Package 'h3o'

August 29, 2025

Title H3 Geospatial Indexing System

Version 0.3.0

Description A dependency free interface to the H3 geospatial indexing system utilizing the Rust library 'h3o' <https://github.com/HydroniumLabs/h3o> via the 'extendr' library <https://github.com/extendr/extendr>.

License MIT + file LICENSE

Encoding UTF-8

Language en

RoxygenNote 7.3.2

SystemRequirements Cargo (Rust's package manager), rustc >= 1.75

Imports rlang, stats, vctrs

Suggests sf, wk

Config/rextendr/version 0.4.0.9000

Depends R (>= 4.2)

URL https://github.com/extendr/h3o, https://extendr.rs/h3o/

BugReports https://github.com/extendr/h3o/issues

NeedsCompilation yes

Author Josiah Parry [aut, cph] (ORCID:
<https://orcid.org/0000-0001-9910-865X>),
Kenneth Vernon [cre, ctb] (ORCID:
<https://orcid.org/0000-0003-0098-5092>)

Maintainer Kenneth Vernon <kenneth.b.vernon@gmail.com>

Repository CRAN

Date/Publication 2025-08-29 11:00:02 UTC

Contents

`compact_cells` *Compact H3 Cells*

Description

Reduce a set of H3 indices of the same resolution to the minimum number of H3 indices of varying resolution that entirely covers the input area.

Usage

```
compact_cells(x)

uncompact_cells(x, resolution)
```

Arguments

`x`	a vector of H3 indexes.
`resolution`	a scalar integer representing the grid resolution in the range [0, 15].

Value

An `H3` vector.

Examples

```
x <- h3_from_strings("841f91dfffffffff")
y <- uncompact_cells(x, 5)[[1]]
z <- compact_cells(y)
all.equal(x, z)
```

get_parents *Hierarchical H3 Grid Functions*

Description

Functions used to traverse the hierarchy of H3 grids.

Usage

```
get_parents(x, resolution)

get_children(x, resolution)

get_children_count(x, resolution)

get_children_center(x, resolution)

get_children_position(x, resolution)

get_children_at(x, position, resolution)
```

Arguments

x	an H3 vector.
resolution	a scalar integer representing the grid resolution in the range [0, 15].
position	the integer position in the ordered set of cells.

Details

- `get_parents()`: returns the parent cells for an H3 vector at a given resolution. Errors if the resolution is smaller than the provided cell.
- `get_children()`: returns a list of H3 vectors containing the children of each H3 cell at a specified resolution. If the resolution is greater than the cell's resolution an empty vector is returned.
- `get_children_count()`: returns an integer vector containing the number of children for each cell at the specified resolution.
- `get_children_center()`: returns the middle child (center child) for all children of an H3 cell at a specified resolution as an H3 vector.
- `get_children_position()`: returns the position of the observed H3 cell in an ordered list of all children as a child of a higher resolution cell (PR for clearer language welcome).
- `get_children_at()`: returns the child of each H3 cell at a specified resolution based on its position in an ordered list (PR for clearer language welcome).

Value

See details.

Examples

```
h3_strs <- c("841f91dfffffffff", "841fb59fffffffff")
h3 <- h3_from_strings(h3_strs)

get_parents(h3, 3)
get_children(h3, 5)
get_children_count(h3, 6)
get_children_position(h3, 3)
get_children_at(h3, 999, 10)
```

---

grid_disk *Grid Traversal*

Description

Functions used to traverse the H3 grid.

Usage

```
grid_disk(x, k = 1, safe = TRUE)

grid_ring(x, k = 1)

grid_distances(x, k = 1)

grid_path_cells(x, y)

grid_path_cells_size(x, y)

grid_distance(x, y)

grid_local_ij(x, y)
```

Arguments

x	an H3 vector.
k	the order of ring neighbors. 0 is the focal location (the observed H3 index). 1 is the immediate neighbors of the H3 index. 2 is the neighbors of the 1st order neighbors and so on.
safe	default TRUE. If FALSE uses the fast algorithm which can fail.
y	an H3 vector.

Details

- `grid_disk()`: returns the disk of cells for the identified K ring. It is a disk because it returns all cells to create a complete geometry without any holes. See `grid_ring()` if you do not want inclusive neighbors.
- `grid_ring()`: returns a K ring of neighbors around the H3 cell.
- `grid_distances()`: returns a list of numeric vectors indicating the network distances between neighbors in a K ring. The first element is always 0 as the travel distance to one's self is 0. If the H3 index is missing a 0 length vector will be returned.
- `grid_path_cells()`: returns a list of `H3` vectors indicating the cells traversed to get from `x` to `y`. If either `x` or `y` are missing, an empty vector is returned.
- `grid_path_cells_size()`: returns an integer vector with the cell path distance between pairwise elements of `x` and `y`. If either `x` or `y` are missing the result is `NA`. `grid_distance()`: returns an integer vector with the network distance between pairwise elements of `x` and `y`. If either `x` or `y` are missing the result is `NA`. Effectively `grid_path_cells_size() - 1`.
- `grid_local_ij()` returns a two column data frame containing the columns `i` and `j` which correspond to the i,j coordinate directions to the destination cell.

