. Can be: • A number (direct cex value) • A preset defining what fraction of a hex cell one point covers: "tiny" (~2\ "large" (~20\
point_color	Color of points (default "red")
point_alpha	Transparency for points (0-1, default 1)
jitter	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.
crop	Crop to data extent (default TRUE)
crop_expand	Expansion factor for crop (default 0.1)
main	Plot title

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.

Unlike [.data.frame, drop defaults to FALSE: selecting a single column returns a HexData object (preserving grid/ cell_id/cell_center) rather than dropping to a bare vector. Pass drop = TRUE explicitly to get data.frame-style dropping.

Value

- `show()`: the object, invisibly, after printing two lines: rows, columns and cells, and the grid
- `summary()`: a list of class `hexify_data_summary` carrying `rows`, `columns`, `column_names`, `n_cells`, `type`, the `grid` summary and a `preview` of the first rows; printing it lists all of these and returns it invisibly
- `grid_info()`: HexGridInfo object containing grid specification
- `cells()`: vector of unique cell IDs
- `n_cells()`: integer count of unique cells
- `nrow()`, `ncol()`, `dim()`: integer dimensions
- `names()`: character vector of column names (including virtual cell columns)
- `$`, `[[`: the requested column or cell data as a vector
- `$<-`, `[[<-`: the modified HexData object
- `[`: subsetted HexData object or extracted data
- `as.data.frame()`: data frame with original data plus cell columns
- `as.list()`: named list containing data, grid, cell_id, and cell_center
- `as_tibble()`: the original data (without geometry) as a tibble
- `plot()`: the object, invisibly
- `st_as_sf()`: an sf object, one feature per row
- `st_crs()`: the grid's coordinate reference system, as for a HexGridInfo

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: `bit64::integer64` for ISEA grids, character for H3 grids.

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

Plotting data

`plot()` draws the cells the data falls in, with an optional basemap. It builds polygons for the cells present in the data when it draws them, rather than storing them, to keep memory use low.

See Also

`hexify` for creating HexData objects, `HexGridInfo-class` for grid specifications

`hexify_heatmap` for ggplot2 plotting

Examples

```
df <- data.frame(lon = c(0, 10, 20), lat = c(45, 50, 55), n = 1:3)
result <- hexify(df, lon = "lon", lat = "lat", area_km2 = 1000)
result
summary(result)
summary(result)$column_names
cells(result)
n_cells(result)
head(as.data.frame(result))
result$cell_id
result[1:2, ]
st_crs(result)

# sf points at the cell centres, or the cells as polygons
sf_pts <- st_as_sf(result)
sf_poly <- st_as_sf(result, geometry = "polygon")

if (requireNamespace("tibble", quietly = TRUE)) {
  tibble::as_tibble(result)
}

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")
```

HexGridInfo-class *HexGridInfo Class*

Description

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

Usage

```
## S4 method for signature 'HexGridInfo'
x$name

## S4 method for signature 'HexGridInfo'
n_cells(x)

## S4 method for signature 'HexGridInfo'
names(x)
```

```
## S4 method for signature 'HexGridInfo'
show(object)

## S4 method for signature 'HexGridInfo'
summary(object, ...)

## S3 method for class 'hexify_grid_summary'
print(x, ...)

## S4 method for signature 'HexGridInfo'
as.list(x, ...)

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

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

## S4 method for signature 'HexGridInfo,missing'
plot(
  x,
  y,
  surface = c("sphere", "solid", "net"),
  center = c(lon = 15, lat = 32),
  projection = c("orthographic", "perspective"),
  distance = NULL,
  tilt = 0,
  rotation = 0,
  fov = NULL,
  cells = NULL,
  land = TRUE,
  ocean_fill = "#F4F6F8",
  land_fill = "#C9D0D6",
  land_border = "#8C969F",
  land_lwd = 0.4,
  grid_border = "#0072B2",
  grid_lwd = 0.8,
  face_edges = NULL,
  edge_col = "black",
  edge_lwd = 1.1,
  layout = "plane",
  seams = FALSE,
  seam_col = "#D55E00",
  seam_lwd = 1.6,
  graticule = FALSE,
  graticule_col = "#9AA3AB",
  graticule_lwd = 0.5,
```

```
  distortion = c("none", "angular", "areal"),
  tissot = FALSE,
  tissot_col = "#2E3439",
  tissot_lwd = 0.8,
  parents = NULL,
  parent_col = "#1B2A38",
  parent_lwd = 1.8,
  area_legend = FALSE,
  tabs = FALSE,
  step = 0.01,
  main = NULL,
  ...
)
```

Arguments

x	A HexGridInfo object; for `print()`, a grid summary as `summary()` returns one
name	Slot name
object	A HexGridInfo object
...	For `st_as_sf()`, passed on to `grid_global`; for `st_crs()`, passed on to `sf::st_crs()`; ignored by the other methods
y	Ignored
surface	`"sphere"`, `"solid"` or `"net"`. `"solid"` draws the flat faces of the grid's polyhedron (the icosahedron, or the octahedron or tetrahedron of a grid built on it). The solid and the net need an ISEA grid; an H3 grid is drawn on the sphere.
center	Point the view looks down on: a preset name from `globe_centers` or `c(lon, lat)`.
projection	`"orthographic"` or `"perspective"`.
distance	Distance of the untilted perspective camera from the sphere's centre, in sphere radii; greater than 1. `NULL` uses 3. A geostationary satellite sits at about 6.6 Earth radii.
tilt	Degrees the perspective camera swings about the point at `center`, between -90 and 90, keeping its range of `distance - 1` radii to that point. Positive values move the camera towards the bottom of the view, so it looks towards the top.
rotation	Degrees the view turns about the line of sight; positive values turn north clockwise.
fov	Field of view of the perspective camera in degrees, across the square the plot shows, centred on `center`; below 170. `NULL` frames the whole visible sphere instead.
cells	Cell IDs to draw. `NULL` draws every cell of the grid.
land	`TRUE` for the built-in `hexify_world`, `FALSE` for none, or an sf/sfc object of polygons.
ocean_fill	Fill colour of the surface.
land_fill	Fill colour of land; `NA` leaves land unfilled.

land_border	Colour of country outlines; NA draws none.
land_lwd	Line width of country outlines.
grid_border	Colour of cell boundaries.
grid_lwd	Line width of cell boundaries.
face_edges	Draw the edges of the solid's faces. NULL draws them for an ISEA grid; H3 is built on an icosahedron of its own, so its grid takes none.
edge_col	Colour of face edges.
edge_lwd	Line width of face edges.
layout	For surface = "net", the layout of the faces: a name net_layout takes, or a hexify_net object it built for this grid.
seams	For surface = "net", draw the cuts of the net: the sides of its pieces across which the map is not continuous, including its outline.
seam_col	Colour of seams.
seam_lwd	Line width of seams.
graticule	FALSE for none, TRUE for meridians and parallels every 30 degrees, or a number of degrees between them.
graticule_col	Colour of the graticule.
graticule_lwd	Line width of the graticule.
distortion	"none", "angular" for the maximum angular deformation of the face projection, or "areal" for its areal scale (1 everywhere on Snyder's equal-area projection). Needs an ISEA grid; a colour bar is drawn along the bottom.
tissot	FALSE for none, TRUE for Tissot ellipses at points about 30 degrees apart, or a number of degrees between them. Each ellipse is the image of a circle a fifth of that spacing in radius. Drawn on the net and the solid.
tissot_col	Colour of Tissot ellipses.
tissot_lwd	Line width of Tissot ellipses.
parents	Resolutions up whose cell outlines are drawn over the cells, such as 1 or 1:2; NULL for none.
parent_col	Colour of parent outlines, recycled over parents.
parent_lwd	Line width of parent outlines, recycled over parents.
area_legend	For surface = "net" of a grid on Snyder's equal-area projection, draw one hexagonal cell of each drawn resolution at the scale of the map, labelled with its area.
tabs	For surface = "net", draw glue tabs on the seams.
step	Spacing of the points along a cell boundary, as a fraction of a face edge.
main	Plot title.

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". The separator tells the two apart: "4,7" and the family "4/7" are the same grid at resolution 2, but their grids one level coarser are "4" and "7".

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

Value

- `show()`: the object, invisibly, after printing one line naming the grid
- `summary()`: a list of class `hexify_grid_summary` carrying `grid_type`, `aperture`, `resolution`, `area_km2`, `diagonal_km`, `crs`, `radius_km`, `earth`, `orientation` (`c(vert0_lon, vert0_lat, azimuth)`, empty for H3), `projection` (`NA` for H3), `polyhedron` (`NA` for H3), `ellipsoid` (`c(a_km, f)`, empty for a grid on the sphere) and `n_cells`; printing it lists every property and returns it invisibly
- `$`: the value of the requested slot
- `names()`: character vector of slot names
- `n_cells()`: the number of cells the grid contains
- `as.list()`: a named list of slot values
- `plot()`: the grid, invisibly
- `st_as_sf()`: an sf object of the grid's global cell set
- `st_crs()`: an object of class `crs`, as `sf::st_crs()` returns: an Earth grid gives 'WGS84', a grid on another body a longlat CRS on the sphere of its radius

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: for ISEA 0-30 for aperture 3, 0-29 for aperture 4, 0-21 for aperture 7 on the icosahedron (cell IDs fit in 64 bits); 0-15 for H3.

`area_km2` Numeric. Cell area in square kilometers.

`diagonal_km` Numeric. Centre spacing in kilometers: the short (flat-to-flat) diagonal of a regular hexagon of area `area_km2`, $\sqrt{2A/\sqrt{3}}$.

`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.

`orientation` Numeric. Where an ISEA grid's solid sits: `c(vert0_lon, vert0_lat, azimuth)` in degrees, vertex 0 and the azimuth of vertex 1 seen from it. Empty for H3 grids, whose orientation H3 fixes.

`projection` Character. How an ISEA-family grid projects each face of its solid onto its plane triangle: "isea" (Snyder's equal-area projection), "ivea" (van Leeuwen and Strebe's vertex-oriented equal-area projection) or "fuller" (Fuller's projection). Empty for H3 grids.

