Package 'ICESat2VegR'

September 18, 2026

Type Package

Title ICESat-2 Data Analysis for Land and Vegetation

Version 0.0.2

Description Provides tools for downloading, reading, processing, visualizing, and exporting NASA's ICESat-2 ATL03 (Global Geolocated Photon Data) and ATL08 (Land and Vegetation Height) products. Supports photon- and segment-level analysis, spatial sampling, gridding, statistical and machine-learning modeling, and integration with 'Google Earth Engine' (<https://earthengine.google.com/>) for wall-to-wall mapping of vegetation structure and other land attributes.

Depends R (>= 4.1.0)

Imports curl, data.table, sf, fs, getPass, geojsonsf, googledrive, googleCloudStorageR, grDevices, jsonlite, httr2, magrittr, mathjaxr, methods, Rcpp, Rdpack, R6, randomForest, reticulate, terra, dplyr, servr, stringr, xml2

Encoding UTF-8

URL https://github.com/carlos-alberto-silva/ICESat2VegR

BugReports https://github.com/carlos-alberto-silva/ICESat2VegR/issues

NeedsCompilation yes

RdMacros Rdpack, mathjaxr

BuildManual TRUE

Suggests chromote, devtools, ggplot2, grid, gridExtra, hdf5r, htmlwidgets, knitr, leaflet, leafsync, lwgeom, mapview, png, rmarkdown, rstudioapi, signal, stars, testthat (>= 3.0.0), webshot, viridis, withr

LinkingTo Rcpp

SystemRequirements - GDAL (>= 3.0.0) - PROJ (>= 6.0.0) - HDF5 (>= 1.8.13)

Collate 'ANNIndex.R' 'lazy_applier.R'
'ATL03_ATL08_compute_seg_attributes_dt_segStat.R' 'utmTools.R'
'lasTools.R' 'ATL03_ATL08_photons_attributes_dt_LAS.R'
'ATL03_ATL08_photons_attributes_dt_clipBox.R'
'ATL03_ATL08_photons_attributes_dt_clipGeometry.R'
'ATL03_ATL08_photons_attributes_dt_gridStat.R'
'ATL03_ATL08_photons_attributes_dt_join.R'
'ATL03_ATL08_photons_attributes_dt_polyStat.R'
'ATL03_ATL08_photons_seg_dt_fitground.R'
'ATL03_ATL08_photons_seg_dt_height_normalize.R'
'ATL03_ATL08_seg_attributes_dt_clip.R'
'ATL03_ATL08_seg_cover_dt_compute.R'
'ATL03_ATL08_segment_create.R' 'ATL03_h5_clip.R' 'clipTools.R'
'ATL03_h5_clipBox.R' 'ATL03_h5_clipGeometry.R'
'ATL03_photons_attributes_dt.R'
'ATL03_photons_attributes_dt_LAS.R'
'ATL03_photons_attributes_dt_clipBox.R'
'ATL03_photons_attributes_dt_clipGeometry.R' 'class_tools.R'
'class.icesat2.R' 'zzz.R' 'ATL03_read.R'
'ATL03_seg_metadata_dt.R' 'ATL08_read.R' 'ATL08_h5_clip.R'
'ATL08_h5_clipBox.R' 'ATL08_h5_clipGeometry.R'
'ATL08_photons_attributes_dt.R'
'ATL08_photons_attributes_dt_LAS.R' 'ATL08_seg_attributes_dt.R'
'ATL08_seg_attributes_dt_LAS.R'
'ATL08_seg_attributes_dt_clipBox.R'
'ATL08_seg_attributes_dt_clipGeometry.R'
'ATL08_seg_attributes_dt_gridStat.R'
'ATL08_seg_attributes_dt_polyStat.R'
'ATL08_seg_attributes_h5_gridStat.R' 'ATLAS_dataDownload.R'
'ATLAS_dataFinder.R' 'earthaccess.R' 'ATLAS_dataFinder_cloud.R'
'ATLAS_dataFinder_direct.R' 'ICESat2VegR-package.R'
'ICESat2VegR_configure.R' 'addEEImage.R' 'argParse.R'
'class.icesat2.h5ds_cloud.R' 'class.icesat2.h5_cloud.R'
'class.icesat2.h5ds_local.R' 'class.icesat2.h5_local.R'
'clip.R' 'ee_build_AlphaEarth_embedding_terrain_stack.R'
'ee_build_hls_s1c_terrain_stack.R' 'extract.R' 'fit_metrics.R'
'fit_model.R' 'gdalBindings.R' 'gee-base-feature.R'
'gee-base-list.R' 'gee-base-number.R' 'gee-base.R'
'gee-search.R' 'globals.R' 'map_tools.R' 'model_tools.R'
'plot_icesat2_orbit_animation.R' 'predict_h5.R'
'rasterize_h5.R' 'rgt_extract.R' 'sample.R' 'to_vect.R'
'varSel.R' 'vect_as_ee.R'

License GPL (>= 3)

Config/roxygen2/version 8.1.0

Author Carlos Alberto Silva [aut, cph, cre],
Caio Hamamura [aut, cph],
Cesar Alvites [aut, ctb],
Alexander J. Gaskins [aut, ctb],

Sunil Arya [ctb, cph] (Author of the bundled ANN library (src/)),
David Mount [ctb, cph] (Author of the bundled ANN library (src/)),
University of Maryland [cph] (Copyright holder of the bundled ANN library (src/)),
Chuck Gantz [ctb] (Author of the lat/long to UTM conversion algorithm used in R/utmTools.R),
Cole Krehbiel [ctb] (Author of the NASA data download routine adapted in R/ATLAS_dataDownload.R)

Maintainer Carlos Alberto Silva <c.silva@ufl.edu>

Repository CRAN

Date/Publication 2026-09-18 06:50:02 UTC

Contents

`ICESat2VegR-package` *ICESat2VegR: NASA ICESat-2 data for land & vegetation*

Description

Tools to download, read, process, model, and visualize ICESat-2 ATL03/ATL08 for land and vegetation applications.

Author(s)

Maintainer: Carlos Alberto Silva <c.silva@ufl.edu> [copyright holder]

Authors:

- Carlos Alberto Silva <c.silva@ufl.edu> [copyright holder]
- Caio Hamamura <caiohamamura@gmail.com> [copyright holder]
- Cesar Alvites <c.alvitesdiaz@ufl.edu> [contributor]
- Alexander J. Gaskins <alexandergaskins@ufl.edu> [contributor]

Other contributors:

- Sunil Arya (Author of the bundled ANN library (src/)) [contributor, copyright holder]
- David Mount (Author of the bundled ANN library (src/)) [contributor, copyright holder]
- University of Maryland (Copyright holder of the bundled ANN library (src/)) [copyright holder]
- Chuck Gantz (Author of the lat/long to UTM conversion algorithm used in R/utmTools.R) [contributor]
- Cole Krehbiel (Author of the NASA data download routine adapted in R/ATLAS_dataDownload.R) [contributor]

See Also

Useful links:

- https://github.com/carlos-alberto-silva/ICESat2VegR
- Report bugs at https://github.com/carlos-alberto-silva/ICESat2VegR/issues

`.as_ee_geom` *Convert R geometry objects to an Earth Engine geometry*

Description

`.as_ee_geom()` converts a variety of R geometry representations (including `sf`, `sfc`, `terra` objects, bounding boxes, WKT/GeoJSON, and even existing Earth Engine python objects) into a Python `ee$Geometry` suitable for use in Earth Engine workflows.

This helper is designed to be flexible and permissive: it accepts most common spatial formats used in R and normalizes them into a standard Earth Engine geometry. When a buffer distance is supplied, the geometry is optionally buffered in meters on the Earth Engine side.

Usage

```
.as_ee_geom(geom, xcol = "lon", ycol = "lat", crs = 4326, buffer_m = NULL)
```

Arguments

`geom`	Geometry input. Supported types include: • An Earth Engine python object (e.g. `ee$Geometry`, `ee$Feature`, `ee$FeatureCollection`). Features/FeatureCollections are reduced to their geometry via `$geometry()`. • An `sf` or `sfc` object. • A `terra::SpatVector`. • A `terra::SpatExtent` (or numeric vector of length 4 specifying a bounding box as `c(xmin, ymin, xmax, ymax)`). • A `data.frame` with longitude/latitude columns (`xcol`, `ycol`). • A single-character WKT string. • A single-character GeoJSON string. • A parsed GeoJSON list with a non-`NULL` `type` element.
`xcol`	Character. Name of the longitude column when `geom` is a `data.frame`. Default is `lon`.
`ycol`	Character. Name of the latitude column when `geom` is a `data.frame`. Default is `lat`.
`crs`	Coordinate reference system of the input geometry when `geom` is an `sf`/`sfc` object or a `data.frame`. Can be any `sf`-compatible CRS specification (default is EPSG 4326).
`buffer_m`	Optional numeric. Buffer distance in meters applied to the resulting Earth Engine geometry. If `NULL` (default) no buffering is applied. For sf/data.frame inputs, buffering is done on the EE side using `ee$Geometry$buffer()`.

Details

For sf/sfc and data.frame inputs, geometries are first transformed to EPSG:4326 and written to a temporary GeoJSON file, which is then read and wrapped into an ee$FeatureCollection(...)$geometry(). For terra::SpatVector and terra::SpatExtent inputs, conversion proceeds via terra::writeVector() / terra::as.polygons() and an EE rectangle geometry, respectively.

Existing Earth Engine python objects are returned as-is, except that ee$Feature and ee$FeatureCollection objects are coerced to their underlying geometry via $geometry().

Value

A Python ee$Geometry object (or compatible geometry-like EE object) suitable for use in Earth Engine operations.

Examples

```
## Not run:
  ee <- reticulate::import("ee", delay_load = FALSE)

  # 1) From sf polygon
  library(sf)
  poly <- st_as_sfc(st_bbox(c(
    xmin = -82.4, xmax = -82.2,
    ymin =  29.6, ymax =  29.8
  ), crs = 4326))

  ee_geom1 <- .as_ee_geom(poly)


  # 2) From terra::SpatVector
  library(terra)
  v <- vect(system.file("extdata", "clip_geom.shp", package = "ICESat2VegR"))
  ee_geom2 <- .as_ee_geom(v, buffer_m = 30)


  # 3) From numeric extent (xmin, ymin, xmax, ymax)
  bbox_vec <- c(-82.4, 29.6, -82.2, 29.8)
  ee_geom3 <- .as_ee_geom(bbox_vec)


  # 4) From data.frame of points
  df <- data.frame(
    lon = c(-82.3, -82.25),
    lat = c(29.65, 29.7)
  )
  ee_geom4 <- .as_ee_geom(df, buffer_m = 1000)

## End(Not run)
```

.ee_ping *Check if Google Earth Engine is initialized*

Description

`.ee_ping()` verifies that the Earth Engine Python API is initialized and responsive. It attempts to evaluate a trivial Earth Engine expression (`ee$Image$constant(1)$getInfo()`) and returns invisibly if successful. If the call fails, an error is raised with a hint to authenticate and initialize Earth Engine.

Usage

```
.ee_ping(ee)
```

Arguments

`ee`	The Earth Engine Python module as returned by `reticulate::import("ee")`.

Value

Invisibly returns `TRUE` if Earth Engine is initialized and responsive. Otherwise, an error is thrown indicating that Earth Engine must be authenticated and initialized.

Examples

```
## Not run:
  library(reticulate)
  ee <- import("ee", delay_load = FALSE)

  # Will error if EE is not authenticated/initialized
  .ee_ping(ee)

## End(Not run)
```

addEEImage *Add an Earth Engine Image to a leaflet map*

Description

Add an Earth Engine Image to a leaflet map

Usage

```
addEEImage(
  map,
  img,
  bands = NULL,
  min_value = 0,
  max_value = 1,
  palette = c("#d55e00", "#cc79a7", "#f0e442", "#0072b2", "#009e73"),
  group = NULL,
  aoi = NULL,
  ...
)
```

Arguments

map	A leaflet::leaflet widget.
img	An Earth Engine image (reticulate python.builtin.object with $visualize, $select).
bands	Character or numeric vector (length 1 or 3). If NULL, uses all bands from the image.
min_value	Numeric. Minimum visualization range.
max_value	Numeric. Maximum visualization range.
palette	Character vector of colors for single-band rendering.
group	Optional overlay group name.
aoi	Optional EE geometry to clip before rendering.
...	Passed to leaflet::addTiles() (e.g., options = leaflet::tileOptions(opacity = 0.8)).