Value

See details.

Examples

```
h3_strs <- c("841f91dfffffffff", "841fb59fffffffff")
h3 <- h3_from_strings(h3_strs)

grid_disk(h3, 1)
grid_ring(h3, 2)
grid_distances(h3, 2)
grid_path_cells(h3, rev(h3))
grid_path_cells_size(h3, rev(h3))
grid_distance(h3, rev(h3))
grid_local_ij(h3, rev(h3))
```

`h3_edges` *H3 Edges*

Description

Functions to create or work with `H3Edge` vectors. See `Details` for further details.

Usage

```
h3_edges(x, flat = FALSE)

h3_shared_edge_sparse(x, y)

h3_shared_edge_pairwise(x, y)

is_edge(x)

is_valid_edge(x)

h3_edges_from_strings(x)

flatten_edges(x)

h3_edge_cells(x)

h3_edge_origin(x)

h3_edge_destination(x)

## S3 method for class 'H3Edge'
as.character(x, ...)
```

Arguments

x	an H3 vector
flat	default FALSE. If TRUE return a single vector combining all edges of all H3 cells.
y	an H3 vector
...	unused.

Details

- h3_edges(): returns a list of H3Edge vectors for each H3 index. When flat = TRUE, returns a single H3Edge vector.
- h3_shared_edge_pairwise(): returns an H3Edge vector of shared edges. If there is no shared edge NA is returned.
- h3_shared_edge_sparse(): returns a list of H3Edge vectors. Each element iterates through each element of y checking for a shared edge.
- is_edge(): returns TRUE if the element inherits the H3Edge class.
- is_valid_edge(): checks each element of a character vector to determine if it is a valid edge ID.
- h3_edges_from_strings(): create an H3Edge vector from a character vector.
- flatten_edges(): flattens a list of H3Edge vectors into a single H3Edge vector.
- h3_edge_cells(): returns a list of length 2 named H3Edge vectors of origin and destination cells

- h3_edge_origin(): returns a vector of H3Edge origin cells
- h3_edge_destination(): returns a vector of H3Edge destination cells

Value

See details.

Examples

```
# create an H3 cell
x <- h3_from_xy(-122, 38, 5)

# find all edges and flatten
edges <- h3_edges(x) |>
  flatten_edges()

# check if they are all edges
is_edge(edges)

# check if valid edge strings
is_valid_edge(c("115e22da7fffffff", "abcd"))

# get the origin cell of the edge
h3_edge_origin(edges)

# get the destination of the edge
h3_edge_destination(edges)

# get both origin and destination cells
h3_edge_cells(edges)

# create edges from strings
h3_edges_from_strings(c("115e22da7fffffff", "abcd"))

# create a vector of cells
cells_ids <-c(
  "85e22da7fffffff", "85e35ad3fffffff",
  "85e22daffffffff", "85e35adbffffffff",
  "85e22da3fffffff"
)

cells <- h3o::h3_from_strings(cells_ids)

# find shared edges between the two pairwise
h3_shared_edge_pairwise(cells, rev(cells))

# get the sparse shared eddge. Finds all possible shared edges.
h3_shared_edge_sparse(cells, cells)
```

h3_from_xy *Create H3 Index*

Description

Create H3 indices from sfc objects, vectors of x and y coordinates, or H3 string IDs.

Usage

```
h3_from_xy(x, y, resolution)

h3_from_points(x, resolution)

h3_from_strings(x)

h3_to_points(x)

h3_to_vertexes(x)

## S3 method for class 'H3'
as.character(x, ...)

flatten_h3(x)

is_h3(x)
```

Arguments

x	for h3_from_points() an object of class sfc_POINT. For h3_from_strings() a character vector of H3 index IDs. For h3_from_xy() a numeric vector of longitudes.
y	a numeric vector of latitudes.
resolution	an integer indicating the H3 cell resolution. Must be between 0 and 15 inclusive.
...	unused.

Details

- h3_from_points(): takes an sfc_POINT object and creates a vector of H3 cells
- h3_from_strings(): converts a character vector of cell indexes to an H3 vector
- h3_from_xy(): converts vectors of x and y coordinates to an H3 vector
- h3_to_points(): converts an H3 vector to a either an sfc_POINT object or a list of sfg POINT objects.
- h3_to_vertexes(): converts an H3 vector to an sfc_MULTIPOINT object or a list of MULTIPOINT objects.