`polyhedron` Character. The solid an ISEA-family grid is built on: "icosahedron" or "octahedron". Empty for H3 grids.

`ellipsoid` Numeric. The ellipsoid of revolution an ISEA-family grid reads geodetic latitude on, `c(a_km, f)`: semi-major axis in km and flattening. Empty for a grid that reads latitude on the sphere, and for H3 grids.

Plotting a grid

`plot()` draws the cells of a grid in 3D, seen from above a chosen point: on the sphere, or on the flat faces of the solid the grid is built on. All surfaces take their cell boundaries from the same points, so a cell on the solid is the cell on the sphere folded flat.

`surface = "net"` lays the faces out flat, by default in the PLANE layout of DGGRID (`hexify_cell_to_plane` gives cell centres in the same coordinates); `layout` picks another, such as Van de Sande's Gosper World on the octahedron (see `net_layout`). A cell on a cut of the net is drawn in parts, one on each side. The net is drawn without a camera, so `center`, `projection`, `distance`, `tilt`, `rotation` and `fov` apply to the other two surfaces.

`distortion` colours the surface by the distortion of the face projection that maps the sphere onto the faces (see `projection_distortion`), and `tissot` draws Tissot's indicatrix on the faces: the ellipse each small circle of the sphere maps to. With `graticule` on the net, Snyder's projection shows small bends in the meridians and parallels where they cross the arcs from each face's centre to its corners.

`parents` draws the outlines of the cells some resolutions up over the grid's own cells, to show the hierarchy. On a net of an equal-area grid, `area_legend` draws one cell at the scale of the map, with its area, and the same for each resolution `parents` draws. `tabs` adds glue tabs along the seams of a net, one on each pair of edges glued together, so a printed net (for example through `pdf`) folds into the solid (Carr et al. 1997).

The view is an orthographic projection by default: parallel lines of sight, so the whole near half of the sphere shows. `projection = "perspective"` places a camera `distance` sphere radii from the sphere's centre, above `center`; it sees a smaller cap, and nearer cells appear larger. `tilt` swings that camera about the point at `center`, which it keeps looking at, so the surface is seen at a slant with the horizon towards the top; by default the plot frames the whole visible sphere. `fov` sets how wide the camera sees, and `rotation` turns the view about the line of sight in either projection.

References

Carr, D. B., Kahn, R., Sahr, K., Olsen, A. R. (1997). ISEA discrete global grids. Statistical Computing & Graphics Newsletter 8(2/3): 31-39.

See Also

`hex_grid` for the constructor function, `HexData-class` for hexified data objects

grid_global for the cells as sf polygons, projection_distortion, net_cells

Examples

```
grid <- hex_grid(area_km2 = 100000)
grid
summary(grid)
summary(grid)$n_cells
n_cells(grid)
grid$resolution
st_crs(grid)

# Every cell of a coarse grid as sf polygons
cells_sf <- st_as_sf(hex_grid(resolution = 2))
grid <- hex_grid(resolution = 3, aperture = 3)
plot(grid)
plot(grid, surface = "solid")
plot(grid, surface = "solid", land = FALSE, center = "pacific")
plot(grid, projection = "perspective", distance = 2, center = "europe")
plot(grid, surface = "solid", projection = "perspective",
     distance = 2.5, tilt = 25, rotation = 30)
plot(grid, surface = "net")
octa <- hex_grid(resolution = 3, aperture = 4, polyhedron = "octahedron",
                 orientation = "gosper")
plot(octa, surface = "net", layout = "gosper", seams = TRUE, graticule = TRUE)


# Distortion of the face projections, with Tissot's indicatrix
plot(grid, surface = "net", land = FALSE, distortion = "angular",
     tissot = TRUE, graticule = 15, cells = numeric(0))
fuller <- hex_grid(resolution = 3, aperture = 3, projection = "fuller")
plot(fuller, distortion = "areal", land = FALSE, cells = numeric(0))

# Two resolutions with an area legend, and a net to print and fold
fine <- hex_grid(resolution = 4, aperture = 3)
plot(fine, surface = "net", land = FALSE, parents = 1, area_legend = TRUE)
plot(grid, surface = "net", layout = "land", land = FALSE, seams = TRUE,
     tabs = TRUE)
```

---

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 spacing in km: the short (flat-to-flat) diagonal, equal to the distance between neighbouring cell centres
resolution	Grid resolution (0-30 for aperture 3, 0-29 for aperture 4, 0-21 for aperture 7 on the icosahedron). 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`: Cell IDs for each row (integer64 for ISEA, character for H3)
- `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, `st_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_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,
  shape = c("hexagon", "gosper", "descendants"),
  depth = 3L,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  legend_title = NULL,
  na_color = "gray90",
  theme_void = TRUE
)
```

Arguments

`data`	A HexData object from `hexify()`, a data frame with cell_id and cell_area columns, or an sf object with hexagon polygons.
`value`	Column name (as string) to use for fill color. If NULL, 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:

- NULL or FALSE: No basemap (default)
- "world" or TRUE: Use built-in hexify_world map (low resolution)
- "world_hires": Use high-resolution map from rnaturalearth (requires package)
- An sf object: User-supplied vector map, from any format sf::st_read() reads, such as a shapefile, 'GeoJSON' or 'GeoPackage'
- A SpatRaster: User-supplied raster, from any format terra::rast() reads, such as a 'GeoTIFF', drawn in grayscale

crs — 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)

colors — Color palette for the heatmap. Can be:

- A character vector of colors (for manual scale)
- A single RColorBrewer palette name (e.g., "YlOrRd", "Greens"), interpolated over its own range when a binned scale asks for more levels than the palette holds
- A viridis palette name ("viridis", "magma", "turbo", ...)
- NULL to use viridis

A name in neither family is an error naming the families.

breaks — Numeric vector of break points for binning continuous values, or NULL for continuous scale. Use Inf and -Inf for open-ended bins.

labels — Labels for the breaks (length should be one less than breaks). If NULL, labels are auto-generated.

hex_border — Border color for hexagons

hex_lwd — Line width for hexagon borders

hex_alpha — Transparency for hexagon fill (0-1)

basemap_fill — Fill color for basemap polygons

basemap_border — Border color for basemap polygons

basemap_lwd — Line width for basemap borders

mask_outside — Logical. If TRUE and basemap is provided, mask hexagon portions that fall outside the basemap polygons.

aperture — Grid aperture (default 3), used if data is from hexify()

shape — "hexagon" (default) draws each cell's own boundary. "gosper" draws it as a Gosper island and "descendants" as the outline of its descendants; both need an ISEA grid, and "descendants" one of aperture 7. See Details.

depth — For shape = "gosper", the number of times each edge is replaced, multiplying its segments by 3; for "descendants", the number of resolutions down, multiplying the cells behind one outline by 7.

xlim — Optional x-axis limits (in target CRS units) as c(min, max)

`ylim`	Optional y-axis limits (in target CRS units) as c(min, max)
`title`	Plot title
`legend_title`	Title for the color legend
`na_color`	Color for NA values
`theme_void`	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_world` *Simplified World Map*

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

sovereignt 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_aggregate *Aggregate Cell Values to a Coarser Resolution*

Description

Adds up, or averages, values given on cells into the cells `levels` resolutions coarser. The hexagons of successive resolutions do not nest, so a cell can lie partly in two or three coarser cells; `rule` says how its value is divided among them.

Usage