Value

The input leaflet map with the EE image layer added.

Examples

```
## Not run:
# Initialize Earth Engine (use your own project id)
Sys.setenv(EE_PROJECT = "your-ee-project-id")
ee_initialize()

# Harmonized Landsat-Sentinel (HLS) surface reflectance collection
collection_id <- "NASA/HLS/HLSL30/v002"
ee_collection <- ee$ImageCollection(collection_id)

# Mask cloud, cloud-shadow and adjacent-cloud pixels (Fmask bits 1-3)
cloudMask <- 2^1 + 2^2 + 2^3
hlsMask <- function(image) {
  image$updateMask(!(image[["Fmask"]] & cloudMask))
```

```
}

# Cloud-free median composite over two summer seasons
image <- c(
  ee_collection$filterDate("2020-04-01", "2020-07-31")$map(hlsMask),
  ee_collection$filterDate("2021-04-01", "2021-07-31")$map(hlsMask)
)$filter("CLOUD_COVERAGE < 30")$median()

# Show the RGB composite (bands 3-2-1) on an interactive map
if (require("leaflet")) {
  leaflet() %>%
    addEEImage(
      image,
      bands = c(3, 2, 1),
      min_value = 0.001,
      max_value = 0.2
    ) %>%
    setView(lng = -82.2345, lat = 29.6552, zoom = 10)
}

## End(Not run)
```

`ANNIndex` *ANNIndex Class*

Description

This class uses Approximate Neareast Neighbor Index for finding points that are within a specified range.

Public fields

`tree` The C++ pointer for the built tree ANNIndex constructor

Methods

Public methods:

- `ANNIndex$new()`
- `ANNIndex$searchFixedRadius()`

`ANNIndex$new()`**:** Creates a new instance of the `ANNIndex` class.

Usage:

```
ANNIndex$new(x, y)
```

Arguments:

`x` NumericVector of x/longitude

`y` NumericVector of y/latitude
searchFixedRadius method

`ANNIndex$searchFixedRadius()`: Given a x, y point get the indexes that are within a specified radius.

Usage:

```
ANNIndex$searchFixedRadius(x, y, radius)
```

Arguments:

`x` NumericVector of x/longitude

`y` NumericVector of y/latitude

`radius` the minimum radius between the points

See Also

Mount, D. M.; Arya, S. ANN: A Library for Approximate Nearest Neighbor Searching, available in https://www.cs.umd.edu/~mount/ANN/

`aspect` *Compute terrain aspect (degrees) from a DEM image*

Description

Computes the terrain *aspect* in degrees for each pixel of an Earth Engine `ee$Image` representing a digital elevation model (DEM).

Aspect describes the downslope direction of the steepest gradient and is expressed in degrees clockwise from north. The computation is performed using Earth Engine's built-in `ee$Terrain$aspect()` function. As with slope, Earth Engine uses the 4-connected neighborhood, so missing values may occur near image edges.

Usage

```
aspect(x)
```

Arguments

`x`	An `ee$Image` representing a DEM from which aspect will be derived.

Value

An `ee$Image` with one band named `aspect` containing aspect values in degrees clockwise from north.

Examples

```
## Not run:
  ee <- reticulate::import("ee")
  dem <- ee$Image("NASA/NASADEM_HGT/001")
  asp <- aspect(dem)

## End(Not run)
```

atan2.ee.ee_number.Number

Calculates the angle formed by the 2D vector [x, y]

Description

Calculates the angle formed by the 2D vector [x, y]

Usage

```
atan2.ee.ee_number.Number(x, e2)
```

Arguments

x	An ee$Number
e2	An ee$Number

Value

An ee$Number representing the angle in radians

See Also

https://developers.google.com/earth-engine/apidocs/ee-number-atan2

ATL03_ATL08_compute_seg_attributes_dt_segStat

Statistics of ATL03 and ATL08 labeled photons at the segment level

Description

Computes a series of statistics from ATL03 and ATL08 labeled photons within a given segment length.

Usage

```
ATL03_ATL08_compute_seg_attributes_dt_segStat(
  atl03_atl08_seg_dt,
  list_expr,
  ph_class = c(0, 1, 2, 3),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = NULL,
  night_flag = NULL
)
```

Arguments

atl03_atl08_seg_dt	An S4 object of class icesat2.atl03_atl08_seg_dt containing ATL03 and ATL08 data (output of ATL03_ATL08_photons_attributes_dt_join() function).
list_expr	The function to be applied for computing the defined statistics
ph_class	Character vector indicating photons to process based on the classification (1=ground, 2=canopy, 3=top canopy), Default is c(2,3)
beam	Character vector indicating beams to process. Default is c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
quality_ph	Indicates the quality of the associated photon. 0 = nominal, 1 = possible_afterpulse, 2 = possible_impulse_response_ effect, 3=possible_tep. Default is 0
night_flag	Flag indicating the data were acquired in night conditions: 0=day, 1=night. Default is 1

Value

Returns an S4 object of class icesat2.atl08_dt Containing Statistics of ATL03 and ATL08 labeled photons

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying ATL03 and ATL08 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 labeled photons
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

# Computing the max canopy height at 30 m segments
atl03_atl08_dt_seg <- ATL03_ATL08_segment_create(atl03_atl08_dt, segment_length = 30)

max_canopy <- ATL03_ATL08_compute_seg_attributes_dt_segStat(atl03_atl08_dt_seg,
  list_expr = max(ph_h),
  ph_class = c(2, 3),
```

```
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = 0,
  night_flag = 0
)

head(max_canopy)

# Computing a series of canopy height statistics from customized list expressions
canopy_metrics <- ATL03_ATL08_compute_seg_attributes_dt_segStat(atl03_atl08_dt_seg,
  list_expr = list(
    max_ph_elevation = max(h_ph),
    h_canopy = quantile(ph_h, 0.98),
    n_canopy = sum(classed_pc_flag == 2),
    n_top_canopy = sum(classed_pc_flag == 3)
  ),
  ph_class = c(2, 3),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = 0,
  night_flag = 0 # there are no night photons in this dataset
)

head(canopy_metrics)

close(atl03_h5)
close(atl08_h5)

}
```

`ATL03_ATL08_photons_attributes_dt_clipBox` *Clip joined ATL03 and ATL08 photons by bounding extent*

Description

Clips joined ATL03 and ATL08 photon attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
ATL03_ATL08_photons_attributes_dt_clipBox(atl03_atl08_dt, clip_obj)
```

Arguments

`atl03_atl08_dt`	An S4 object of class `icesat2.atl03atl08_dt` containing joined ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).

`clip_obj` Bounding extent used to perform the clipping. Supported inputs:

- Numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees.
- `SpatExtent` (package **terra**): the extent is used directly.

Details

When `clip_obj` is a `SpatExtent`, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns an S4 object of class `icesat2.atl03atl08_dt` containing a subset of the joined ATL03 and ATL08 photon attributes restricted to the clipping extent.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5",
                          package = "ICESat2VegR")

atl08_path <- system.file("extdata", "atl08_clip.h5",
                          package = "ICESat2VegR")

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Joining ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# 1) Using a numeric bounding box: xmin ymin xmax ymax
bbox <- c(-106.571, 41.531, -106.569, 41.540)

atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt = atl03_atl08_dt,
  clip_obj = bbox
)
head(atl03_atl08_dt_clip)

# 2) Using a SpatExtent (example)
ext <- terra::ext(-106.57, -106.569, 41.531, 41.540)
```

```
atl03_atl08_dt_clip_ext <- ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt = atl03_atl08_dt,
  clip_obj = ext
)

close(atl03_h5)
close(atl08_h5)

}
```

`ATL03_ATL08_photons_attributes_dt_clipGeometry`
Clip Joined ATL03 and ATL08 by Geometry

Description

This function clips joined ATL03 and ATL08 photon attributes within a given geometry.

Usage

```
ATL03_ATL08_photons_attributes_dt_clipGeometry(
  atl03_atl08_dt,
  clip_obj,
  split_by = NULL
)
```

Arguments

`atl03_atl08_dt`	An S4 object of class `icesat2.atl03atl08_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`clip_obj`	An object of class `terra::SpatVector`, which can be loaded as an ESRI shapefile using terra::vect function.
`split_by`	Optional. clip_obj ID. If defined, the data will be clipped by each clip_obj using the clip_obj ID from the attribute table.

Value

Returns an S4 object of class `icesat2.atl03atl08_dt` containing the clipped ATL08 attributes.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path)
```

```
atl08_h5 <- ATL08_read(atl08_path)

# Joining ATL03 and ATL08 photon attributes
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# Specifying the path to the shapefile
clip_obj_filepath <- system.file("extdata", "clip_geom.shp", package = "ICESat2VegR")

# Reading shapefile as a SpatVector object
clip_obj <- terra::vect(clip_obj_filepath)

# Clipping ATL08 terrain attributes by geometry
atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipGeometry(
  atl03_atl08_dt,
  clip_obj,
  split_by = "id"
)
head(atl03_atl08_dt_clip)

close(atl03_h5)
close(atl08_h5)

}
```

ATL03_ATL08_photons_attributes_dt_gridStat

Statistics of ATL03 and ATL08 photon attributes

Description

This function computes a series of user defined descriptive statistics within each given grid cell for ATL03 and ATL08 photon attributes

Usage

```
ATL03_ATL08_photons_attributes_dt_gridStat(
  atl03_atl08_dt,
  func,
  res = 0.5,
  ph_class = c(2, 3),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = 0,
  night_flag = 1
)
```

Arguments

atl03_atl08_dt	An S4 object of class icesat2.atl03atl08_dt containing ATL03 and ATL08 attributes (output of the ATL03_ATL08_photons_attributes_dt_join() function).
func	The function to be applied for computing the defined statistics
res	Spatial resolution in decimal degrees for the output SpatRast raster layer. Default is 0.5.
ph_class	Character vector indicating photons to process based on the classification (1=ground, 2=canopy, 3=top canopy), Default is c(2,3)
beam	Character vector indicating beams to process. Default is c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
quality_ph	Indicates the quality of the associated photon. 0=nominal, 1=possible_afterpulse, 2=possible_impulse_response_effect, 3=possible_tep. Default is 0
night_flag	Flag indicating the data were acquired in night conditions: 0=day, 1=night. Default is 1

Value

Return a SpatRast raster layer(s) of selected ATL03 and ATL08 photon attribute(s)

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)
# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# # Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

# Computing the mean of ph_h attribute at 0.0002 degree grid cell
mean_ph_h <- ATL03_ATL08_photons_attributes_dt_gridStat(atl03_atl08_dt,
  func = mean(ph_h),
  res = 0.0002
)
```

```
plot(mean_ph_h)

# Define your own function
mySetOfMetrics <- function(x) {
  metrics <- list(
    min = min(x), # Min of x
    max = max(x), # Max of x
    mean = mean(x), # Mean of x
    sd = sd(x) # Sd of x
  )
  return(metrics)

# Computing a series of ph_h stats at 0.0002 degree grid cell from customized function
ph_h_metrics <- ATL03_ATL08_photons_attributes_dt_gridStat(atl03_atl08_dt,
  func = mySetOfMetrics(ph_h), res = 0.0002
)

plot(ph_h_metrics)

close(atl03_h5)
close(atl08_h5)
}

}
```

`ATL03_ATL08_photons_attributes_dt_join` *Join ATL03 and ATL08 photons attributes*

Description

This function joins ATL03 and ATL08 computed photons attributes

Usage