Value

See details.

Examples

```
h3_from_xy(-90, 120, 5)

h3_from_strings("85f29383fffffff")

if (requireNamespace("sf")) {
  # create random points
  pnts <- sf::st_cast(
    sf::st_sfc(
      sf::st_multipoint(matrix(runif(10, max = 90), ncol = 2)),
      crs = 4326
    ),
    "POINT"
  )

  # convert to H3 objects
  h3s <- h3_from_points(pnts, 5)

  h3_to_vertexes(h3s)

  h3_to_points(h3s)
}

h3_ids <- c("831f91fffffffff", "831fb5fffffffff", "831f94fffffffff")

flatten_h3(
  list(
    NULL,
    h3_from_strings(h3_ids),
    h3_from_strings(h3_ids[1])
  )
)
```

---

h3_resolution *H3 Inspection Functions*

Description

Functions that provide metadata about H3 indexes.

Usage

```
h3_resolution(x)

h3_base_cell(x)
```

```
is_valid_h3(x)

is_res_class_iii(x)

is_pentagon(x)

get_face_count(x)
```

Arguments

x	an H3 vector.

Details

- `h3_resolution()`: returns the resolution of each H3 cell.
- `h3_base_cell()`: returns the base cell integer.
- `is_valid_h3()`: given a vector of H3 index string IDs, determine if they are valid.
- `is_res_class_iii()`: determines if an H3 cell has Class III orientation.
- `is_pentagon()`: determines if an H3 cell is one of the rare few pentagons.
- `get_face_count()`: returns the number of faces that intersect with the H3 index.

Value

See details.

Examples

```
cells_ids <-c(
    "85e22da7fffffff", "85e35ad3fffffff",
    "85e22daffffffff", "85e35adbfffffff",
    "85e22db7fffffff", "85e35e6bfffffff",
    "85e22da3fffffff"
  )

cells <- h3o::h3_from_strings(cells_ids)

h3_resolution(cells)
h3_base_cell(cells)
is_valid_h3(c("85e22db7fffffff", NA, "oopsies"))
is_res_class_iii(cells)
is_res_class_iii(h3_from_xy(0, 0, 10))
is_pentagon(h3_from_strings("08FD600000000000"))
get_face_count(cells)
```

is_nb_pairwise *H3 index neighbors*

Description

Test if two H3 cells are neighbors.

Usage

```
is_nb_pairwise(x, y)

is_nb_sparse(x, y)
```

Arguments

x	an H3 vector.
y	and H3 vector.

Value

`is_nb_pairwise()` returns a logical vector wheraas `is_nb_sparse()` returns a list with logical vector elements.

Examples

```
cells_ids <-c(
  "85e22da7fffffff", "85e35ad3fffffff",
  "85e22daffffffff", "85e35adbfffffff",
  "85e22db7fffffff", "85e35e6bfffffff",
  "85e22da3fffffff"
)

cells <- h3o::h3_from_strings(cells_ids)

is_nb_pairwise(cells, rev(cells))
is_nb_sparse(cells, cells)
```

sfc_to_cells *Convert sf geometry to H3 Cells*

Description

Given a vector of sf geometries (class `sfc`) create a list of H3 vectors. Each list element contains the vector of H3 cells that cover the geometry.

Usage

```
sfc_to_cells(x, resolution, containment = "intersect")
```

Arguments

x	for h3_from_points() an object of class sfc_POINT. For h3_from_strings() a character vector of H3 index IDs. For h3_from_xy() a numeric vector of longitudes.
resolution	an integer indicating the H3 cell resolution. Must be between 0 and 15 inclusive.
containment	default "intersect". Must be one of "intersect", "centroid", or "boundary". See details.

Details

Note, use flatten_h3() to reduce the list to a single vector.

The Containment Mode determines if an H3 cell should be returned.

- "centroid" returns every cell whose centroid are contained inside of a polygon. This is the fastest option but may not cover the entire polygon.
- "boundary" this returns the cells which are completely contained by the polygon. Much of a polygon might not be covered using this approach.
- "intersect" ensures that a polygon is entirely covered. If an H3 cell comes in contact with the polygon it will be returned. This is the default.
- "contains" behaves the same as "intersect", but also handles the case where the geometry is being covered by a cell without intersecting with its boundaries. In such cases, the covering cell is returned.

Value

An H3 vector.

Examples

```
if (interactive() && rlang::is_installed("sf")) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geo <- sf::st_geometry(nc)
  cells <- sfc_to_cells(geo, 5)

  head(cells)

  plot(flatten_h3(cells))
}
```

Index