```
hex_aggregate(
  cell_id,
  value,
  grid,
  levels = 1L,
  rule = c("area", "centre"),
  measure = c("total", "mean"),
  na.rm = FALSE
)
```

Arguments

`cell_id`	Cell IDs, each listed once.
`value`	Numeric vector with one value per cell, or a numeric matrix or data frame with one row per cell, each column aggregated on its own.
`grid`	A HexGridInfo or HexData object at the resolution of `cell_id`.
`levels`	Number of resolutions up.
`rule`	`"area"` divides each cell among the coarser cells it overlaps by the share of its area in each; `"centre"` gives each cell wholly to the coarser cell holding its centre, the one `get_parent` returns.
`measure`	`"total"` adds the values up, for counts and other amounts; `"mean"` takes their mean weighted by the area each cell contributes, for densities and other values per unit area.
`na.rm`	If `TRUE`, a cell whose value is `NA` counts as absent from that column.

Details

The two rules. Write $|c \cap p|/|c|$ for the share of cell c's area that lies in the coarser cell p. The area rule gives p that share of c's value, which assumes the amount is spread evenly over c; the centre rule gives p all of it when p holds c's centre and nothing otherwise. A cell's shares add up to one under both, so both keep the total of the values. Only the area rule keeps where the amount lies: a uniform density gives every hexagonal coarser cell the same total under it, and each vertex cell (the twelve pentagons of the icosahedron) its sides over six of that. A mean is the total of value times area over the total area, both aggregated by the same rule.

Shares on the lattice. Within a face, the cells of one resolution are the Voronoi cells of a hexagonal lattice and those of the next coarser resolution the Voronoi cells of a sublattice, of index 3, 4 or 7, the aperture of the step. After an aperture-3 step the cells are centred on the parent centres and corners: the three parent edges meeting at a corner run through every second edge midpoint of the cell centred there and cut it into thirds. After an aperture-4 step the cells are centred on the parent centres and edge midpoints: a parent edge runs through the centre and two opposite corners of the cell on its midpoint and cuts it in half. After an aperture-7 step every centre lies inside a parent. Take the finer lattice as the Eisenstein integers $Z[\omega]$, $\omega = e^{2\pi i/3}$, with unit spacing, so a cell has circumradius $s = 1/\sqrt{3}$. The parent lattice is $\mu Z[\omega]$ with $\mu = 3 + \omega$, $|\mu|^2 = 7$, turned by $\arg \mu = \arctan(\sqrt{3}/5)$ (the mirror image $3+\bar{\omega}$ at the other twist). The centre child, of circumradius s, lies inside its parent, of inradius $\sqrt{7}/2$. The parent edge facing the neighbour at μ runs from the parent corner $\mu(1-\omega)/3 = 1 + (1-\omega)/3$, which is a corner A of the ring child centred at 1, through the parent's edge midpoint $\mu/2 = (1 + (2+\omega))/2$, which is the midpoint of the edge the children at 1 and $2+\omega$ share. That edge is the second from A, so the part of the child at 1 beyond its parent is the triangle of A, the next corner B and $\mu/2$: sides s and $s/2$ at 120 degrees, area $\sqrt{3}s^2/8$ against $3\sqrt{3}s^2/2$ for the cell, one twelfth. The half-turn about $\mu/2$ swaps the two parents and the two children, so the edge cuts one twelfth off the child at $2+\omega$ the other way. Each of the six ring children lies 11/12 in its parent and 1/12 in one neighbour, and the parent's area checks: $1 + 6 \cdot 11/12 + 6 \cdot 1/12 = 7$.

A Hex9 step (aperture 9 on the octahedron, Griffin 2026) divides each triangle of the solid's lattice into nine. A cell is the six lattice triangles around its centre, so a parent's edges are lines of the coarser lattice and therefore of the finer one. Six of a parent's nine children lie inside it; the other three are centred on its edges, and the edge through such a child's centre cuts it into its two half-hexagons, three triangles each: half in each of two parents. The parent's area checks: $6+6\cdot1/2 = 9$ children's worth.

The vertices of the solid are cell centres at every resolution, and the turn by the angle deficit about a vertex maps both lattices onto themselves, so the cells around a vertex are cut as in the plane; the cell at a vertex lies wholly inside the vertex cell above it. On Hex9 the vertices are lattice points no cell is centred on, and the cells there are cut along lattice lines as everywhere. Snyder's projection and the vertex-oriented one (`"ivea"`) are equal-area, so these plane shares are the shares on the sphere, for every cell, across quad and face edges and at the vertices. `get_parent` with `overlapping = TRUE` finds the coarser cells a cell overlaps, its own parent first, and the share follows from their number and the aperture of the step: one parent, 1; three after an aperture-3 step, 1/3 each; two after aperture 4, 1/2 each; two after aperture 7, 11/12 to the parent and 1/12 to the neighbour; two after a Hex9 step, 1/2 each, the parent holding the child's mode-0 half first.

Several levels. `levels > 1` applies the one-level rule level by level, each step with the shares of its own aperture: a grid built with `aperture = c(3, 7, 4)` aggregates from resolution 3 with halves, then twelfths, then thirds. A value at an intermediate resolution is again taken as spread evenly over its cell, so several levels up the area rule is the one-level rule repeated rather than the share of a fine cell's area in a coarse cell.

Fuller's projection and H3. Neither is equal-area, so the plane shares do not carry over to the sphere, and each share is measured. Both cells are clipped against each other where they are straight, which makes the pieces exact: an ISEA-family cell on each face plane it covers, an H3 cell on the gnomonic plane at the finer cell's centre, where its great-circle edges are straight. An H3 piece is a spherical polygon whose area is a sum of spherical triangles, exact to rounding. A face-plane piece has curved edges on the sphere; its area is that of the polygon through equally spaced points of its edges, extrapolated to infinitely many points (Romberg), which leaves an error near rounding. The

share is a piece's area over that of all the cell's pieces. Measured this way on Snyder's projection, the shares come out as the lattice's.

Value

A data frame with one row per coarser cell that receives a share of any listed cell, sorted by `cell_id`, and the aggregated values: a column `value` for a vector, the columns of `value` otherwise. With `measure = "mean"`, a coarser cell whose contributing cells are all `NA` gets `NA`.

References

Carr, D. B., Kahn, R., Sahr, K., Olsen, A. R. (1997). ISEA discrete global grids. Statistical Computing & Graphics Newsletter 8(2/3): 31-39.

See Also

`get_parent` for the coarser cells a cell overlaps, `hex_summarize` to aggregate points into cells

Examples

```
g <- hex_grid(resolution = 4, aperture = 3)
ids <- grid_global(g)$cell_id
set.seed(1)
counts <- rpois(length(ids), 5)

area <- hex_aggregate(ids, counts, g)
centre <- hex_aggregate(ids, counts, g, rule = "centre")
c(sum(counts), sum(area$value), sum(centre$value))

# A uniform density: every parent gets its own area, the hexagons one
# value and the twelve pentagons 5/6 of it
density <- cell_area(ids, g)
range(hex_aggregate(ids, density, g, levels = 2)$value)

# Values per unit area average instead
hex_aggregate(ids, data.frame(a = counts, b = 2 * counts), g,
              measure = "mean")[1:3, ]
```

`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 st_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, st_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, so that the same set of cells is named by fewer of them.

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, as `cell_to_index` returns for apertures 3, 4, 7 and 9.
`grid`	A HexGridInfo object specifying the grid.

Details

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

ISEA backend: Cells are grouped by `get_parent()`, and a group holding as many distinct cells as its parent has children replaces them, from the finest resolution up until no further compaction is possible. That count is the aperture away from the twelve icosahedron vertices and fewer at one, and it is read rather than assumed: a vertex cell sits at the corner of several quads and carries an index spelling in each, so appending a digit to any one spelling names neither all of its children nor only its children.

What is preserved is the set of cells: uncompacting the result at the original resolution returns the input. The covered area is not identical, because a hexagonal hierarchy is not congruent at any aperture – a parent does not tile exactly into its children – so an area computed on the compacted set differs from one computed on the original.

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


# ISEA, on the default aperture
g <- hex_grid(resolution = 2, aperture = 3)
children <- cell_to_index(get_children(40L, g)[[1]],
                          hex_grid(resolution = 3, aperture = 3))
hex_compact(children, g)
```

`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: Cells on the same quad are counted in the quad's own cell lattice, where the six neighbours are one step away and the distance is the axial hex distance `max(|di|, |dj|, |di - dj|)`. The lattice is not always the substrate the cell IDs are packed on – at aperture 3's odd resolutions one substrate point in three is a cell – so the difference is divided by the lattice generator first. Cross-quad distances use breadth-first search over `get_neighbors()` and return `NA` beyond a hundred rings.

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)


# ISEA: neighbouring cells are one hop apart
g <- hex_grid(resolution = 3, aperture = 3)
a <- lonlat_to_cell(10, 50, g)
hex_distance(a, get_neighbors(a, g)[[1]], g)
```

---

hex_extract *Extract Raster Values at Hex Cell Centers*

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 terra::SpatRaster object.
grid	A HexGridInfo or HexData object specifying the grid.
cells	Optional cell IDs to extract. If NULL (default), extracts at all cell centers from grid (only works for HexData objects or if a boundary is provided).
boundary	Optional sf 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_globe *Interactive Globe of a Grid*

Description

Draws the cells of a grid on a globe that turns under the mouse, rendered on the graphics card through WebGPU. A slider folds the solid the grid is built on into the sphere, and the shader blends the two surfaces. Values given per cell fill the cells through a colour ramp.

Usage

```
hex_globe(
  x,
  values = NULL,
  cells = NULL,
  surface = c("sphere", "solid"),
  lambert = NULL,
  lambert_path = c("steps", "single"),
  center = c(lon = 15, lat = 32),
  projection = c("orthographic", "perspective"),
  distance = NULL,
  tilt = 0,
  rotation = 0,
  fov = NULL,
  land = TRUE,
  ocean_fill = "#F4F6F8",
  land_fill = "#C9D0D6",
  land_border = "#8C969F",
  land_lwd = 0.4,
  grid_border = "#0072B2",
  grid_lwd = 0.8,
  face_edges = NULL,
  edge_col = "black",
  edge_lwd = 1.1,
  palette = "viridis",
  limits = NULL,
  na_fill = NA,
  smooth = FALSE,
  step = 0.01,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

x	A HexGridInfo object; for `print()`, a grid summary as `summary()` returns one
values	Numeric values, one per cell in `cells`, filling the cells. `NULL` draws no fill.
cells	Cell IDs to draw. `NULL` draws every cell of the grid.
surface	Surface shown first, `"sphere"` or `"solid"`, the flat faces of the grid's polyhedron; the slider folds one into the other. An H3 grid is drawn on the sphere only.
lambert	`NULL`, or a number from 0 to 3: the slider follows Lambert's construction of Snyder's projection, and the globe opens at that step (0 the sphere, 1 the Lambert points, 2 the nudged points, 3 the faces), in place of `surface`. Needs a grid with `projection = "isea"`.

lambert_path	How the points move along the construction: "steps", its three steps one after the other, or "single", one path each.
center	Point the view looks down on: a preset name from globe_centers or c(lon, lat).
projection	"orthographic" or "perspective".
distance	Distance of the untilted perspective camera from the sphere's centre, in sphere radii; greater than 1. NULL uses 3. A geostationary satellite sits at about 6.6 Earth radii.
tilt	Degrees the perspective camera swings about the point at center, between -90 and 90, keeping its range of distance - 1 radii to that point. Positive values move the camera towards the bottom of the view, so it looks towards the top.
rotation	Degrees the view turns about the line of sight; positive values turn north clockwise.
fov	Field of view of the perspective camera in degrees, across the square the plot shows, centred on center; below 170. NULL frames the whole visible sphere instead.
land	TRUE for the built-in hexify_world, FALSE for none, or an sf/sfc object of polygons.
ocean_fill	Fill colour of the surface.
land_fill	Fill colour of land; NA leaves land unfilled.
land_border	Colour of country outlines; NA draws none.
land_lwd	Line width of country outlines.
grid_border	Colour of cell boundaries.
grid_lwd	Line width of cell boundaries.
face_edges	Draw the edges of the solid's faces. NULL draws them for an ISEA grid; H3 is built on an icosahedron of its own, so its grid takes none.
edge_col	Colour of face edges.
edge_lwd	Line width of face edges.
palette	Colours of the ramp from low to high values, or the name of a palette of hcl.colors.
limits	Values at the two ends of the ramp; NULL uses the range of values. Values outside are drawn in the end colours.
na_fill	Fill colour of cells whose value is NA; NA leaves them unfilled.
smooth	Whether the fill of an ISEA grid blends between the values of neighbouring cells, linearly across the triangles of cell centres, rather than filling each cell with its own value.
step	Spacing of the points along a cell boundary, as a fraction of a face edge.
width, height	Size of the widget, as CSS units or pixels.
elementId	Id of the widget's HTML element.