```
ATL03_ATL08_photons_attributes_dt_join(
  atl03_h5,
  atl08_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
)
```

Arguments

`atl03_h5`	A ICESat-2 ATL03 object (output of `ATL03_read()` function). An S4 object of class `icesat2.atl03_dt`.
`atl08_h5`	A ICESat-2 ATL08 object (output of `ATL08_read()` function). An S4 object of class `icesat2.atl08_dt`.
`beam`	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")

Details

These are the photons attributes extracted by default:

- ph_segment_id: Georeferenced segment id (20-m) associated with each photon.
- lon_ph: Longitude of each received photon. Computed from the ECEF Cartesian coordinates of the bounce point.
- lat_ph: Latitude of each received photon. Computed from the ECEF Cartesian coordinates of the bounce point.
- h_ph: Height of each received photon, relative to the WGS-84 ellipsoid including the geophysical corrections noted in section 6.0. Please note that neither the geoid, ocean tide nor the dynamic atmospheric corrections (DAC) are applied to the ellipsoidal heights.
- quality_ph: Indicates the quality of the associated photon. 0=nominal, 1=possible_afterpulse, 2=possible_impulse_response_effect, 3=possible_tep. Use this flag in conjunction with signal_conf_ph to identify those photons that are likely noise or likely signal.
- solar_elevation: Elevation of the sun above the horizon at the photon bounce point.
- dist_ph_along: Along-track distance of the photon from the beginning of the segment.
- dist_ph_across: Across-track distance of the photon from the center of the segment.
- night_flag: Flag indicating the data were acquired in night conditions: 0=day, 1=night. Night flag is set when solar elevation is below 0.0 degrees.
- classed_pc_indx: Indices of photons tracking back to ATL03 that surface finding software identified and used within the creation of the data products.
- classed_pc_flag: The L2B algorithm is run if this flag is set to 1 indicating data have sufficient waveform fidelity for L2B to run.
- ph_h: Height of photon above interpolated ground surface.
- d_flag: Flag indicating whether DRAGANN labeled the photon as noise or signal.
- delta_time: Mid-segment GPS time in seconds past an epoch. The epoch is provided in the metadata at the file level.
- orbit_number: Orbit number identifier to identify data from different orbits.
- beam: Beam identifier.
- strong_beam: Logical indicating if the beam is a strong beam.

Value

Returns an S4 object of class icesat2.atl03atl08_dt containing the ATL08 computed photons attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# # Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

close(atl03_h5)
close(atl08_h5)

}
```

ATL03_ATL08_photons_attributes_dt_LAS

Export Merged ICESat-2 ATL03/ATL08 Photon Data to LAS Files

Description

Converts the merged ATL03/ATL08 photon dataset generated by `ATL03_ATL08_photons_attributes_dt_join()` into one or more LAS files. The output preserves ATL03 photon geolocation and elevation information together with ATL08 photon classifications and normalized heights, enabling the resulting point cloud to be visualized and analyzed in standard LiDAR software.

Usage

```
ATL03_ATL08_photons_attributes_dt_LAS(
  atl03_atl08_dt,
  output,
  normalized = TRUE
)
```

Arguments

atl03_atl08_dt	An S4 object of class ICESat2VegR::icesat2.atl08_dt-class or a data.table containing ATL03 and ATL08 photon data, usually the output of ATL03_ATL08_photons_attributes_dt
output	Character. Output LAS file path. The function creates one LAS file per UTM zone in the WGS84 datum.
normalized	Logical. If TRUE, writes normalized ATL08 photon height ph_h; if FALSE, writes raw ATL03 photon height h_ph. Default is TRUE.

Details

This function converts the output of ATL03_ATL08_photons_attributes_dt_join() to LAS format using the package internal dt_to_las() and writeLAS() functions.

The input table must contain photon longitude, photon latitude, ATL03 photon height, ATL08 normalized photon height, and ATL08 photon classification.

If normalized = TRUE, the ph_h column is used as the LAS Z coordinate. If normalized = FALSE, the raw ATL03 h_ph column is used as the LAS Z coordinate.

Photon classes are written to the LAS Classification field using classed_pc_flag + 1.

Because LAS files require projected coordinates, the longitude and latitude values are split by UTM zone and projected to the corresponding UTM CRS before writing. One LAS file is created per UTM zone. The internal UTM-zone assignment follows helper routines based on the latitude/longitude to UTM conversion algorithms developed by Chuck Gantz.

Value

Invisibly returns a character vector with the written LAS file paths.

References

Gantz, C. Latitude/Longitude to UTM Conversion Algorithms. https://www.gpsy.com/gpsinfo/geotoutm/

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
outdir <- tempdir()

atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(
  atl03_h5,
  atl08_h5
)

ATL03_ATL08_photons_attributes_dt_LAS(
```

```
  atl03_atl08_dt,
  output = file.path(outdir, "output.las"),
  normalized = TRUE
)

close(atl03_h5)
close(atl08_h5)

}
```

`ATL03_ATL08_photons_attributes_dt_polyStat` *Statistics of ATL03 and ATL08 joined photons attributes within a given area*

Description

Computes a series of statistics ATL03 and ATL08 joined photons attributes within area defined by a polygon

Usage

```
ATL03_ATL08_photons_attributes_dt_polyStat(
  atl03_atl08_dt,
  func,
  poly_id = NULL
)
```

Arguments

`atl03_atl08_dt`	An S4 object of class `icesat2.atl08_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`func`	The function to be applied for computing the defined statistics
`poly_id`	Polygon id. If defined, statistics will be computed for each polygon

Value

Returns an S4 object of class `icesat2.atl08_dt` Containing Statistics of ATL08 classified canopy photons

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
```

```
)

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# Specifying the path to shapefile
polygon_filepath <- system.file("extdata", "clip_geom.shp", package = "ICESat2VegR")

# Reading shapefile as sf object
polygon <- terra::vect(polygon_filepath)

# Clipping ATL08 terrain attributes by Geometry
atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipGeometry(atl03_atl08_dt,
  polygon, split_by = "id")

# Computing the maximum ph_h by polygon id
max_ph_h <- ATL03_ATL08_photons_attributes_dt_polyStat(atl03_atl08_dt_clip,
  func = max(ph_h), poly_id = "poly_id")
head(max_ph_h)

# Define your own function
mySetOfMetrics <- function(x) {
  metrics <- list(
    min = min(x), # Min of x
    max = max(x), # Max of x
    mean = mean(x), # Mean of x
    sd = sd(x) # Sd of x
  )
  return(metrics)

# Computing a series of ph_h statistics from customized function
ph_h_metrics <- ATL03_ATL08_photons_attributes_dt_polyStat(
  atl03_atl08_dt_clip,
  func = mySetOfMetrics(ph_h),
  poly_id = "poly_id"
)

head(ph_h_metrics)

close(atl03_h5)
```

```
close(atl08_h5)
}

}
```

`ATL03_ATL08_photons_seg_dt_fitground` *Fit and estimate ground elevation for photons or arbitrary distances from the track beginning*

Description

Function to estimate ground elevation using smoothing and interpolation functions. Ground photons (classed_pc_flag == 1) are first aggregated within a smoothing window, then an interpolation function is applied to estimate the ground elevation at each photon location or at arbitrary distances along the track.

Usage

```
ATL03_ATL08_photons_seg_dt_fitground(
  atl03_atl08_seg_dt,
  smoothing_window = NA,
  smoothing_func = median,
  interpolation_func = NA,
  xout_parameter_name = "xout",
  ...
)
```

Arguments

`atl03_atl08_seg_dt`
: An S4 object of class `icesat2.atl03_atl08_seg_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_segment_create()` function).

`smoothing_window`
: numeric. The smoothing window size in meters for aggregating ground photons before interpolation. Default is `NA`, which uses the ATBD adaptive window size. See Details for more information.

`smoothing_func`
: function. The aggregation function applied to ground photon elevations within each smoothing window. Default is `median`.

`interpolation_func`
: function. The interpolation function used to estimate ground elevation from the smoothed ground photons. Default is `NA`, which falls back to `stats::approxfun`. See Details for more information.

`xout_parameter_name`
: character. The name of the parameter used by `interpolation_func` to receive the prediction vector. When set to `"xout"` (default), the function predicts ground elevation at every photon's `dist_ph_along` position. When set to `NA`,

the raw output of `interpolation_func` is returned directly without predicting at photon locations (useful when `interpolation_func` returns a function, e.g. `approxfun`). See Details for more information.

`...` Optional additional parameters passed to `interpolation_func`. See Details for more information.

Details

The function for calculating the ground will first pass a smoothing window with `smoothing_window` size, applying the `smoothing_func` to aggregate the ground photons (`classed_pc_flag == 1`).

Then it will use an interpolation function between those aggregated photons to estimate a smooth ground surface.

The `smoothing_func` signature will depend on the function used. It is assumed that the first two arguments are vectors of `x` (independent variable) and `y` (the values to be aggregated). The remaining arguments are passed through `....`

The interpolation functions need a third parameter which is the `x` vector to predict at. Functions from the `stats` base package such as `stats::approx()` and `stats::spline()` name this argument `xout`, so you can use:

```
ATL03_ATL08_photons_seg_dt_fitground(
  atl03_atl08_seg_dt,
  interpolation_func = approx,
  xout_parameter_name = "xout"
)
```

to predict at every photon location along the track. Other functions may name the prediction parameter differently — for example, `signal::pchip()` uses `xi` instead of `xout`. The `pchip` algorithm (as implemented in the **signal** package) is the algorithm used by the ATL08 ATBD.

The `smoothing_window` can be left `NA`, which will use the ATBD adaptive algorithm for calculating the window size:

$Sspan = \lceil 5 + 46 \times (1 - e^{-a \times length}) \rceil$, where *length* is the number of photons within the segment.

$$a \approx 21 \times 10^{-6}$$

$$window\_size = \frac{2}{3} \times Sspan$$

This is not the same algorithm as used in ATL08 but is an adapted version that uses the ATL08 pre-classification.

Value

When `xout_parameter_name` is not `NA`, returns a named list with two numeric vectors:

x Distance along track (`dist_ph_along`) for each photon, in meters.

y Interpolated ground elevation (h_ph) at each photon location, in meters. Values are NA for photons outside the smoothed ground range (controlled by the rule argument of the interpolation function).

When xout_parameter_name = NA, returns the raw output of interpolation_func directly — for example, a callable function when interpolation_func = approxfun, which can then be evaluated at arbitrary distances along the track.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

# Converting to seg_dt class (required input for fitground)
atl03_atl08_seg_dt <- ATL03_ATL08_segment_create(
  atl03_atl08_dt,
  segment_length = 30
)

# Example 1: ATBD adaptive smoothing window + linear interpolation
# at each photon location along the track.
# Returns a list with $x (dist_ph_along) and $y (ground elevation in meters).
# NAs appear at photon locations outside the smoothed ground range.
ground_approx <- ATL03_ATL08_photons_seg_dt_fitground(
  atl03_atl08_seg_dt,
  interpolation_func = approx,
  xout_parameter_name = "xout"
)
head(ground_approx$x) # photon distances along track (meters)
head(ground_approx$y) # interpolated ground elevations (meters)

# Example 2: Return an interpolation function (approxfun) for
# querying ground elevation at arbitrary distances along the track.
# xout_parameter_name = NA skips prediction at photon locations
# and returns the interpolation function directly.
ground_fun <- ATL03_ATL08_photons_seg_dt_fitground(
  atl03_atl08_seg_dt,
  interpolation_func = approxfun,
  xout_parameter_name = NA
)
# Query ground elevation at custom distances along track (meters)
ground_fun(c(100, 200, 300))

# Example 3: Fixed 5 m smoothing window using mean instead of median
```

```
ground_mean <- ATL03_ATL08_photons_seg_dt_fitground(
  atl03_atl08_seg_dt,
  smoothing_window = 5,
  smoothing_func = mean,
  interpolation_func = approx,
  xout_parameter_name = "xout"
)
head(ground_mean$y)

close(atl03_h5)
close(atl08_h5)
}
```

`ATL03_ATL08_photons_seg_dt_height_normalize`
Normalize photon heights relative to estimated ground elevation

---

Description

Normalizes ATL03 and ATL08 photon heights (ph_h) by subtracting the estimated ground elevation at each photon location. Ground elevation is estimated internally using `ATL03_ATL08_photons_seg_dt_fitground`. The function updates the ph_h column in place and returns the modified `atl03_atl08_seg_dt` object.

Usage

```
ATL03_ATL08_photons_seg_dt_height_normalize(
  atl03_atl08_seg_dt,
  smoothing_window = NA,
  smoothing_func = median,
  interpolation_func = NA,
  xout_parameter_name = "xout",
  ...
)
```

Arguments

`atl03_atl08_seg_dt`
: An S4 object of class `icesat2.atl03_atl08_seg_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_segment_create()` function).

`smoothing_window`
: numeric. The smoothing window size in meters for aggregating ground photons before interpolation. Default is `NA`, which uses the ATBD adaptive window size. See Details for more information.

`smoothing_func`
: function. The aggregation function applied to ground photon elevations within each smoothing window. Default is `median`.

`interpolation_func`
: function. The interpolation function used to estimate ground elevation from the smoothed ground photons. Default is `NA`, which falls back to `stats::approx`. See Details for more information.

`xout_parameter_name`
: character. The name of the parameter used by `interpolation_func` to receive the prediction vector. Default is `"xout"`. See Details for more information.

`...`
: Additional parameters passed forward to `ATL03_ATL08_photons_seg_dt_fitground`.

Details

This function is a wrapper around `ATL03_ATL08_photons_seg_dt_fitground`. It first estimates the ground elevation at every photon location using the smoothing and interpolation approach described in that function, then subtracts the estimated ground elevation from the raw photon height (`h_ph`) to produce height above ground level.

For full details on the smoothing window, smoothing function, and interpolation function behaviour, see `ATL03_ATL08_photons_seg_dt_fitground`.

Value

Returns the input `atl03_atl08_seg_dt` object with the `ph_h` column updated **in place** to contain height above ground level (AGL) in meters, computed as `h_ph - ground_elevation`. Values are `NA` for photons outside the interpolated ground range. Note: as this function uses `data.table` assignment by reference, the original input object is also modified.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

# Converting to seg_dt class (required input)
atl03_atl08_seg_dt <- ATL03_ATL08_segment_create(
  atl03_atl08_dt,
  segment_length = 30
)

# Example 1: Normalize photon heights using default ATBD smoothing window
atl03_atl08_seg_dt_norm <- ATL03_ATL08_photons_seg_dt_height_normalize(
  atl03_atl08_seg_dt,
  interpolation_func = approx,
  xout_parameter_name = "xout"
)
```

```
head(atl03_atl08_seg_dt_norm)

# Example 2: Fixed 5 m smoothing window using mean aggregation
# Recreate seg_dt since ph_h was modified in place by the previous call
atl03_atl08_seg_dt <- ATL03_ATL08_segment_create(
  atl03_atl08_dt,
  segment_length = 30
)
atl03_atl08_seg_dt_norm2 <- ATL03_ATL08_photons_seg_dt_height_normalize(
  atl03_atl08_seg_dt,
  smoothing_window = 5,
  smoothing_func = mean,
  interpolation_func = approx,
  xout_parameter_name = "xout"
)
head(atl03_atl08_seg_dt_norm2)

close(atl03_h5)
close(atl08_h5)
}
```

`ATL03_ATL08_segment_create` *Compute segments id for a given segment length*

Description

This function reads the ICESat-2 Land and Vegetation Along-Track Products (ATL08) as h5 file.

Usage

```
ATL03_ATL08_segment_create(
  atl03_atl08_dt,
  segment_length,
  centroid = "mean",
  output = NA,
  overwrite = FALSE
)
```

Arguments

`atl03_atl08_dt`	`icesat2.atl03atl08_dt`. The output of the `ATL03_ATL08_photons_attributes_dt_join()`.
`segment_length`	`numeric`. The desired segment length to split the photons.
`centroid`	character. Method used to calculate the segment centroid, either "mean" or "midpoint", see details. Default 'mean'.
`output`	Character vector. The output vector file. The GDAL vector format will be inferred by the file extension using `terra::writeVector()`
`overwrite`	logical input to control if the output vector file should be overwritten. Default FALSE.

Details

The centroid will be computed using either the photons centroid or the approximate segment centroid.

- "mean": calculated using the average coordinates from all photons within the segment. This approach will better represent the mean statistics location.
- "mid-point": the minimum and maximum coordinates will be averaged to calculate a midpoint within the segment. This will give a better representation of the segment true mid-point

Value

Returns an S4 object of class `icesat2.atl03atl08_dt` containing ICESat-2 ATL08 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ICESat-2 ATL03 and ATL08 data
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ICESat-2 ATL08 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

atl03_atl08_dt_seg <- ATL03_ATL08_segment_create(atl03_atl08_dt,
  segment_length = 30,
  centroid = "mean",
  output = NA,
  overwrite = FALSE
)

head(atl03_atl08_dt_seg)

close(atl03_h5)
close(atl08_h5)
}
```

`ATL03_ATL08_seg_attributes_dt_clipBox` *Clip joined ATL03/ATL08 segment attributes by bounding extent*

Description

Clips an `icesat2.atl08_dt` object to a given spatial extent. Only rows whose `longitude` and `latitude` fall inside the specified bounding extent are retained. This is the bounding-box counterpart to geometry-based clipping via `ATL03_ATL08_seg_attributes_dt_clipGeometry`.

Usage

```
ATL03_ATL08_seg_attributes_dt_clipBox(atl03_atl08_dt, clip_obj)
```

Arguments

`atl03_atl08_dt`	An object of class `icesat2.atl08_dt`, typically produced by `ATL08_seg_attributes_dt()`.
`clip_obj`	Bounding extent used for clipping. Supported inputs: • A numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees. • A `SpatExtent` object.

Details

The function:

1. Validates that the input has `longitude` and `latitude` columns.
2. Converts `clip_obj` to numeric bounds `xmin`, `ymin`, `xmax`, `ymax`.
3. Drops rows with missing coordinates.
4. Returns a subset of `atl03_atl08_dt` where `longitude` and `latitude` fall inside the bounding box.

Value

An object of the same class as `atl03_atl08_dt`, containing only segments whose `longitude` and `latitude` fall inside the bounding extent.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
```

```
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08 segment attributes
atl08_seg_dt <- ATL08_seg_attributes_dt(atl08_h5)
head(atl08_seg_dt)

# Example 1: Clip using a numeric bounding box (xmin, ymin, xmax, ymax)
bbox <- c(-106.571, 41.532, -106.570, 41.537)
atl08_seg_clip <- ATL03_ATL08_seg_attributes_dt_clipBox(
  atl08_seg_dt,
  clip_obj = bbox
)
head(atl08_seg_clip)
nrow(atl08_seg_clip)

# Example 2: Clip using a terra SpatExtent object
library(terra)
ext_obj <- terra::ext(bbox)
atl08_seg_clip2 <- ATL03_ATL08_seg_attributes_dt_clipBox(
  atl08_seg_dt,
  clip_obj = ext_obj
)
head(atl08_seg_clip2)
nrow(atl08_seg_clip2)

close(atl08_h5)
}
```

`ATL03_ATL08_seg_attributes_dt_clipGeometry` *Clip joined ATL03/ATL08 segment attributes by geometry*

Description

Clips an `icesat2.atl08_dt` or `icesat2.atl03_atl08_seg_dt` object to the features inside a polygon. Accepts `terra::SpatVector` or `sf/sfc` polygons. Performs a fast bounding box pre-clip, then a precise intersection in the polygon CRS.

Usage

```
ATL03_ATL08_seg_attributes_dt_clipGeometry(
  atl03_atl08_dt,
  clip_obj,
  split_by = NULL
)
```

Arguments

`atl03_atl08_dt`	An object of class `icesat2.atl08_dt` or `icesat2.atl03_atl08_seg_dt`, typically produced by `ATL08_seg_attributes_dt()`.

clip_obj	A polygon as terra::SpatVector or sf/sfc.
split_by	character. Optional name of a polygon attribute to copy into the output as column poly_id. Default is NULL.

Details

The function performs clipping in two steps for efficiency:

1. A fast bounding box pre-clip using ATL03_ATL08_seg_attributes_dt_clipBox to reduce the number of points before the precise intersection.
2. A precise point-in-polygon intersection using terra::intersect(), reprojecting the points to the polygon CRS if needed.

Value

An object of the same class as atl03_atl08_dt containing only segments that fall inside the polygon. If split_by is provided, an additional poly_id column is added identifying which polygon feature each segment belongs to.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08 segment attributes
atl08_seg_dt <- ATL08_seg_attributes_dt(atl08_h5)
head(atl08_seg_dt)

# Reading polygon from shapefile
polygon_filepath <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)
polygon <- terra::vect(polygon_filepath)

# Example 1: Clip by polygon geometry
atl08_seg_clip_geom <- ATL03_ATL08_seg_attributes_dt_clipGeometry(
  atl08_seg_dt,
  clip_obj = polygon
)
head(atl08_seg_clip_geom)
nrow(atl08_seg_clip_geom)

# Example 2: Clip by polygon geometry and add polygon id column
```

```
atl08_seg_clip_geom2 <- ATL03_ATL08_seg_attributes_dt_clipGeometry(
  atl08_seg_dt,
  clip_obj = polygon,
  split_by = "id"
)
head(atl08_seg_clip_geom2)
nrow(atl08_seg_clip_geom2)

close(atl08_h5)

}
```

`ATL03_ATL08_seg_cover_dt_compute` *Compute canopy cover from ATL03/ATL08 classified photons*

Description

Computes canopy cover metrics from ATL03 and ATL08 classified photons within each segment. Cover is calculated using the ratio of ground to vegetation photons, optionally weighted by a reflectance ratio. The function adds three new columns (`cover`, `n_g`, `n_v`) to the input object and returns it.

Usage

```
ATL03_ATL08_seg_cover_dt_compute(atl03_atl08_dt, reflectance_ratio = 1)
```

Arguments

`atl03_atl08_dt`	An object of class `icesat2.atl03_atl08_seg_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_segment_create()` function).
`reflectance_ratio`	numeric. The reflectance ratio ρ_v/ρ_g, where ρ_v is the vegetation reflectance and ρ_g is the ground reflectance. Default is 1 (same reflectance for ground and vegetation).

Details

Cover is calculated with the formula:

$$cover = \frac{1}{1 + \frac{\rho_v \cdot N_g}{\rho_g \cdot N_v}}$$

where N_g is the number of ground photons (`classed_pc_flag == 1`) and N_v is the number of vegetation photons (`classed_pc_flag > 1`) within each segment.

Value

Returns the input object with three additional columns:

cover Numeric. Canopy cover fraction per segment (0-1).

n_g Integer. Number of ground photons per segment (`classed_pc_flag == 1`).

n_v Integer. Number of vegetation photons per segment (`classed_pc_flag > 1`).

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

# Converting to seg_dt class (required input)
atl03_atl08_seg_dt <- ATL03_ATL08_segment_create(
  atl03_atl08_dt,
  segment_length = 30
)

# Computing canopy cover metrics per segment
cover <- ATL03_ATL08_seg_cover_dt_compute(atl03_atl08_seg_dt)
head(cover[, c("segment_id", "cover", "n_g", "n_v")])

# Computing cover with a custom reflectance ratio (rho_v/rho_g = 1.5)
cover2 <- ATL03_ATL08_seg_cover_dt_compute(
  atl03_atl08_seg_dt,
  reflectance_ratio = 1.5
)
head(cover2[, c("segment_id", "cover", "n_g", "n_v")])

close(atl03_h5)
close(atl08_h5)

}
```

ATL03_h5_clipBox *Clip ICESat-2 ATL03 HDF5 Data Using a Bounding Extent*

Description

Clips an ICESat-2 ATL03 HDF5 file to a specified spatial extent. Only geolocated photon and segment datasets within the ATL03 beam groups are clipped; all metadata, orbit information, and ancillary groups are preserved unchanged. This function provides bounding-box-based clipping, complementing geometry-based clipping workflows.