Details

The cells of an ISEA grid are found per pixel on the graphics card, by the same projection and cell numbering as `lonlat_to_cell`, so the page holds the grid's description and the values rather than the cells' outlines, and a grid of any resolution draws as fast as a coarse one. Hex9 grids are found the same way; on Hex9's own projection (`projection = "akw"`) the page also carries the warp field (`hex9_warp_download`), 9.6 MB as a texture. The values sit on the graphics card as a texture laid out like the grid's quads, one block per quad and resolution, which the shader reads at the place it finds a cell. Where the cells shrink below a pixel the fill is read from the grid one resolution coarser, and so on, whose cells take the mean of the values in them, so a fine grid seen whole shows its pattern rather than noise. Borders keep their width at every zoom and fade out where the cells get too small to see. The pointer shows the ID and value of the cell under it. H3 cells are drawn from their outlines.

With `lambert`, the slider instead follows Lambert's construction of Snyder's projection (see `projection_stages`), on a grid with that projection: from the sphere (0), every point swings about its face's centre down onto the plane touching the sphere there, keeping its distance from the centre (1, Lambert's azimuthal equal-area projection); within that plane it moves to Snyder's azimuth and radius (2); and the plane scales about the sphere's centre onto the face (3), the face of the solid whose faces have the area of theirs on the sphere. The neighbouring faces' planes come apart along the way and meet again on the solid. A menu beside the slider switches between these three steps and one path per point that changes the angle, azimuth and radius together.

Drag to turn the globe. Drag with Shift held to turn the view about the line of sight and, in the perspective view, to tilt the camera. The mouse wheel moves the perspective camera closer or zooms the orthographic view. The view starts where the camera arguments, those of the grid's `plot` method, place it.

The globe needs the 'htmlwidgets' package and a viewer or browser with WebGPU: Chrome and Edge (on Linux with some graphics cards only), Firefox on Windows and macOS, Safari 26, and RStudio's viewer. Without WebGPU the widget shows a notice instead. The article https://gillescolling.com/hexify/articles/globe.html lists the versions.

Value

An htmlwidget.

See Also

`plot` for the same view as a static plot

Examples

```
if (requireNamespace("htmlwidgets", quietly = TRUE)) {
  grid <- hex_grid(resolution = 3, aperture = 3)
  hex_globe(grid)
  hex_globe(grid, surface = "solid", center = "pacific")

  cells <- seq_len(n_cells(grid))
  centres <- cell_to_lonlat(cells, grid)
  hex_globe(grid, values = centres$lat_deg, palette = "Blue-Red 3")
```

```
  # Lambert's construction of Snyder's projection, opened at the Lambert
  # points
  hex_globe(grid, lambert = 1, land = FALSE)
}
```

---

hex_globe_png *Save a Globe as a PNG Image*

Description

Draws a globe made by hex_globe with the same WebGPU shader and pipelines the widget uses, and saves the view it opens with, without its controls, as a PNG file. The image is drawn on the graphics card and read back from it, so it is the same on a machine without a display. Pixels off the globe are transparent.

Usage

```
hex_globe_png(
  widget,
  file,
  width = 800,
  height = 800,
  scale = 1,
  timeout = 30,
  renderer = c("wgpu", "chrome")
)
```

Arguments

widget	A globe from hex_globe.
file	Path of the PNG file to write.
width, height	Size of the view in CSS pixels.
scale	Image pixels per CSS pixel; 2 draws the view at twice the resolution.
timeout	Seconds to wait for the globe to be drawn in Chrome.
renderer	"wgpu" or "chrome", as above.

Details

renderer = "wgpu" draws through 'wgpu', the 'Rust' implementation of WebGPU, in the 'hexglobe' package, with no browser. renderer = "chrome" draws in headless Chrome, which needs the 'chromote' and 'htmlwidgets' packages and a Chromium browser, such as Chrome or Edge, that chromote::find_chrome() finds; the browser is started with its GPU, which WebGPU draws on.

Value

file, invisibly.

See Also

plot for a static plot that needs no graphics card

Examples

```
# Drawing needs the hexglobe package and a GPU adapter
if (requireNamespace("hexglobe", quietly = TRUE) && interactive()) {
  grid <- hex_grid(resolution = 3, aperture = 3)
  globe <- hex_globe(grid, surface = "solid", center = "pacific")
  hex_globe_png(globe, tempfile(fileext = ".png"), scale = 2)
}
```

---

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,
  orientation = "standard",
  projection = c("isea", "fuller", "ivea", "ak", "akw"),
  region = NULL,
  polyhedron = c("icosahedron", "octahedron", "tetrahedron"),
  ellipsoid = NULL
)
```

Arguments

area_km2	Target cell area in square kilometers. Mutually exclusive with resolution.
resolution	Grid resolution level: for ISEA 0-30 for aperture 3, 0-29 for aperture 4, 0-21 for aperture 7 on the icosahedron, where the cell count stays within the 2^63 - 1 cell IDs a 64-bit integer numbers; 0-18 for aperture 9 (Hex9); 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, 9 for Hex9 on the octahedron (see the Hex9 section), 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`; one shorter than `resolution` recurs, as OGC Topic 21 reads a list of refinement ratios, so `c(4, 3)` at resolution 5 refines by 4, 3, 4, 3, 4. It is stored as a comma-separated string with one aperture per level (`"4,3,4,3,4"`), which `aperture` also accepts. It names no level finer than `resolution`, so `get_children()` needs a family name. 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".

`orientation` Where the icosahedron of an ISEA grid sits on the sphere: "standard" (default), "ogc", "dymaxion", "gosper", "random", "region", "face", or `c(vert0_lon, vert0_lat, azimuth)` in degrees. See the Orientation section. H3 fixes its own orientation, so H3 grids take only "standard".

`projection` How an ISEA-family grid projects each icosahedron face onto its plane triangle: "isea" (default), Snyder's equal-area projection, "ivea", van Leeuwen and Strebe's vertex-oriented equal-area projection, "fuller", Fuller's projection, or, on the octahedron, "ak", Kaseorg's projection, or "akw", Kaseorg's with Hex9's trained warp (see `hex9_warp_download`). See the Projection section. H3 fixes its own projection, so H3 grids take only "isea".

`region` For `orientation = "region"` or `"face"`, the area to centre the grid on: `c(lon, lat)` in degrees, or an sf, sfc or bbox object, whose spherical centroid is used.

`polyhedron` The solid an ISEA-family grid is built on: "icosahedron" (default), "octahedron" or "tetrahedron". See the Polyhedron section. H3 grids are built on the icosahedron and take only the default. Aperture 9 is defined on the octahedron, which it takes when no solid is named.

`ellipsoid` The ellipsoid of revolution an ISEA-family grid reads geodetic latitude on: `NULL` (default) reads latitude on the sphere; "WGS84", "GRS80", "earth" (WGS84), "mars", "jupiter", "saturn", "uranus" or "neptune"; or `c(a, f)`, the semi-major axis in km and the flattening. Cells are then exactly equal-area on the ellipsoid, and the grid is sized on the radius of the sphere of the ellipsoid's area, which `radius_km` need not give. See the Earth Model section. H3 reads latitude on the sphere and takes only `NULL`.

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 on the icosahedron, 4 * aperture^res + 2 on the octahedron, 2 * aperture^res + 2 on the tetrahedron) 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: hexagon area varies by about a factor of 2 within a resolution, and by about 2.4 once the twelve pentagons are counted. The variation follows position on the icosahedron rather than latitude, with the smallest cells near face centres.

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.

Earth Model

By default a grid reads geodetic latitude as latitude on its sphere, as DGGRID and H3 do, and an Earth grid shares the 'WGS84' ellipsoid's area among its cells. Measured geodesically on 'WGS84', the cells' areas then run from 0.9955 times the reported area at the equator to 1.0090 times it at the poles (`paper/bench/bench_ellipsoid_area.R`): within the 1 percent error budget OGC Topic 21 allows a spherical earth model for an equal-area grid (Requirement 28), and systematic in latitude.

With `ellipsoid`, geodetic latitude is converted to authalic latitude, the latitude on the sphere of the ellipsoid's area below which that sphere holds the same area as the ellipsoid below the geodetic one, before projecting, and back after. This carries the ellipsoid onto that sphere keeping every area, so cells of Snyder's and of the vertex-oriented projection are equal-area on the ellipsoid itself; on 'WGS84' the measured areas agree with the reported ones to the measurement's own error, about 2e-6. Fuller's projection stays not equal-area, but its cell areas are then the cells' areas on the ellipsoid. The conversion moves a point by up to 0.13 degrees of latitude on 'WGS84', so a grid with an ellipsoid places points in other cells than the same grid without one, and than DGGRID.

Cell IDs, the hierarchy and neighbours are the same; kilometre figures of lengths are measured on the sphere of the ellipsoid's area.

The ISEA3H and ISEA7H DGGRS definitions OGC registers are `hex_grid(aperture = 3, ellipsoid = "WGS84", orientation = "ogc")` and its aperture-7 twin, and DGGAL's IVEA3H and IVEA7H the same with `projection = "ivea"`.

```
g <- hex_grid(resolution = 9, aperture = 3, ellipsoid = "WGS84")
hex_grid(resolution = 6, aperture = 7, ellipsoid = "mars")
hex_grid(resolution = 6, ellipsoid = c(6378.137, 1 / 298.257222101)) # GRS80
```

Orientation

An ISEA grid is built on a solid, an icosahedron unless `polyhedron` names another, and its orientation places that solid on the sphere: vertex 0 at longitude `vert0_lon` and latitude `vert0_lat`, and vertex 1 at azimuth `azimuth` seen from vertex 0, all in degrees, as DGGRID's `dggs_vert0_lon`, `dggs_vert0_lat` and `dggs_vert0_azimuth` do.

- "standard" is the ISEA orientation on the icosahedron: vertex 0 at 11.25E, 58.28N, azimuth 0, symmetric about the equator, with eleven of the twelve pentagons over the oceans and one in Sichuan (Sahr et al. 2003). On the octahedron it puts vertices at both poles and at longitudes 0, 90E, 180 and 90W on the equator.
- "ogc" is the orientation of the ISEA3H and ISEA7H DGGRS definitions registered with OGC and of DGGAL's grids: vertex 0 at 11.20E and latitude arctan of the golden ratio (58.28N) on the sphere, azimuth 0. With `ellipsoid = "WGS84"` that vertex lies at geodetic latitude 58.397145907431N.
- "dymaxion" is Fuller's orientation of the icosahedron for his Dymaxion map: vertex 0 at 5.2454W, 2.3009N, azimuth 7.46658 (Sahr et al. 2003), with all twelve pentagons in the ocean. With `projection = "fuller"` the grid lies on Fuller's Dymaxion map.
- "gosper" places the octahedron as Van de Sande's Gosper World places its poles, at midpoints of octahedron edges: vertex 0 at 21.25W, 45N, vertex 1 across the north pole from it, all six vertices at least 619 km offshore. `net_layout` lays it out as four hexagons. The standard octahedron puts the poles at vertices instead, as Rus's four-hexagon map does.
- "random" draws vertex 0 uniformly on the sphere and the azimuth uniformly in [0, 360), from R's random number generator, so `set.seed()` repeats it. Grids in several random orientations show how much a result depends on where the cell boundaries fall.
- "region" places the grid as DGGRID's `REGION_CENTER` does: the centre of `region` lands on the midpoint of an edge of the solid, the middle of the two faces sharing it, far from the vertices where the vertex cells and the largest distortion sit.
- "face" places the centre of `region` on the centre of a face. On the icosahedron that is about 20.9 degrees from the nearest face edge and 37.4 degrees from the nearest vertex, so a region up to about 40 degrees across lies on one face, away from the face edges where the projection bends.

Rotating the icosahedron rotates the grid with it, so cell IDs, the cell hierarchy and neighbours are the same under every orientation; only where each cell sits on the sphere changes. Every function taking the grid reads its orientation. On a grid with an `ellipsoid` the orientation places the solid on the sphere of the ellipsoid's area, so `vert0_lat` is an authalic latitude; a `region` is given in geodetic latitude.

```
alps <- hex_grid(area_km2 = 100, orientation = "region", region = c(10, 46.5))
hex_grid(resolution = 8, orientation = c(0, 90, 0))   # vertex 0 at the pole
```

Projection

An ISEA-family grid lays its cells out on the plane triangles of the icosahedron's faces, and the projection carries each spherical face onto its triangle. With `"isea"`, Snyder's icosahedral equal-area projection, every cell of a resolution has the same area. With `"ivea"`, van Leeuwen and Strebe's (2006) vertex-oriented great-circle projection, cells are equal-area as well: the projection maps the great circles through each vertex of a face to straight lines, which leaves no cusps along the face edges and gives lower, more even angular distortion than Snyder's (their Table 1). These are DGGAL's IVEA3H and IVEA7H (Jacovella-St-Louis et al. 2025); DGGAL places the icosahedron at `orientation = c(11.20,58.282525588538995, 0)`, 0.05 degrees west of the standard one, and reads latitudes as WGS84 geodetic, mapping them to the authalic sphere. With `"fuller"`, Fuller's projection keeps lengths along the face edges and is not equal-area, so `area_km2` is the mean cell area and `cell_area()` reports each cell's own; the Equal-Area Earth Reference System of OGC Topic 21 asks for cells of equal area, so a Fuller grid falls outside it. These are DGGRID's `dggs_proj` `ISEA` and `FULLER`, giving FULLER3H, FULLER4H, FULLER7H and FULLER43H. Cell IDs, the hierarchy and neighbours are the same under every projection; where each cell's centre and corners sit on the sphere differs.

```
hex_grid(resolution = 5, aperture = 3, projection = "ivea")     # IVEA3H
hex_grid(resolution = 5, aperture = 4, projection = "fuller")   # FULLER4H
```

Polyhedron

Snyder's equal-area projection and the vertex-oriented one are defined on every regular solid with triangular faces, and a hexagonal grid lies on the icosahedron, the octahedron and the tetrahedron alike: each pair of faces sharing an edge forms a diamond of cells, and the cell at each vertex has one side per face meeting there. The icosahedron has twelve pentagons, the octahedron six squares and the tetrahedron four triangles; every other cell is a hexagon, and every cell of a resolution has the same area except the vertex cells, which have five sixths, two thirds or one half of it, by their sides. The octahedron's cells are more distorted than the icosahedron's and the tetrahedron's more again, so they suit layouts that need their four- or three-fold symmetry rather than analysis. Fuller's projection is defined on the icosahedron only.

The tetrahedron unfolds into a strip of its two diamonds. Each long side of the strip folds onto itself at its midpoint, so the second diamond holds the far edge of its box in place of the near edge it shares with the first; its cells still number one box's worth.

```
hex_grid(resolution = 5, aperture = 4, polyhedron = "octahedron")
hex_grid(resolution = 5, aperture = 4, polyhedron = "tetrahedron")
```

Hex9

`aperture = 9` builds Hex9 (Griffin 2026), the hexagonal grid of the octahedron that refines by 9 with no central child. Level L divides each octahedron edge into 3^{L+1} steps of a triangular lattice and centres a hexagon on every third lattice point, one class of a three-colouring of the lattice that leaves the solid's vertices out; the octahedron's even vertex valence keeps that class consistent across its faces, which the icosahedron's odd valence does not. There are $12 \cdot 9^L$ cells, all hexagons. Two

meet at each of the six vertices, a corner each, and share both edges there, so those twelve cells have five neighbours. On an equal-area projection every cell has the same area.

A line through each cell's centre cuts it into two halves, each inside one triangle of the next coarser level, and the cell is named through its half in a triangle of Griffin's mode 0: one digit for each level of triangles holding that half, digit 0 picking one of the twelve cells of resolution 0 (0-11) and every further digit one of nine (0-8). The cell ID is that digit string read as a number, digit 0 counting 9^L, plus one, so the IDs of a resolution run from 1 to $12 \cdot 9^L$; the digit strings are those of 'libhex9', Griffin's reference implementation, and `cell_to_index()` writes a cell's 'libhex9' label, its digits and key tail. A cell's parent is the cell of the coarser resolution holding its mode-0 half, and every cell has nine children. Three of them straddle the cell's boundary, so a cell's children cover it only in part.

Hex9's own projection is Kaseorg's octahedral projection, `"ak"` (Griffin 2026, Sec. 11a): it maps each octant's weights on its three vertices through tangents of the weights, coupled by a fitted constant, and is not equal-area, its cells varying by about 20 percent in area across a face. 'libhex9' corrects that with a warp of each face trained by optimal transport (Sec. 11b): `"akw"` reads it (`hex9_warp_download` fetches it), and a grid on it names every point by the cell 'libhex9"s functions on the sphere name it. Any of the face projections serves the same cells of the octahedron, with the same IDs; where each cell lies on the sphere follows `projection`.

```
hex_grid(resolution = 4, aperture = 9)                        # Hex9
hex_grid(resolution = 4, aperture = 9, projection = "ivea")
hex_grid(resolution = 4, aperture = 9, projection = "ak")
```

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_rays *Ray Glyphs for Cell Values*

Description

Draws one ray per cell from its centre whose angle shows a value, as in Carr et al. (1997): the ray points down for the smallest value, level for the middle of the range and up for the largest. A confidence arc at the ray's tip spans the angles of a lower and an upper bound. A second variable gets a ray of its own pointing to the left from the same centre, so one glyph shows two values.

Usage

```
hex_rays(
  cell_id,
  value,
  grid,
  lower = NULL,
  upper = NULL,
  value2 = NULL,
  lower2 = NULL,
  upper2 = NULL,
  limits = NULL,
  limits2 = NULL,
  length = 0.85
)
```

Arguments

`cell_id`	Cell IDs, each listed once.
`value`	Values shown by the right-hand rays.
`grid`	A HexGridInfo or HexData object.
`lower`, `upper`	Optional bounds of `value` (such as a confidence interval), drawn as an arc at the tip of the ray.
`value2`, `lower2`, `upper2`	Optional second variable and its bounds, drawn to the left.
`limits`	Values drawn straight down and straight up: `c(min,max)`. `NULL` uses the range of the values and bounds; with a second variable both share it unless `limits2` is given.
`limits2`	Limits of the second variable.
`length`	Length of a ray as a fraction of the distance from the cell centre to the middle of a wall.

Details

Rays run in the plane tangent to the sphere at the cell centre, with up towards north, and are returned as lines in longitude and latitude, ready for any map of the cells.

Value

An sf object of LINESTRINGs with columns `cell_id`, `side` (`"right"` or `"left"`) and `part` (`"ray"` or `"arc"`).

References

Carr, D. B., Kahn, R., Sahr, K., Olsen, A. R. (1997). ISEA discrete global grids. Statistical Computing & Graphics Newsletter 8(2/3): 31-39.

See Also

hex_triangles for the change towards each neighbour

Examples

```
g <- hex_grid(resolution = 4, aperture = 3)
cells <- lonlat_to_cell(c(5, 10, 15), c(45, 47, 49), g)
trend <- c(-1, 0.2, 1.4)
rays <- hex_rays(cells, trend, g, lower = trend - 0.5, upper = trend + 0.5)
plot(sf::st_geometry(cell_to_sf(cells, g)))
plot(sf::st_geometry(rays), add = TRUE, col = "#D55E00")
```

---

hex_sample *Random Points Inside Cells*

Description

Draws points uniformly over the area of each cell on the sphere, as DGGRID's randpts output does; on a grid with an ellipsoid, uniformly over its area on the ellipsoid. Each point is assigned back to its cell by lonlat_to_cell, so the points of a cell are exactly the points the grid places in it. Use set.seed() for a reproducible draw.

Usage