Usage

```
ATL03_h5_clipBox(
  atl03,
  output,
  clip_obj,
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

atl03	An icesat2.atl03_h5 object created by ATL03_read(), representing the ATL03 HDF5 file to be clipped.
output	Character. Path to the output HDF5 file that will store the clipped ATL03 dataset.
clip_obj	Bounding extent used for clipping. Supported inputs: • A numeric vector of length 4: c(ul_lat, ul_lon, lr_lat, lr_lon), following the ICESat-2 CMS bounding-box convention (upper-left latitude/longitude and lower-right latitude/longitude). • A terra::SpatExtent object.
beam	Character vector specifying which ATL03 beams to include. Defaults to: c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r").
additional_groups	Character vector of additional non-beam HDF5 groups to copy unchanged into the output file. Defaults to: c("orbit_info").

Details

The function performs spatial subsetting at the photon and segment level. Internally, it:

1. Copies file- and group-level attributes.
2. Extracts beam-level datasets and evaluates segment-level and photon-level masks using the supplied bounding box.
3. Applies these masks to all datasets whose dimensions correspond to segments or photons.

4. Recomputes dependent indexing datasets (e.g., ph_index_beg) to maintain ATL03 structural integrity.
5. Writes the clipped data into a new, valid ATL03 HDF5 file.

This function preserves the full ATL03 metadata, ancillary information, and file structure while trimming the spatial domain to the user-specified bounding extent.

Value

An S4 object of class icesat2.atl03_h5 representing the clipped ATL03 HDF5 file saved at the output location.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL03 data (HDF5)
atl03_h5 <- ATL03_read(atl03_path)

# Bounding rectangle coordinates (ul_lat, ul_lon, lr_lat, lr_lon)
xmin <- -106.5723
xmax <- -106.5693
ymin <- 41.533
ymax <- 41.537

bbox <- c(ymax, xmin, ymin, xmax)

# Clip ATL03 using the bounding extent
output <- tempfile(fileext = ".h5")
atl03_clip <- ATL03_h5_clipBox(
  atl03_h5,
  output = output,
  clip_obj = bbox
)

close(atl03_h5)

}
```

ATL03_h5_clipGeometry *Clip ICESat-2 ATL03 HDF5 Data Using Geometry-Based Boundaries*

Description

Clips an ICESat-2 ATL03 HDF5 file using one or more geometry-based clipping objects provided as a `terra::SpatVector`. Each geometry in `vect` defines an individual clipping region. The function iteratively clips the ATL03 data for each region and writes a separate output HDF5 file for every geometry. The name of each output file is automatically appended with the value of the attribute specified in `split_by`.

Only datasets within the ATL03 beam groups are spatially clipped; metadata and ancillary non-beam groups remain unchanged in the resulting files.

Usage

```
ATL03_h5_clipGeometry(
  atl03,
  output,
  clip_obj,
  split_by = NULL,
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

`atl03`	An `icesat2.atl03_h5` object obtained using `ATL03_read()`, representing the ATL03 HDF5 file to be clipped.
`output`	Character. Path to the output filename. The final written files will append the unique value of `split_by` for each clipping geometry, for example: `output_id1.h5`, `output_id2.h5`, etc.
`clip_obj`	A `terra::SpatVector` object containing the geometric clip boundaries. Each feature (row) in the SpatVector defines an independent clipping region.
`split_by`	Character. The SpatVector attribute whose values are used to identify and name each clipping geometry. Defaults to `id`.
`beam`	Character vector specifying which ATL03 beams to include. The default includes all six beams: `c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")`.
`additional_groups`	Character vector of non-beam HDF5 groups that should be copied unchanged into each clipped file. Defaults to: `c("orbit_info")`.

Value

Returns a list of clipped S4 objects of class `icesat2.atl03_h5`, one for each clipping geometry contained in `vect`.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL03 file path
```

```
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL03 file
atl03_h5 <- ATL03_read(atl03_path)

# Output base filename
output <- tempfile(fileext = ".h5")

# Load clipping geometries
vect_path <- system.file("extdata", "clip_geom.shp",
  package = "ICESat2VegR"
)
clip_obj <- terra::vect(vect_path)

# Clip ATL03 data using polygon geometries
atl03_clipped_list <- ATL03_h5_clipGeometry(
  atl03_h5,
  output,
  clip_obj,
  split_by = "id"
)

close(atl03_h5)

}
```

`ATL03_photons_attributes_dt` *ATL03 photon attributes*

Description

Extract photon-level attributes from ICESat-2 ATL03 data.

Usage

```
ATL03_photons_attributes_dt(
  atl03_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  attributes = c("quality_ph", "dist_ph_along")
)
```

Arguments

`atl03_h5`	An ICESat-2 ATL03 object (output of `ATL03_read()`), i.e. an S4 object of class `icesat2.atl03_h5`.

beam	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r").
attributes	Character vector of additional photon-level attributes to extract. See details below for available options. Default is c("quality_ph", "dist_ph_along").

Details

The returned data.table always includes the following columns:

- beam: Beam ID (e.g., "gt2l").
- strong_beam: Logical indicating whether the beam is classified as strong for that orbit.
- lon_ph: Longitude of each received photon, computed from ECEF Cartesian coordinates of the bounce point (degrees_east).
- lat_ph: Latitude of each received photon, computed from ECEF Cartesian coordinates of the bounce point (degrees_north).
- h_ph: Height of each received photon, relative to the WGS-84 ellipsoid, including the geophysical corrections described in the ATL03 ATBD. (Geoid, ocean tide, and dynamic atmosphere corrections are *not* applied.)
- solar_elevation: Solar elevation interpolated from segment-level geolocation/solar_elevation to photon level.

Additional photon-level attributes can be requested using the attributes argument. These must correspond to names defined in ATL03.photon.map. Available optional attributes currently include:

- delta_time: Photon transmit time in seconds since 2018-01-01 (ATLAS SDP epoch).
- dist_ph_across: Across-track distance of the photon projected to the reference ellipsoid (meters).
- pce_mframe_cnt: Major frame counter (part of photon ID).
- ph_id_channel: Channel number assigned to the photon event.
- ph_id_count: Photon event counter (part of photon ID).
- ph_id_pulse: Laser pulse counter (part of photon ID).
- quality_ph: Photon quality flag indicating nominal, saturation, or noise conditions.
- signal_class_ph: Experimental photon classification flag based on photon weights.
- signal_conf_ph: Signal confidence level per surface type (5 x N array in original product).
- weight_ph: Relative photon weight (0-65535), proportional to photon density.

If a requested attribute is not present in a given beam, the corresponding column is filled with NA.

Value

An S4 object of class data.table::data.table (with class icesat2.atl03_dt prepended) containing photon-level ATL03 attributes.

See Also

https://nsidc.org/sites/default/files/documents/technical-reference/icesat2_atl03_data_dict_v007.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file(
  "extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path)

atl03_photons_dt <- ATL03_photons_attributes_dt(
 atl03_h5,
 beam = c("gt1r"),
 attributes = c("quality_ph", "dist_ph_along", "ph_id_count")
)
head(atl03_photons_dt)

close(atl03_h5)


}
```

ATL03_photons_attributes_dt_clipBox

Clip ATL03 photons by bounding extent

Description

Clips ATL03 photon attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
ATL03_photons_attributes_dt_clipBox(atl03_photons_dt, clip_obj)
```

Arguments

`atl03_photons_dt`	An `icesat2.atl03_dt` object (output of `ATL03_photons_attributes_dt()`).
`clip_obj`	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees. • `SpatExtent` (package **terra**): the extent is used directly.

Details

When `clip_obj` is a `SpatExtent`, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns a subset of the original `icesat2.atl03_dt` object containing only photons within the bounding box.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file("extdata", "atl03_clip.h5",
                          package = "ICESat2VegR")

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5 = atl03_h5)

# 1) Using a numeric bbox: xmin ymin xmax ymax
bbox <- c(-106.57, 41.53, -106.5698, 41.54)
atl03_clip_bbox <- ATL03_photons_attributes_dt_clipBox(
  atl03_photons_dt = atl03_photons_dt,
  clip_obj = bbox
)

# 2) Using a SpatExtent
# library(terra)
# ext <- terra::ext(-106.57, -106.5698, 41.53, 41.54)
# atl03_clip_ext <- ATL03_photons_attributes_dt_clipBox(
#   atl03_photons_dt = atl03_photons_dt,
#   clip_obj = ext
# )

close(atl03_h5)

}
```

ATL03_photons_attributes_dt_clipGeometry

Clip ATL03 photons by Coordinates

Description

This function clips ATL03 photon attributes within given bounding coordinates.

Usage

```
ATL03_photons_attributes_dt_clipGeometry(
  atl03_photons_dt,
  clip_obj,
  split_by = "id"
)
```

Arguments

atl03_photons_dt	An ATL03 photon data table. An S4 object of class icesat2.atl03_dt.
clip_obj	Spatial clip_obj. An object of class terra::SpatVector, which can be loaded as an ESRI shapefile using the terra::vect function in the *sf* package.
split_by	clip_obj id. If defined, GEDI data will be clipped by each clip_obj using the clip_obj id from the attribute table defined by the user.

Value

Returns an S4 object of class icesat2.atl03_dt containing the ATL03 photon attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Extracting ATL03 photon attributes
atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5 = atl03_h5)

# Specifying the path to shapefile
clip_obj_filepath <-
  system.file(
    "extdata",
    "clip_geom.shp",
    package = "ICESat2VegR"
  )

# Reading shapefile as sf object
clip_obj <- terra::vect(clip_obj_filepath)
```

```
# Clipping ATL03 photon attributes by Geometry
atl03_photons_dt_clip <-
  ATL03_photons_attributes_dt_clipGeometry(atl03_photons_dt, clip_obj, split_by = "id")

head(atl03_photons_dt_clip)

close(atl03_h5)
}
```

`ATL03_photons_attributes_dt_LAS` *Export ICESat-2 ATL03 Photon Data to LAS Files*

Description

Converts ATL03 photon-level data extracted with `ATL03_photons_attributes_dt()` into one or more LAS files.

Usage

```
ATL03_photons_attributes_dt_LAS(atl03_dt, output)
```

Arguments

`atl03_dt`	An S4 object of class `ICESat2VegR::icesat2.atl03_dt-class`, usually the output of `ATL03_photons_attributes_dt()`.
`output`	Character. Output LAS file path. The function creates one LAS file per UTM zone in the WGS84 datum.

Details

LAS files require projected coordinates. This function uses internal helper functions to identify the appropriate UTM zone for each photon, reproject the original ICESat-2 geographic coordinates, and write one LAS file per UTM zone using the package internal LAS writer. The internal UTM-zone assignment follows helper routines based on the latitude/longitude to UTM conversion algorithms developed by Chuck Gantz.

Value

Invisibly returns a character vector with the written LAS file paths.

References

Gantz, C. Latitude/Longitude to UTM Conversion Algorithms. https://www.gpsy.com/gpsinfo/geotoutm/

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path = atl03_path)

atl03_dt <- ATL03_photons_attributes_dt(atl03_h5, beam = "gt1r")

outdir <- tempdir()

ATL03_photons_attributes_dt_LAS(
  atl03_dt,
  file.path(outdir, "atl03_photons.las")
)

close(atl03_h5)

}
```

`ATL03_read` *Read ICESat-2 ATL03 data*

Description

This function reads the ICESat-2 Global Geolocated Photons Product (ATL03) from an HDF5 (.h5) file.

Usage

```
ATL03_read(atl03_path)
```

Arguments

`atl03_path`	Either a file path pointing to ICESat-2 ATL03 HDF5 data, or a granule returned by `ATLAS_dataFinder()` when `cloud_computing = TRUE`.

Value

An S4 object of class `icesat2.atl03_dt` containing ICESat-2 ATL03 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r007.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specify the path to an ATL03 file
atl03_path <- system.file(
  "extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Read ICESat-2 ATL03 data (HDF5 file)
ATL03 <- ATL03_read(atl03_path = atl03_path)
close(ATL03)

}
```

`ATL03_read,character-method`

Read ICESat-2 ATL03 data from a local HDF5 file

Description

Method for reading ICESat-2 ATL03 data from a local HDF5 (.h5) file.

Usage

```
## S4 method for signature 'character'
ATL03_read(atl03_path)
```

Arguments

`atl03_path`	A character string specifying the path to the ICESat-2 ATL03 HDF5 file.

Value

An S4 object of class `icesat2.atl03_h5` containing ICESat-2 ATL03 data.

`ATL03_read,icesat2.granule_cloud-method`

Read ICESat-2 ATL03 data from a single cloud granule

Description

Method for reading ICESat-2 ATL03 data from a cloud-based granule.