```
hex_sample(cell_id, grid, n = 1L)
```

Arguments

cell_id	Cell IDs. integer64 for ISEA grids, character for H3 grids.
grid	A HexGridInfo or HexData object.
n	Number of points per cell: one number, or one per cell.

Details

Points are drawn uniformly in a spherical cap around the cell centre that holds the whole cell, and those outside the cell are drawn again.

Value

A data frame with columns cell_id, lon and lat, the points of each cell together in the order of cell_id.

See Also

cell_to_lonlat for cell centres

Examples

```
grid <- hex_grid(resolution = 3, aperture = 3)
cell <- lonlat_to_cell(10, 50, grid)
set.seed(1)
pts <- hex_sample(cell, grid, n = 200)
plot(sf::st_geometry(cell_to_sf(cell, grid)))
points(pts$lon, pts$lat, pch = 20, cex = 0.5)
```

---

hex_smooth *Smooth Cell Values Over Neighbouring Cells*

Description

Replaces each cell's value by the weighted mean of its own value and the values of its edge neighbours, and repeats this `steps` times: the neighbour-matrix smoother of Carr et al. (1997). Each neighbour has weight 1 and the cell itself `self_weight`. On an equal-area grid every cell has the same area, so the mean is also the area-weighted mean.

Usage

```
hex_smooth(cell_id, value, grid, steps = 1L, self_weight = 1, group = NULL)
```

Arguments

`cell_id`	Cell IDs, each listed once.
`value`	Numeric values, one per cell.
`grid`	A HexGridInfo or HexData object.
`steps`	Number of smoothing steps. Each step widens the reach by one ring of cells.
`self_weight`	Weight of a cell's own value relative to each neighbour's. `0` replaces a value by the mean of its neighbours.
`group`	Optional vector, one entry per cell; only neighbours in the same group contribute.

Details

Only the cells in `cell_id` take part: a neighbour that is not listed, or whose value is `NA`, does not contribute, so a cell at the edge of the data averages over the neighbours it has. A pentagon averages over its five neighbours. A cell whose value is `NA` stays `NA`.

`group` restricts the smoothing to cells of the same group, such as land and ocean: a neighbour in another group does not contribute, so values do not mix across a coastline. A cell whose group is `NA` keeps its value and contributes to no other cell.

Value

Numeric vector of smoothed values, in the order of `cell_id`.

References

Carr, D. B., Kahn, R., Sahr, K., Olsen, A. R. (1997). ISEA discrete global grids. Statistical Computing & Graphics Newsletter 8(2/3): 31-39.

See Also

`get_neighbors()` for the neighbours a step averages over

Examples

```
g <- hex_grid(resolution = 4, aperture = 3)
cells <- grid_global(g)
set.seed(1)
noise <- rnorm(nrow(cells))
smooth <- hex_smooth(cells$cell_id, noise, g, steps = 3)
c(sd(noise), sd(smooth))

# Keep land and ocean apart
ctr <- sf::st_as_sf(cell_to_lonlat(cells$cell_id, g),
                    coords = c("lon_deg", "lat_deg"), crs = 4326)
land <- lengths(sf::st_intersects(ctr, hexify_world)) > 0
coastal <- hex_smooth(cells$cell_id, noise, g, steps = 3, group = land)
```

---

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

`hex_data`	A HexData object (from `hexify()`).
`...`	Named summary expressions (tidyeval). Each expression is evaluated per cell group with the group's columns available by name. Examples: `mean_temp = mean(temperature)`, `n_species = length(unique(species))`, `max_elev = max(elevation, na.rm = TRUE)`. Provide either `...` or `.fns`, not both.
`.fns`	Optional named list of functions for formula-style aggregation. Example: `.fns = list(mean_temp = ~mean(temperature))`. Provide either `...` or `.fns`, not both.
`geometry`	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 km^2

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_triangles` *The Triangles of Cells Towards Each Neighbour*

Description

Cuts each cell into one triangle per wall, from the centre to that wall: six for a hexagon, five for a pentagon. With values, each triangle carries the change from its cell's value to the neighbour's across the wall, so colouring the triangles shows the change from cell to cell in every direction (Carr et al. 1997, after K. and R. Keister).

Usage

```
hex_triangles(cell_id, grid, value = NULL)
```

Arguments

cell_id	Cell IDs, each listed once.
grid	A HexGridInfo or HexData object.
value	Optional values, one per cell. A neighbour not in cell_id or with an NA value gives an NA change.

Value

An sf object of POLYGONs with columns cell_id, neighbor_id and, with values, value, neighbor_value and change (neighbor_value - value).

References

Carr, D. B., Kahn, R., Sahr, K., Olsen, A. R. (1997). ISEA discrete global grids. Statistical Computing & Graphics Newsletter 8(2/3): 31-39.

See Also

hex_rays, get_neighbors, wall_metrics

Examples

```
g <- hex_grid(resolution = 3, aperture = 3)
cells <- grid_rect(c(0, 40, 30, 60), g)$cell_id
ctr <- cell_to_lonlat(cells, g)
tri <- hex_triangles(cells, g, value = ctr$lat_deg)
plot(tri["change"], border = NA)
```

---

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: Cells are expanded through `get_children()`, a level at a time, until the target resolution is reached. Appending each digit of the aperture names the children of a cell whose whole ancestry lies inside one quad, but not those of a cell at an icosahedron vertex, whose children carry index spellings in the several quads meeting there.

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)


# ISEA, on the default aperture
g <- hex_grid(resolution = 2, aperture = 3)
hex_uncompact(cell_to_index(40L, g), g, target_resolution = 3L)
```

---

hex_zonal *Zonal Statistics for Hex Cells*

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("8528342bfffffff", "85283473fffffff", "85283447fffffff")
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 *Check If an Object Is a HexData*

Description

Check If an Object Is a HexData

Usage

```
is_hex_data(x)
```

Arguments

x	Object to check

Value

Logical

Examples

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

---

is_hex_grid *Check If an Object Is a HexGridInfo*

Description

Check If an Object Is a HexGridInfo

Usage

```
is_hex_grid(x)
```

Arguments

x	Object to check

Value

Logical

Examples

```
is_hex_grid(hex_grid(resolution = 3))
is_hex_grid(list(resolution = 3))
```

`is_pentagon` *Detect Pentagon Cells*

Description

Identifies which cells are pentagons. A hexagonal grid on the icosahedron has exactly 12 pentagons, one at each icosahedron vertex, with 5 neighbors instead of 6. Grids on the octahedron and the tetrahedron have none: their vertex cells are six squares and four triangles.

Usage

```
is_pentagon(cell_id, grid)
```

Arguments

`cell_id`	Cell IDs to check. integer64 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: A vertex cell sits at a vertex of the solid, which is the (i, j) = (0, 0) cell of its quad, and has one side per face meeting there. Each input cell ID is decoded to its own (quad, i, j) and tested, 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` *Convert Longitude/Latitude to Cell ID*

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

ISEA cell IDs number a grid's cells from 1 and pass 2^53, the largest whole number a double holds exactly, at fine resolutions, so they are returned as 64-bit integers. Functions taking cell IDs also accept whole numbers below 2^53 and character strings of digits.

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

Cell IDs: a `bit64::integer64` vector for an ISEA grid, a character vector of H3 indices for an H3 grid

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)
```

net_cells *The Parts of Cells on a Net Layout*

Description

Cuts each cell into its parts on the pieces of a net layout and places them in the plane. A cell on one piece has one part. A cell that crosses from one piece to another stays whole on the map where the two pieces are joined, and is split where a seam of the net runs through it: its parts then lie apart, and Carr et al. (1997) call it a cloned cell. A cell on a piece the layout shows twice also has a part on each copy.

Usage

```
net_cells(layout, grid, cells = NULL)
```

Arguments

layout	A hexify_net object from net_layout.
grid	The HexGridInfo object the layout was built for.
cells	Cell IDs; NULL for every cell of the grid.

Details

Cell walls are straight on the faces of the solid and the layout moves pieces rigidly (or, for "rhombic", by one shear that keeps areas to one constant), so a part's map area is exact. Snyder's projection is equal-area, so on a grid built on it the map areas of a cell's parts add up to the cell's share of the sphere: a face of the net has area $\sqrt{3}/4$, and $4\pi/n$ of the unit sphere for a solid of n faces.

Value

A data frame with one row per part: cell_id, piece (position in layout$pieces), group (parts of a cell that the map joins share a group, numbered from 1 within the cell), area (map area in units of a face edge squared) and the centroid x, y. A cell whose parts fall into more than one group is cut by a seam.

References

Carr, D. B., Kahn, R., Sahr, K., Olsen, A. R. (1997). ISEA discrete global grids. Statistical Computing & Graphics Newsletter 8(2/3): 31-39.

See Also

net_layout, net_project

Examples

```
g <- hex_grid(resolution = 3, aperture = 3)
net <- net_layout(g, "plane")
parts <- net_cells(net, g)
cut <- tapply(parts$group, parts$cell_id, max) > 1
sum(cut)

# On Snyder's equal-area projection every hexagon has the same map area,
# however the net cuts it
area <- tapply(parts$area, parts$cell_id, sum)
hexagon <- !is_pentagon(as.numeric(names(area)), g)
range(area[hexagon])
```

`net_layout` *Lay Out the Faces of a Grid's Solid in the Plane*

Description

Builds a flat map of the sphere from the faces of the solid an ISEA grid is built on: the face projection of the grid (Snyder's equal-area projection, or Fuller's) maps the sphere onto the faces, and the layout cuts the faces into pieces and places each piece in the plane by a rotation and a translation (the rhombic layout also shears). Draw a grid on it with `plot(grid, surface = "net", layout = )`, and place points on it with `net_project`.

Usage