Usage

```
## S4 method for signature 'icesat2.granule_cloud'
ATL03_read(atl03_path)
```

Arguments

atl03_path	An object of class icesat2.granule_cloud pointing to ICESat-2 ATL03 data stored in the cloud.

Value

An S4 object of class icesat2.atl03_h5 containing ICESat-2 ATL03 data.

ATL03_seg_metadata_dt *ATL03 geolocation segment metadata*

Description

Extract geolocation segment-level metadata from ICESat-2 ATL03 data. Each row in the output corresponds to a single **20m geolocation segment** along track, not to individual photons.

In addition to variables from the ATL03 geolocation group, this function derives a segment-level reference photon height (h_ph) using the reference photon index and the photon-height array in the heights group.

Usage

```
ATL03_seg_metadata_dt(
  atl03_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  attributes = c("h_ph", "altitude_sc", "bounce_time_offset", "delta_time",
   "full_sat_fract", "near_sat_fract", "neutat_delay_derivative", "neutat_delay_total",
    "neutat_ht", "ph_index_beg", "pitch", "podppd_flag", "range_bias_corr",
   "ref_azimuth", "ref_elev", "reference_photon_index", "roll", "segment_dist_x",
   "segment_id", "segment_length", "segment_ph_cnt", "sigma_across", "sigma_along",
   "sigma_h", "sigma_lat", "sigma_lon", "solar_azimuth", "solar_elevation", "surf_type",
    "tx_pulse_energy", "tx_pulse_skew_est",
     "tx_pulse_width_lower",
    "tx_pulse_width_upper", "yaw")
)
```

Arguments

atl03_h5	An ICESat-2 ATL03 object (output of ATL03_read()), i.e. an S4 object of class icesat2.atl03_h5.
beam	Character vector indicating beams to process (e.g. gt1l, gt1r, gt2l, gt2r, gt3l, gt3r). Only beams present in atl03_h5$beams are used.

attributes — Character vector naming the segment-level variables to extract. By default, a broad set of geolocation and quality variables is used, including a derived reference-photon height (h_ph).

Details

The following variables may be requested via 'attributes":

- h_ph: Height of the **reference photon** above the WGS84 ellipsoid for each geolocation segment. This is derived from geolocation/ph_index_beg, geolocation/reference_photon_index, and heights/h_ph.
- altitude_sc: Height of the spacecraft above the WGS84 ellipsoid.
- beta_angle: Acute angle between Sun vector and orbit plane
- bounce_time_offset: Difference between the transmit time and the ground-bounce time of the reference photon.
- delta_time: Transmit time of the reference photon, measured in seconds from 'atlas_sdp_gps_epoch".
- full_sat_fract: Fraction of pulses within the segment that are fully saturated.
- near_sat_fract: Fraction of pulses within the segment that are nearly saturated.
- neutat_delay_derivative: Change in neutral atmospheric delay per unit height change.
- neutat_delay_total: Total neutral atmosphere delay correction (wet + dry).
- neutat_ht: Reference height of the neutral atmosphere range correction.
- ph_index_beg: 1-based index of the first photon in this segment within the photon-rate data.
- pitch: Spacecraft pitch (degrees), 3-2-1 Euler sequence.
- podppd_flag: Composite flag describing the quality of input geolocation products for the segment.
- range_bias_corr: Estimated range bias from geolocation analysis.
- ref_azimuth: Azimuth (radians) of the unit pointing vector for the reference photon in the local ENU frame.
- ref_elev: Elevation (radians) of the unit pointing vector for the reference photon in the local ENU frame.
- reference_photon_index: Index of the reference photon within the photon set for a segment.
- reference_photon_lat: Latitude of the reference photon.
- reference_photon_lon: Longitude of the reference photon.
- roll: Spacecraft roll (degrees), 3-2-1 Euler sequence.
- segment_dist_x: Along-track distance from the equator crossing to the start of the 20 m geolocation segment.
- segment_id: 7-digit along-track geolocation segment identifier.
- segment_length: Along-track length of the geolocation segment (typically 20 m).
- segment_ph_cnt: Number of photons in the segment.
- sigma_across: Estimated Cartesian across-track uncertainty (1-sigma) for the reference photon.

- sigma_along: Estimated Cartesian along-track uncertainty (1-sigma) for the reference photon.
- sigma_h: Estimated height uncertainty (1-sigma) for the reference photon bounce point.
- sigma_lat: Estimated geodetic latitude uncertainty (1-sigma) for the reference photon.
- sigma_lon: Estimated geodetic longitude uncertainty (1-sigma) for the reference photon.
- solar_azimuth: Azimuth (degrees east) of the sun position vector from the reference photon bounce point in the local ENU frame.
- solar_elevation: Elevation (degrees) of the sun position vector from the reference photon bounce point in the local ENU frame.
- surf_type: Flags describing which surface types the segment is associated with (land, ocean, sea ice, land ice, inland water).
- tx_pulse_energy: Average transmit pulse energy per beam.
- tx_pulse_skew_est: Difference between the averages of the lower and upper threshold crossing times, estimating transmit pulse skew.
- tx_pulse_width_lower: Average distance between lower threshold crossing times measured by the Start Pulse Detector.
- tx_pulse_width_upper: Average distance between upper threshold crossing times measured by the Start Pulse Detector.
- velocity_sc: Spacecraft velocity components (east component, north component, up component) an observer on the ground would measure. While values are common to all beams, this parameter is naturally produced as part of geolocation.
- yaw: Spacecraft yaw (degrees), 3-2-1 Euler sequence.

Value

A data.table::data.table with one row per ATL03 geolocation segment, with class icesat2.atl03_seg_dt prepended. Columns include:

- beam - beam ID (e.g., gt2l).
- strong_beam - logical flag indicating whether the beam is classified as strong for the orbit.
- selected geolocation fields (see below).
- a derived h_ph column: height of the reference photon for each segment (one value per segment).

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file(
  "extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
```

```
)

atl03_h5 <- ATL03_read(atl03_path)

# Extract ATL03 geolocation segment metadata
atl03_segment_dt <- ATL03_seg_metadata_dt(atl03_h5)

head(atl03_segment_dt)
close(atl03_h5)

}
```

`ATL08_h5_clipBox` *Clip ICESat-2 ATL08 HDF5 Data Using a Bounding Extent*

Description

Clips an ICESat-2 ATL08 HDF5 file to a specified spatial extent. Only segment-level datasets within the ATL08 beam groups are clipped; all metadata, orbit information, and ancillary groups are preserved unchanged. This function is the bounding-box-based counterpart to geometry-based clipping functions.

Usage

```
ATL08_h5_clipBox(
  atl08,
  output,
  clip_obj,
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

`atl08`	An `icesat2.atl08_h5` object obtained via `ATL08_read()`, representing the ATL08 HDF5 file to be clipped.
`output`	Character path specifying where the clipped HDF5 file should be written.
`clip_obj`	Bounding extent used for clipping. Supported inputs: • A numeric vector of length 4: `c(ul_lat, ul_lon, lr_lat, lr_lon)`, following the ICESat-2 CMS bounding-box convention (upper-left latitude/longitude and lower-right latitude/longitude). • A `terra::SpatExtent` object.
`beam`	Character vector specifying which ATL08 beams to include. Defaults to: `c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")`.
`additional_groups`	Character vector specifying additional non-beam HDF5 groups to copy unchanged into the output file. Defaults to: `c("orbit_info")`.

Details

The function performs the following steps:

1. Copies file-level attributes and all requested non-beam groups.
2. Clips beam-level datasets based on latitude/longitude coordinates and the supplied spatial extent.
3. Reconstructs dependent indexing datasets (e.g., photon index ranges) when necessary to maintain valid ATL08 structure.

The resulting HDF5 file remains fully compliant with the ATL08 product structure, but contains only data within the specified bounding extent.

Value

An S4 object of class `icesat2.atl08_h5` representing the clipped ATL08 HDF5 file.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL08 data
atl08_h5 <- ATL08_read(atl08_path)

# Bounding rectangle coordinates (ul_lat, ul_lon, lr_lat, lr_lon)
ul_lon <- -106.5723
lr_lon <- -106.5693
lr_lat <- 41.533
ul_lat <- 41.537

# Clip ATL08 data using the bounding extent
atl08_clip <- ATL08_h5_clipBox(
  atl08_h5,
  output = tempfile(fileext = ".h5"),
  clip_obj = c(ul_lat, ul_lon, lr_lat, lr_lon)
)

close(atl08_h5)
close(atl08_clip)

}
```

`ATL08_h5_clipGeometry` *Clips ICESat-2 ATL08 data*

Description

This function clips ATL08 HDF5 file within beam groups, but keeps metada and ancillary data the same.

Clips an ICESat-2 ATL08 HDF5 file using one or more geometry-based clipping regions provided as a `terra::SpatVector`. Each feature (row) in `clip_obj` defines a separate clipping boundary. For every geometry, a new clipped ATL08 HDF5 file is created with the beam-level datasets spatially filtered to include only segments within the clipping region.

The resulting output filenames are automatically suffixed with the value of the attribute specified in `split_by`, enabling batch generation of multiple clipped ATL08 files from a single input.

Metadata, ancillary groups, and orbit information are preserved unchanged in every output file.

Usage

```
ATL08_h5_clipGeometry(
  atl08,
  output,
  clip_obj,
  split_by = "id",
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)

ATL08_h5_clipGeometry(
  atl08,
  output,
  clip_obj,
  split_by = "id",
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

`atl08`	An `icesat2.atl08_h5` object obtained via `ATL08_read()`, representing the ATL08 HDF5 file to be clipped.
`output`	Character string defining the base output filepath. The final output files will be named using: `paste0(output, "_", <split_by_value>, ".h5")`.
`clip_obj`	A `terra::SpatVector` containing the polygon or multipolygon geometries used as clipping regions. Each feature corresponds to one clipping output.
`split_by`	Character. Name of the attribute column in `clip_obj` used to uniquely identify each clipping region and to suffix output filenames. Defaults to `id`.

beam
: Character vector specifying which ATL08 beams to include. Defaults to: `c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")`.

additional_groups
: Character vector of additional non-beam groups to be copied unchanged into each output file. The default is: `c("orbit_info")`.

Details

Only the beam-level groups (e.g., `gt1l`, 'gt2r", ...) are spatially filtered. All metadata, orbit information, and ancillary ATL08 groups are preserved without modification.

The function:

1. Computes polygon-based masks for each beam-segment pair.
2. Removes all ATL08 segment datasets outside the clipping geometry.
3. Writes each clipped subset into its own ATL08 HDF5 file.
4. Returns the loaded results as S4 objects.

This is the geometry-based companion to bounding-box clipping functions in the package.

Value

Returns a list of clipped S4 object of class `icesat2.atl08_h5`

A list of clipped S4 objects of class `icesat2.atl08_h5`, one for each geometry in `clip_obj`.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

output <- tempfile(fileext = ".h5")

vect_path <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

clip_obj <- terra::vect(vect_path)

# Clipping ATL08 photons by boundary box extent
atl08_photons_dt_clip <- ATL08_h5_clipGeometry(
  atl08_h5,
  output,
  clip_obj,
```

```
  split_by = "id"
)

close(atl08_h5)
}
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL08 file path
ATL08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL08 file
ATL08_h5 <- ATL08_read(ATL08_path)

# Base output name (actual filenames will append split_by values)
output <- tempfile(fileext = ".h5")

# Load clipping polygons
clip_obj_path <- system.file("extdata", "clip_geom.shp",
  package = "ICESat2VegR"
)
clip_obj <- terra::vect(clip_obj_path)

# Clip ATL08 using polygon geometries
ATL08_clipped_list <- ATL08_h5_clipGeometry(
  ATL08_h5,
  output,
  clip_obj,
  split_by = "id"
)

close(ATL08_h5)

}
```

`ATL08_photons_attributes_dt` *ATL08 computed photons attributes*

Description

This function extracts computed photons attributes from ICESat-2 ATL08 data

Usage

```
ATL08_photons_attributes_dt(
  atl08_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
```

```
  photon_attributes = c("ph_segment_id", "classed_pc_indx", "classed_pc_flag", "ph_h",
    "d_flag", "delta_time")
)
```

Arguments

atl08_h5	A ICESat-2 ATL08 object (output of ATL08_read() function). An S4 object of class icesat2.atl08_dt.
beam	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
photon_attributes	character vector indicating the attributes to extract from the ATL08 photons. Default all c("ph_segment_id", "classed_pc_indx", "classed_pc_flag", "ph_h", "d_flag", "delta_time").

Details

These are the photons attributes extracted by default:

- **ph_segment_id**: Georeferenced bin number (20-m) associated with each photon
- **classed_pc_indx**: Indices of photons tracking back to ATL03 that surface finding software identified and used within the creation of the data products.
- **classed_pc_flag**: The L2B algorithm is run if this flag is set to 1 indicating data have sufficient waveform fidelity for L2B to run
- **ph_h**: Height of photon above interpolated ground surface
- **d_flag**: Flag indicating whether DRAGANN labeled the photon as noise or signal
- **delta_time**: Mid-segment GPS time in seconds past an epoch. The epoch is provided in the metadata at the file level

Value

Returns an S4 object of class data.table::data.table containing the ATL08 computed photons attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
```

```
atl08_h5 <- ATL08_read(atl08_path)

# Extracting ATL08 classified photons and heights
atl08_photons <- ATL08_photons_attributes_dt(atl08_h5 = atl08_h5)

head(atl08_photons)
close(atl08_h5)
}
```

`ATL08_photons_attributes_dt_LAS` *Export ICESat-2 ATL08 photon attributes to LAS files*

Description

Converts ICESat-2 ATL08 photon-level data stored as a data.table to one or more LAS files.

Usage

```
ATL08_photons_attributes_dt_LAS(atl08_dt, output)
```

Arguments

`atl08_dt`	A data.table containing ICESat-2 ATL08 photon attributes. The table must contain longitude, latitude, and photon height columns. Expected column names are `longitude`, `latitude`, and `ph_h`.
`output`	Character. Output LAS file path. If the data span multiple UTM zones, one LAS file is created per UTM zone.

Details

This function is designed for ATL08 photon-level data, including classified photon attributes such as photon height, classification flag, and photon segment ID. The input must contain longitude, latitude, and photon height information. Because LAS files require projected coordinates, the input longitude and latitude values are automatically split by UTM zone and reprojected before writing the LAS file. The internal UTM-zone assignment follows helper routines based on the latitude/longitude to UTM conversion algorithms developed by Chuck Gantz.

Value

Invisibly returns a character vector with the written LAS file paths.

References

Gantz, C. Latitude/Longitude to UTM Conversion Algorithms. https://www.gpsy.com/gpsinfo/geotoutm/

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# ATL08 photon attributes alone have no geolocation; join with ATL03 first
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
atl08_photons <- data.table::data.table(
  longitude = atl03_atl08_dt$lon_ph,
  latitude  = atl03_atl08_dt$lat_ph,
  ph_h      = atl03_atl08_dt$ph_h
)

ATL08_photons_attributes_dt_LAS(
  atl08_dt = atl08_photons,
  output = tempfile(fileext = ".las")
)

close(atl03_h5)
close(atl08_h5)

}
```

ATL08_read *Read ICESat-2 ATL08 data*

Description

Read the ICESat-2 Land and Vegetation Along-Track Product (ATL08) from an HDF5 (.h5) file.

Usage

```
ATL08_read(atl08_path)
```

Arguments

atl08_path	File path to an ICESat-2 ATL08 dataset in HDF5 format (.h5).

Value

An S4 object of class `icesat2.atl08_h5` containing ICESat-2 ATL08 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r007.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specify the path to an example ATL08 file
atl08_path <- system.file(
  "extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Read ICESat-2 ATL08 data (HDF5 file)
atl08 <- ATL08_read(atl08_path = atl08_path)
close(atl08)

}
```

`ATL08_read,character-method`

Read ICESat-2 ATL08 data from a local HDF5 file

Description

Read the ICESat-2 Land and Vegetation Along-Track Product (ATL08) from a local HDF5 (.h5) file.

Usage

```
## S4 method for signature 'character'
ATL08_read(atl08_path)
```

Arguments

`atl08_path`	Character. File path to ICESat-2 ATL08 data in HDF5 format (.h5).

Value

An S4 object of class `icesat2.atl08_h5` containing ICESat-2 ATL08 data.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specify the path to an example ATL08 file
atl08_path <- system.file(
  "extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Read ICESat-2 ATL08 data (HDF5 file)
atl08 <- ATL08_read(atl08_path = atl08_path)
```

```
close(atl08)

}
```

`ATL08_read,icesat2.granules_cloud-method`
Read ICESat-2 ATL08 data from multiple granules

Description

This method stops execution when multiple granules are provided, because the package currently supports only one granule at a time.

Usage

```
## S4 method for signature 'icesat2.granules_cloud'
ATL08_read(atl08_path)
```

Arguments

`atl08_path`	Object of class `icesat2.granules_cloud`. This parameter represents multiple ICESat-2 ATL08 granules.

Value

This method always stops with an error indicating that only one granule at a time is supported.

`ATL08_read,icesat2.granule_cloud-method`
Read ICESat-2 ATL08 data from a single granule in the cloud

Description

Read the ICESat-2 Land and Vegetation Along-Track Product (ATL08) from a single granule accessible in the cloud.

Usage

```
## S4 method for signature 'icesat2.granule_cloud'
ATL08_read(atl08_path)
```

Arguments

`atl08_path`	Object of class `icesat2.granule_cloud`. This parameter represents a single ICESat-2 ATL08 granule in the cloud.

Value

An S4 object of class `icesat2.atl08_h5` containing ICESat-2 ATL08 data.

ATL08_seg_attributes_dt

ATL08 Terrain and Canopy Attributes

Description

This function extracts terrain and canopy attributes by segments from ICESat-2 ATL08 data

Usage

```
ATL08_seg_attributes_dt(
  atl08_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  attributes = c("delta_time", "h_canopy", "canopy_openness", "h_te_mean",
    "terrain_slope")
)
```

Arguments

atl08_h5	A ICESat-2 ATL08 object (output of ATL08_read() function). An S4 object of class icesat2.atl08_dt.
beam	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
attributes	A character vector containing the list of terrain and canopy attributes to be extracted. Default is attribute = c("h_canopy","canopy_h_metrics","canopy_openness","h_te_mean","h_te_

Details

Segment-level attributes:

- segment_id_beg
- segment_id_end
- delta_time
- delta_time_beg
- delta_time_end
- latitude_20m
- longitude_20m
- rgt
- n_seg_ph
- ph_ndx_beg
- segment_landcover
- segment_snowcover
- segment_watermask
- surf_type
- urban_flag

- night_flag
- permafrost_alt
- permafrost_prob
- cloud_flag_atm
- cloud_fold_flag
- column_od_asr
- brightness_flag
- layer_flag
- msw_flag
- psf_flag
- sat_flag
- asr
- atlas_pa
- beam_azimuth
- beam_coelev
- solar_azimuth
- solar_elevation
- snr
- dem_flag
- dem_removal_flag
- dem_h
- h_dif_ref
- last_seg_extend
- sigma_h
- sigma_along
- sigma_across
- sigma_topo
- sigma_atlas_land

ATL08 Canopy attributes:

- can_noise
- can_quality_score
- canopy_h_metrics_abs
- canopy_openness
- h_canopy
- canopy_rh_conf
- h_median_canopy_abs
- h_min_canopy
- h_mean_canopy_abs
- h_median_canopy
- h_canopy_abs
- toc_roughness

- h_min_canopy_abs
- h_dif_canopy
- h_canopy_quad
- h_canopy_20m
- n_ca_photons
- photon_rate_can
- ‘photon_rate_can_nr“
- centroid_height
- canopy_h_metrics_abs
- h_mean_canopy
- subset_can_flag
- canopy_h_metrics
- n_toc_photons
- h_max_canopy_abs
- h_canopy_uncertainty
- canopy_openness
- h_max_canopy
- segment_cover

ATL08 Terrain attributes:

- h_te_best_fit
- h_te_best_fit_20m
- h_te_interp
- h_te_max
- h_te_mean
- h_te_median
- h_te_mode
- h_te_rh25
- h_te_skew
- h_te_std
- h_te_uncertainty
- n_te_photons
- photon_rate_te
- subset_te_flag
- terrain_slope
- te_quality

Value

Returns an S4 object of class icesat2.atl08_dt containing the ATL08-derived terrain and canopy attributes by segments.

See Also

https://nsidc.org/sites/default/files/documents/technical-reference/icesat2_atl08_atbd_v007.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08-derived terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)
head(atl08_seg_att_dt)

close(atl08_h5)
}
```

ATL08_seg_attributes_dt_clipBox

Clip ATL08 terrain and canopy attributes by bounding extent

Description

Clips ATL08 terrain and canopy segment attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
ATL08_seg_attributes_dt_clipBox(atl08_seg_att_dt, clip_obj)
```

Arguments

`atl08_seg_att_dt`	An `icesat2.atl08_dt` object (output of `ATL08_seg_attributes_dt()` function).
`clip_obj`	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees. • `SpatExtent` (package **terra**): the extent is used directly.

Details

When `clip_obj` is a `SpatExtent`, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns an S4 object of class icesat2.atl08_dt containing the clipped ATL08 terrain and canopy attributes.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to the ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path)

# Extracting ATL08-derived segment attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

# 1) Using a numeric bounding box: xmin ymin xmax ymax
bbox <- c(-103.7604, 59.4672, -103.7600, 59.4680)

atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipBox(
  atl08_seg_att_dt = atl08_seg_att_dt,
  clip_obj = bbox
)

# 2) Using a SpatExtent
ext <- terra::ext(-103.7604, -103.7600, 59.4672, 59.4680)
atl08_seg_att_dt_clip_ext <- ATL08_seg_attributes_dt_clipBox(
  atl08_seg_att_dt = atl08_seg_att_dt,
  clip_obj = ext
)

close(atl08_h5)

}
```

ATL08_seg_attributes_dt_clipGeometry

Clip ATL08 Terrain and Canopy Attributes by Geometry

Description

This function clips ATL08 Terrain and Canopy Attributes within a given geometry

Usage

```
ATL08_seg_attributes_dt_clipGeometry(
  atl08_seg_att_dt,
  clip_obj,
  split_by = NULL
)
```

Arguments

atl08_seg_att_dt	A atl08_seg_att_dt object (output of ATL08_seg_attributes_dt() function). An S4 object of class icesat2.atl08_dt
clip_obj	clip_obj. An object of class terra::SpatVector, which can be loaded as an ESRI shapefile using terra::vect function in the *sf* package.
split_by	clip_obj id. If defined, ATL08 data will be clipped by each clip_obj using the clip_obj id from table of attribute defined by the user

Value

Returns an S4 object of class icesat2.atl08_dt containing the clipped ATL08 Terrain and Canopy Attributes.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08-derived terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

clip_obj_path <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

if (require(terra)) {
  polygon <- terra::vect(clip_obj_path)

  head(atl08_seg_att_dt)
  # Clipping ATL08 Terrain and Canopy Attributes by Geometry
  atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipGeometry(
   atl08_seg_att_dt,
   polygon,
```

```
 split_by = "id"
)

hasLeaflet <- require(leaflet)

if (hasLeaflet) {
  leaflet() %>%
    addCircleMarkers(atl08_seg_att_dt_clip$longitude,
      atl08_seg_att_dt_clip$latitude,
      radius = 1,
      opacity = 1,
      color = "red"
    ) %>%
    addScaleBar(options = list(imperial = FALSE)) %>%
    addPolygons(
      data = polygon, weight = 1, col = "white",
      opacity = 1, fillOpacity = 0
    ) %>%
    addProviderTiles(providers$Esri.WorldImagery,
      options = providerTileOptions(minZoom = 3, maxZoom = 17)
    )
}
close(atl08_h5)
}

}
```

`ATL08_seg_attributes_dt_gridStat` *Statistics of ATL08 terrain and canopy attributes at grid level*

Description

Computes a series of user-defined descriptive statistics within each grid cell for ATL08 terrain and canopy attributes.

Usage

```
ATL08_seg_attributes_dt_gridStat(atl08_seg_att_dt, func, res = 0.5)
```

Arguments

`atl08_seg_att_dt`	An object of class `icesat2.atl08_dt` containing ATL08 data (output of `ATL08_seg_attributes_dt()` function).
`func`	The function to be applied for computing the defined statistics. Can be a single function (e.g. `mean(h_canopy)`) or a list of functions returning named metrics.
`res`	numeric. Spatial resolution in decimal degrees for the output `SpatRaster` layer. Default is `0.5`.

Value

Returns a SpatRaster object containing one layer per computed statistic of the selected ATL08 terrain and canopy attribute(s).

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08-derived terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

# Computing mean h_canopy at 0.0001 degree grid cell
library(terra)
mean_h_canopy <- ATL08_seg_attributes_dt_gridStat(
  atl08_seg_att_dt,
  func = mean(h_canopy),
  res = 0.0001
)
terra::plot(mean_h_canopy)

# Define a custom set of metrics
mySetOfMetrics <- function(x) {
  metrics <- list(
    min = min(x, na.rm = TRUE),
    max = max(x, na.rm = TRUE),
    mean = mean(x, na.rm = TRUE),
    sd = sd(x, na.rm = TRUE)
  )
  return(metrics)

# Computing h_canopy statistics at 0.05 degree grid cell
h_canopy_metrics <- ATL08_seg_attributes_dt_gridStat(
  atl08_seg_att_dt,
  func = mySetOfMetrics(h_canopy),
  res = 0.05
)
terra::plot(h_canopy_metrics)

close(atl08_h5)
}
}
```

ATL08_seg_attributes_dt_LAS *Export ICESat-2 ATL08 Segment Data to LAS Files*

Description

Converts ATL08 segment-level vegetation attributes extracted with `ATL08_seg_attributes_dt()` into one or more LAS files.