```
net_layout(
  x,
 layout = c("plane", "gosper", "gosper_flower", "gosper_land", "rhombic", "land"),
  centre = NULL,
  mirror = FALSE,
  land = TRUE
)
```

Arguments

`x`	A HexGridInfo object from `hex_grid`, built on the icosahedron, the octahedron or the tetrahedron.
`layout`	Name of the layout; see Details.
`centre`	For the Gosper layouts, the point `c(lon, lat)` whose tile lies in the middle, at the origin. The map is turned so the hexagons have horizontal top and bottom sides, with north at that point within 30 degrees of up the page. `NULL` uses the north pole.
`mirror`	For the Gosper layouts, `TRUE` builds the hexagons on the other four faces of the octahedron, the mirror image of the default tiling.
`land`	For `layout = "land"` and `"gosper_land"`, the land the net keeps joined: `TRUE` for the built-in `hexify_world`, or an sf/sfc object of polygons.

Details

The layouts are:

`"plane"` The faces in the PLANE layout of DGGRID, the layout `hexify_cell_to_plane` reads: five strips of four triangles on the icosahedron, four diamonds side by side on the octahedron, one triangle of edge 2 on the tetrahedron.

`"gosper"` Van de Sande's Gosper World on the octahedron: four regular hexagons, each one face of the octahedron and a third of each of its three neighbours, the three neighbouring faces cut along the lines from their centres to their corners. The tile holding `centre` lies in the middle and the other three are attached on alternate sides.

`"gosper_flower"` The tile holding `centre` surrounded by the six tiles that border it, so each of the other three tiles appears twice (Van de Sande 2024, Fig. 5). The map is continuous across the sides of the middle tile; the three seams run out from its corners that are vertices of the octahedron.

`"gosper_land"` The four Gosper tiles joined along the three hexagon sides, out of the twelve, that cross the most land of `land`, so the continents stay whole where they can, as in Van de Sande's Fig. 1 (2024).

`"rhombic"` DGGAL's 5 x 6 rhombic space on the icosahedron (Jacovella-St-Louis et al. 2025): each face sheared onto half of a unit square, the ten diamonds a staircase of squares. Areas keep one constant. DGGAL's y axis points down the page, so a point at DGGAL's (x, y) is placed at (x, -y).

`"land"` A net cut along the solid's edges that cross the least land of `land`, so the faces stay joined across land where they can, in the spirit of Fuller's Dymaxion map, which also splits two faces. With `hex_grid(orientation = "dymaxion", projection = "fuller")` it is a Dymaxion-style map of whole faces.

On the octahedron with Snyder's projection the Gosper World is equal-area, where Van de Sande built it from a gnomonic projection onto a rhombic dodecahedron. His poles lie at midpoints of octahedron edges, which is `hex_grid(polyhedron = "octahedron", orientation = "gosper")`; Rus's four-hexagon map puts them at octahedron vertices, which is the standard orientation of the octahedron.

Value

A `hexify_net` object: a list with the layout's `name`, the grid's solid in `polyhedron`, the `icosa` argument of the grid, and `pieces`, one list per placed piece with its `face` (from 0), its `region` (three corners in the face's triangle coordinates), their `labels`, the affine map `A` and `b` placing a point `t` at `A t + b`, and the `tile` it belongs to.

References

Van de Sande, A. (2024). Gosper World: A Hexagonal Map Using Gosper Fractals. *Bridges 2024 Conference Proceedings*, 507-510.

Rus, J. (2017). Flowsnake Earth. *Bridges 2017 Conference Proceedings*, 237-244.

Jacovella-St-Louis, J., Padilla Ruiz, M., Moreira de Sousa, L., Peterson, P., Stefanakis, E. (2025). Interoperable global and local indexing of discrete global grid systems based on the ISEA projection for efficient storage, processing and transmission. *Abstracts of the International Cartographic Association* 10, 126. doi:10.5194/icaabs101262025

See Also

net_project to place points on a layout

Examples

```
grid <- hex_grid(resolution = 3, aperture = 4, polyhedron = "octahedron",
                 orientation = "gosper")
world <- net_layout(grid, "gosper")
plot(grid, surface = "net", layout = world, seams = TRUE)
plot(grid, surface = "net", layout = "gosper_flower", land = TRUE)
```

---

net_project *Place Points on a Net Layout*

Description

Projects longitude/latitude points onto their face of the grid's solid and places them where the layout puts that part of the face. A point on a piece the layout shows more than once, or on a cut between two pieces, has one row per place.

Usage

```
net_project(layout, lon, lat)
```

Arguments

layout	A hexify_net object from net_layout
lon, lat	Longitudes and latitudes in degrees

Value

A data frame with columns point (position in lon), piece (position in layout$pieces), x and y, in units of a face edge.

Examples

```
grid <- hex_grid(resolution = 3, aperture = 4, polyhedron = "octahedron",
                 orientation = "gosper")
world <- net_layout(grid, "gosper")
net_project(world, c(16.37, -74.0), c(48.21, 40.71))
```

n_cells *Get Number of Cells*

Description

Counts cells: those a dataset occupies, or those a grid contains.

Usage

```
n_cells(x)
```

Arguments

x	A HexData or HexGridInfo object

Value

For a HexData, the number of distinct cells its rows fall in. For a HexGridInfo, the number of cells the grid divides the body into, which runs past integer range at fine resolutions and so comes back as a double.

Examples

```
grid <- hex_grid(area_km2 = 100000)
n_cells(grid)

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

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

```
# Metropolitan France: the world map also carries the overseas departements,
# and grid_clip() covers the bounding box of whatever it is given.
france <- sf::st_crop(
  sf::st_geometry(hexify_world[hexify_world$name == "France", ]),
  c(xmin = -5, ymin = 41, xmax = 10, ymax = 52)
)
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")
```

`projection_distortion` *Distortion of a Grid's Face Projection*

Description

Tissot's indicatrix of the projection that maps the sphere onto the faces of the grid's solid (Snyder's or the vertex-oriented equal-area projection, or Fuller's): a small circle on the sphere maps to an ellipse on the face, with semi-axes `a` and `b` times the circle's radius.

Usage

```
projection_distortion(x, lon, lat, stage = c("face", "lambert", "nudge"))
```

Arguments

`x`	A HexGridInfo object from `hex_grid` with an ISEA grid. H3 is built on a gnomonic projection of its own and is not covered.
`lon`, `lat`	Longitudes and latitudes in degrees.
`stage`	`"face"`, the grid's face projection, or a step of Lambert's construction, `"lambert"` or `"nudge"`, on a grid with Snyder's projection (`projection = "isea"`).

Details

The scale factors are the singular values of the projection's derivative, computed exactly by forward-mode automatic differentiation of the projection code. Face-plane lengths are measured on the plane triangle that has the face's area, so the areal scale `a * b` is 1 everywhere under Snyder's projection and IVEA, and its mean over a face is 1 under Fuller's.

Snyder's projection bends along the arcs from each face's centre to its corners, where its derivative jumps, and IVEA along those and the arcs to the edge midpoints; there and on face edges the result is the derivative on one side. Its angular deformation is largest at the face centre, approached along those arcs: 17.27 degrees, with scale factors 1.163 and 0.860 (Snyder 1992, Table 1). At a face centre the projection has a different derivative along each direction, and the result is the limit along the azimuth the point's coordinates give.

`stage` reads the indicatrix at an earlier step of Lambert's construction of Snyder's projection (see `projection_stages`): `"lambert"`, Lambert's azimuthal equal-area projection about the face centre onto the tangent plane, with scale factors $1/\cos(z/2)$ across the radius and $\cos(z/2)$ along it at arc z from the centre; `"nudge"`, Snyder's point on the tangent plane before the scaling onto the face plane, whose scale factors are those of `"face"` over R', so its areal scale is $1/R'^2$ everywhere. Both read lengths on the tangent plane in units of the sphere's radius.

Value

A data frame with one row per point: `lon`, `lat`, `face` (from 0), the scale factors `a` (largest) and `b` (smallest), the maximum angular deformation `angular` $2\arcsin((a-b)/(a+b))$ in degrees, the areal scale `areal` ab, and `angle`, the direction of the longer axis on the face plane in degrees anticlockwise from the face's x axis.

References

Snyder, J. P. (1992). An equal-area map projection for polyhedral globes. *Cartographica* 29(1), 10-21. doi:10.3138/27H78K8848821752

van Leeuwen, D. and Strebe, D. (2006). A "slice-and-dice" approach to area equivalence in polyhedral map projections. *Cartography and Geographic Information Science* 33(4), 269-286. doi:10.1559/152304006779500687

See Also

`plot` with `distortion` and `tissot` to map it; `projection_stages` for the points at each step of Lambert's construction

Examples

```
g <- hex_grid(resolution = 3, aperture = 3)
projection_distortion(g, c(0, 16.37), c(0, 48.21))
projection_distortion(g, c(0, 16.37), c(0, 48.21), stage = "lambert")

gf <- hex_grid(resolution = 3, aperture = 3, projection = "fuller")
d <- projection_distortion(gf, runif(1000, -180, 180),
                           asin(runif(1000, -1, 1)) * 180 / pi)
range(d$areal)
```

`projection_stages` *Points Through the Steps of Snyder's Projection*

Description

Snyder's equal-area projection of a face is Lambert's azimuthal equal-area projection about the face's centre, adjusted to fill the face's plane triangle. Lambert's projection has a construction with compass and ruler, and this function returns each point at every step of it, on one face.

Usage

```
projection_stages(x, lon, lat, face = NULL)
```

Arguments

`x`	A HexGridInfo object from `hex_grid` with Snyder's projection (`projection = "isea"`).
`lon, lat`	Longitudes and latitudes in degrees (geodetic on the grid's ellipsoid, if it has one: the construction starts from the point's place on the authalic sphere).
`face`	`NULL` to take each point onto the face it lies on, or face numbers from 0, one or one per point; the face's formulas are then applied as they stand to points off it.

Details

With T the face centre on the sphere and S a point at arc z from it and azimuth Az:

sphere S itself.

lambert S swings about T, in the plane of S and the normal at T, down onto the tangent plane at T, keeping its distance $|TS| = 2\sin(z/2)$: the Lambert point L, at azimuth Az. This map is equal-area and takes the spherical face to a triangle with curved edges.

nudge Within the tangent plane, Snyder's adjustment moves L to azimuth Az' and radius $2f\sin(z/2)$, which fill each of the face's six right triangles in the plane with area in proportion (Snyder 1992, eqs. 6-11).

snyder The tangent plane scales by R' about the sphere's centre onto the face plane, at distance R' from it; the point lands on Snyder's point P, at radius $2R'f\sin(z/2)$, and the face on the plane triangle of the face's area.

Plane coordinates (`u`, `v`) run from the face centre along the x and y axes of the face's plane triangle, the axes of `tx` and `ty`, in units of the sphere's radius; an azimuth a lies along $(\sin a, \cos a)$. Seen from the sphere's centre the planes are parallel, and the point's height along the face's normal is $\cos z$ for S, 1 on the tangent plane and R' on the face plane. The sphere point's `u` and `v` are its projection onto the planes along the normal.

The scale factors at every step come from `projection_distortion` with its `stage` argument; they and these points are computed by the same code as the forward projection, and `tx`, `ty` are those `hexify_forward` returns.

Value

A data frame with one row per point: `lon`, `lat`, `face` (from 0); `arc`, z in degrees; `az` (Az) and `az_prime` (Az') in degrees from the face's first vertex, clockwise seen from outside the sphere; Snyder's factor `f`; the plane coordinates `sphere_u`, `sphere_v`, `lambert_u`, `lambert_v`, `nudge_u`, `nudge_v`, `snyder_u`, `snyder_v`; and `tx`, `ty`, Snyder's point in triangle coordinates (a face edge is 1). The attribute `"r1"` holds R'.

References

Snyder, J. P. (1992). An equal-area map projection for polyhedral globes. *Cartographica* 29(1), 10-21. doi:10.3138/27H78K8848821752

See Also

`projection_distortion`; `hex_globe` with `lambert` to follow the construction on a globe

Examples

```
g <- hex_grid(resolution = 3, aperture = 3)
s <- projection_stages(g, c(0, 10, 16.37), c(40, 45, 48.21))
s

# The Lambert point keeps its distance from the face centre
cbind(lambert = sqrt(s$lambert_u^2 + s$lambert_v^2),
      chord = 2 * sin(s$arc * pi / 360))
```

`wall_metrics` *Spacing Metrics of Neighbouring Cells: Wall Length, Centre Distance and Cell Wall Midpoint Ratio*

Description

Measures every wall between a cell and an adjacent cell: how long it is, how far apart the two cell centres are, and how far the wall's midpoint lies from the line between the centres.

Usage

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

Arguments

`cell_id`	Cell IDs. When `NULL`, the cells of a HexData `grid`, or every cell of a HexGridInfo `grid`.
`grid`	A HexGridInfo or HexData object.

Details

The centre distance is the great-circle distance between the two cell centres. Gregory et al. (2008) measure the spread of a grid's intercell distances by their coefficient of variation.

The cell wall midpoint ratio (Gregory et al. 2008, after Heikes and Randall 1995) is the distance from the wall's midpoint to the midpoint of the great-circle arc joining the two centres, over the wall's length. It is 0 where the arc between the centres crosses the wall at its middle, as it does between two regular hexagons; a finite-volume scheme that evaluates a flux at the wall's midpoint is most accurate there. The wall's midpoint is the point halfway along the wall as the grid draws it (see `cell_metrics`).

Value

Data frame with one row per wall: `cell_id`, `neighbor_id`, `wall_km`, `center_distance_km` and `midpoint_ratio`. A wall between two of the given cells appears once, on the row of the cell given first; a wall to a cell not given appears on the given cell's row.

References

Gregory, M. J., Kimerling, A. J., White, D. and Sahr, K. (2008). A comparison of intercell metrics on discrete global grid systems. Computers, Environment and Urban Systems 32(3), 188-203. doi:10.1016/j.compenvurbsys.2007.11.003

Heikes, R. and Randall, D. A. (1995). Numerical integration of the shallow-water equations on a twisted icosahedral grid. Part II. A detailed description of the grid and an analysis of numerical accuracy. Monthly Weather Review 123(6), 1881-1887. doi:10.1175/15200493(1995)123<1881:NIOTSW>2.0.CO;2

See Also

cell_metrics for cell shape, get_neighbors

Examples

```
g <- hex_grid(resolution = 3, aperture = 3)
w <- wall_metrics(grid = g)
sd(w$center_distance_km) / mean(w$center_distance_km)
summary(w$midpoint_ratio)

# The walls around one cell
cell <- lonlat_to_cell(16.37, 48.21, g)
wall_metrics(cell, g)
```

Index

∗ **'dggridR' compatibility**
    as_dggrid, 4
    dggrid_is_compatible, 12
    from_dggrid, 14
∗ **datasets**
    hexify_world, 40
∗ **grid statistics**
    dgearthstat, 11
∗ **hexify main**
    hexify, 34
∗ **visualization**
    hexify_heatmap, 37
    plot_world, 79
[,HexData-method *(*HexData-class*)*, 23
[[,HexData-method *(*HexData-class*)*, 23
[[<-,HexData,ANY,missing-method
        *(*HexData-class*)*, 23
$,HexData-method *(*HexData-class*)*, 23
$,HexGridInfo-method
        *(*HexGridInfo-class*)*, 28
$<-,HexData-method *(*HexData-class*)*, 23

as.data.frame,HexData-method
        *(*HexData-class*)*, 23
as.list,HexData-method *(*HexData-class*)*,
        23
as.list,HexGridInfo-method
        *(*HexGridInfo-class*)*, 28
as_dggrid, 4
as_dggrid(), *12*, *14*
as_tibble.HexData *(*HexData-class*)*, 23

cell_area, 6, *8*, *22*
cell_metrics, 7, *83*, *84*
cell_to_index, *13*, *46*
cell_to_lonlat, 8, *62*, *72*
cell_to_lonlat(), *47*, *48*
cell_to_sf, 9, *25*, *39*
cell_to_sf(), *68*
cells, 5
cells,HexData-method *(*HexData-class*)*, 23

dg_closest_res_to_area(), *12*
dgearthstat, 11
dggrid_43h_sequence(), *5*, *12*, *14*
dggrid_is_compatible, 12
dggrid_is_compatible(), *5*, *14*
dggrs_definition, 13
dim,HexData-method *(*HexData-class*)*, 23

format.dggs *(*as_dggrid*)*, 4
from_dggrid, 14
from_dggrid(), *5*, *12*

get_children(), *46*, *67*
get_neighbors, 15, *66*, *84*
get_neighbors(), *47*, *48*, *64*, *65*, *71*
get_parent, *42–44*
get_parent(), *46*
globe_centers, 16, *30*, *51*
grid_clip, 17, *78*
grid_global, *18*, 18, *20*, *30*, *34*
grid_info, 19
grid_info,HexData-method
        *(*HexData-class*)*, 23
grid_rect, *18*, *19*, 20

h3_crosswalk, *7*, 21, *69*
hcl.colors, *51*
hex9_warp_download, 22, *52*, *55*, *59*
hex_aggregate, 42
hex_browse, 44
hex_compact, 46
hex_compact(), *67*
hex_distance, 47
hex_distance(), *16*
hex_extract, 48
hex_extract(), *67*, *68*
hex_globe, 49, *53*, *82*
hex_globe_png, 53

hex_grid, *4*, *7*, *11*, *13*, *22*, *32*, *33*, *35*, *36*, 54, *69*, *72*, *74*, *78*, *80*, *81*
hex_rays, 60, *66*
hex_sample, 62
hex_smooth, 63
hex_summarize, *44*, 64
hex_triangles, *62*, 65
hex_uncompact, 66
hex_uncompact(), *46*
hex_zonal, 67
hex_zonal(), *48*, *49*
HexData-class, 23
HexGridInfo-class, 28
hexify, *26*, *27*, 34, *60*
hexify(), *16*, *49*, *64*, *65*
hexify-package, 3
hexify_area_to_eff_res(), *12*
hexify_cell_to_plane, *33*, *75*
hexify_compare_resolutions(), *12*
hexify_eff_res_to_area(), *12*
hexify_eff_res_to_resolution(), *12*
hexify_forward, *82*
hexify_grid(), *36*
hexify_heatmap, *27*, 37
hexify_heatmap(), *45*, *79*
hexify_resolution_to_eff_res(), *12*
hexify_world, *30*, 40, *51*, *74*

import_h3, 68
is_hex_data, 69
is_hex_grid, 70
is_pentagon, 71

lonlat_to_cell, *9*, *10*, *52*, *62*, 72

n_cells, 77
n_cells,HexData-method *(*HexData-class*)*, 23
n_cells,HexGridInfo-method *(*HexGridInfo-class*)*, 28
names,HexData-method *(*HexData-class*)*, 23
names,HexGridInfo-method *(*HexGridInfo-class*)*, 28
ncol,HexData-method *(*HexData-class*)*, 23
net_cells, *34*, 73
net_layout, *31*, *33*, *57*, *73*, 74, *76*
net_project, *73*, *74*, *76*, 76
nrow,HexData-method *(*HexData-class*)*, 23

pdf, *33*
plot, *16*, *52*, *54*, *74*, *81*
plot,HexData,missing-method *(*HexData-class*)*, 23
plot,HexGridInfo,missing-method *(*HexGridInfo-class*)*, 28
plot_grid, *39*, 77
plot_world, 79
plot_world(), *39*
print.dggs *(*as_dggrid*)*, 4
print.hexify_data_summary *(*HexData-class*)*, 23
print.hexify_grid_summary *(*HexGridInfo-class*)*, 28
projection_distortion, *33*, *34*, 80, *82*
projection_stages, *52*, *80*, *81*, 81

show,HexData-method *(*HexData-class*)*, 23
show,HexGridInfo-method *(*HexGridInfo-class*)*, 28
st_as_sf, *11*, *36*
st_as_sf(), *45*
st_as_sf.HexData *(*HexData-class*)*, 23
st_as_sf.HexGridInfo *(*HexGridInfo-class*)*, 28
st_crs.HexData *(*HexData-class*)*, 23
st_crs.HexGridInfo *(*HexGridInfo-class*)*, 28
summary,HexData-method *(*HexData-class*)*, 23
summary,HexGridInfo-method *(*HexGridInfo-class*)*, 28

wall_metrics, *8*, *66*, 83