Usage

```
ATL08_seg_attributes_dt_LAS(atl08_dt, output)
```

Arguments

`atl08_dt`	An S4 object of class `ICESat2VegR::icesat2.atl08_dt-class`, typically the output of `ATL08_seg_attributes_dt()`.
`output`	Character. Output LAS file path. The function creates one LAS file per UTM zone in the WGS84 datum.

Details

This function exports ATL08 segment-level observations as LAS point clouds using longitude, latitude, and canopy height `h_canopy`.

Because LAS files require projected coordinates, the input geographic coordinates are automatically reprojected to the appropriate UTM coordinate reference system before export. If the data span multiple UTM zones, one LAS file is generated for each zone.

The internal UTM-zone assignment follows helper routines based on the latitude/longitude to UTM conversion algorithms developed by Chuck Gantz.

Value

Invisibly returns a character vector with the written LAS file paths.

References

Gantz, C. Latitude/Longitude to UTM Conversion Algorithms. https://www.gpsy.com/gpsinfo/geotoutm/

See Also

`ATL08_seg_attributes_dt`, `dt_to_las`

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

atl08_h5 <- ATL08_read(atl08_path = atl08_path)

atl08_dt <- ATL08_seg_attributes_dt(atl08_h5)

outputLas <- tempfile(fileext = ".las")

ATL08_seg_attributes_dt_LAS(
  atl08_dt,
  outputLas
)

close(atl08_h5)

}
```

ATL08_seg_attributes_dt_polyStat

Statistics of ATL08 Terrain and Canopy Attributes by Geometry

---

Description

Computes a series of statistics of ATL08 terrain and canopy attributes within area defined by a polygon

Usage

```
ATL08_seg_attributes_dt_polyStat(atl08_seg_att_dt, func, poly_id = NULL)
```

Arguments

atl08_seg_att_dt	An S4 object of class icesat2.atl08_dt containing ATL08 terrain and canopy attributes (output of ATL08_seg_attributes_dt() function).
func	The function to be applied for computing the defined statistics
poly_id	Polygon id. If defined, statistics will be computed for each polygon

Value

Returns an S4 object of class icesat2.atl08_dt Containing Statistics of ATL08 terrain and canopy attributes

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file(
  "extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08 terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

# Specifying the path to shapefile
polygon_filepath <- system.file(
  "extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

# Reading shapefile as sf object
polygon <- terra::vect(polygon_filepath)

# Clipping ATL08 terrain and canopy attributes by Geometry
atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipGeometry(
  atl08_seg_att_dt,
  polygon,
  split_by = "id"
)

# Computing the max h_canopy by polygon id
max_h_canopy <- ATL08_seg_attributes_dt_polyStat(
  atl08_seg_att_dt_clip,
  func = max(h_canopy),
  poly_id = "poly_id"
)
head(max_h_canopy)

# Define your own function
mySetOfMetrics <- function(x) {
  metrics <- list(
    min = min(x), # Min of x
    max = max(x), # Max of x
    mean = mean(x), # Mean of x
    sd = sd(x) # Sd of x
  )
  return(metrics)
}

# Computing a series of canopy statistics from customized function
```

```
h_canopy_metrics <- ATL08_seg_attributes_dt_polyStat(
  atl08_seg_att_dt_clip,
  func = mySetOfMetrics(h_canopy),
  poly_id = "poly_id"
)

head(h_canopy_metrics)

close(atl08_h5)
}
```

`ATL08_seg_attributes_h5_gridStat` *Rasterize ATL08 canopy attributes from h5 files at large scale*

Description

This function will read multiple ATL08 H5 files and create a stack of raster layers: count, and 1st, 2nd, 3rd and 4th moments (count, m1, m2, m3 and m4) for each metric selected, from which we can calculate statistics such as Mean, SD, Skewness and Kurtosis.

Usage

```
ATL08_seg_attributes_h5_gridStat(
  atl08_dir,
 metrics = c("h_canopy", "canopy_rh_conf", "h_median_canopy_abs", "h_min_canopy",
    "h_mean_canopy_abs", "h_median_canopy", "h_canopy_abs", "toc_roughness",
   "h_min_canopy_abs", "h_dif_canopy", "h_canopy_quad", "h_canopy_20m", "n_ca_photons",
   "photon_rate_can", "centroid_height", "canopy_h_metrics_abs", "h_mean_canopy",
    "subset_can_flag", "canopy_h_metrics", "n_toc_photons", "h_max_canopy_abs",
   "h_canopy_uncertainty", "canopy_openness", "h_max_canopy", "segment_cover"),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  out_root,
  clip_obj,
  res,
  creation_options = def_co,
  agg_function = default_agg_function,
  agg_join = default_agg_join,
  finalizer = default_finalizer
)
```

Arguments

`atl08_dir`	CharacterVector. The directory paths where the ATL08 H5 files are stored;
`metrics`	CharacterVector. A vector of canopy attributes available from ATL08 product (e.g. "h_canopy")
`beam`	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")

`out_root` Character. The root name for the raster output files, the pattern is {out_root}*{metric}*{count/m1/m2}.tif. This should include the full path for the file.

`clip_obj` Bounding extent or spatial object used to define the clipping area. Supported inputs:

- Numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees.
- `terra::SpatExtent`.
- `terra::SpatVector` (polygon/multipolygon).
- `sf` / `sfc` object.

When a polygon object is provided (`SpatVector`, `sf`, `sfc`), ATL08 segments are clipped using `ATL08_seg_attributes_dt_clipBox()`. Otherwise (numeric bbox or `SpatExtent`), clipping is done with `ATL08_seg_attributes_dt_clipBox()`.

`res` NumericVector. Resolution lon lat for the output raster in coordinates decimal degrees

`creation_options` CharacterVector. The GDAL creation options for the tif file. Default c("COMPRESS=PACKBITS", "BIGTIFF=IF_SAFER", "TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512") will create BIGTIFF if needed, with DEFLATE compression and tiled by 512x512 pixels.

`agg_function` Formula function-like. An aggregate function which should return a data.table with the aggregate statistics

`agg_join` Function. A function to merge two different agg objects.

`finalizer` List<name, formula>. A list with the final raster names and the formula which uses the base statistics.

Details

This function will create five different aggregate statistics (n, mean, variance, min, max). One can calculate mean and standard deviation with the following formulas according to Terriberry (2007).

The `agg_function` is a formula which return a data.table with the aggregate function to perform over the data. The default is:

```
~data.table(
    n = length(x),
    mean = mean(x,na.rm = TRUE),
    var = var(x) * (length(x) - 1),
    min = min(x, na.rm=T),
    max = max(x, na.rm=T)
  )
```

The `agg_join` is a function to merge two data.table aggregates from the `agg_function`. Since the h5 files will be aggregated one by one, the statistics from the different h5 files should have a function to merge them. The default function is:

```
function(x1, x2) {
    combined = data.table()
    x1$n[is.na(x1$n)] = 0
    x1$mean[is.na(x1$mean)] = 0
    x1$variance[is.na(x1$variance)] = 0
    x1$max[is.na(x1$max)] = -Inf
    x1$min[is.na(x1$min)] = Inf

    combined$n = x1$n + x2$n

    delta = x2$mean - x1$mean
    delta2 = delta * delta

    combined$mean = (x1$n * x1$mean + x2$n * x2$mean) / combined$n
    combined$variance = x1$variance + x2$variance +
      delta2 * x1$n * x2$n / combined$n

    combined$min = pmin(x1$min, x2$min, na.rm=F)
    combined$max = pmax(x1$max, x2$max, na.rm=F)
    return(combined)
}
```

The `finalizer` is a list of formulas to generate the final rasters based on the intermediate statistics from the previous functions. The default `finalizer` will calculate the `sd`, `skewness` and `kurtosis` based on the `variance`, `M3`, `M4` and `n` values. It is defined as:

```
list(
  sd = ~sqrt(variance/(n - 1)),
)
```

Value

Nothing. It outputs multiple raster tif files to the out_root specified path.

References

Terriberry, Timothy B. (2007), Computing Higher-Order Moments Online, archived from the original on 23 April 2014, retrieved 5 May 2008

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

library(data.table)

# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)
```

```
# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Bounding rectangle as numeric bbox: c(xmin, ymin, xmax, ymax)
clip_obj <- c(
  xmin = -106.5708541870117188,
  ymin =  41.5314979553222656,
  xmax = -106.5699081420898438,
  ymax =  41.5386848449707031
)

res <- 100 # meters
lat_to_met_factor <- 1 / 110540
lon_to_met_factor <- 1 / 111320
xres <- lon_to_met_factor * res
yres <- lat_to_met_factor * res

agg_function <- ~ data.table(
  min = min(x),
  max = max(x),
  sum = sum(x),
  n = length(x)
)

agg_join <- function(agg1, agg2) {
  agg1[is.na(agg1)] <- 0
  data.table(
    min = pmin(agg1$min, agg2$min),
    max = pmax(agg1$max, agg2$max),
    sum = agg1$sum + agg2$sum,
    n = agg1$n + agg2$n
  )
}

finalizer <- list(
  mean  = "sum/n",
  range = "max-min"
)

outdir <- file.path(tempdir(), "gridstat_out")
dir.create(outdir)
out_root <- file.path(outdir, "h_canopy")

ATL08_seg_attributes_h5_gridStat(
  atl08_dir        = dirname(atl08_path),
  metrics          = c("h_canopy"),
  out_root         = out_root,
  beam             = c("gt1l","gt1r","gt2l","gt2r","gt3l","gt3r"),
  clip_obj         = clip_obj,
  res              = c(xres, -yres),
  creation_options = c(
    "COMPRESS=DEFLATE",
```

```
    "BIGTIFF=IF_SAFER",
    "TILED=YES",
    "BLOCKXSIZE=512",
    "BLOCKYSIZE=512"
  ),
  agg_function     = agg_function,
  agg_join         = agg_join,
  finalizer        = finalizer
)

gc()
unlink(outdir, recursive = TRUE)
close(atl08_h5)

}
```

`ATLAS_dataDownload` *Download ICESat-2 ATL03/ATL08 data*

Description

Download ICESat-2 ATL03 and ATL08 data from the LP DAAC / NSIDC endpoints. Handles connection drops by resuming partial files and retrying with exponential backoff. Authentication is handled via a `.netrc` file: if it does not exist, the user is prompted for Earthdata Login credentials. If authentication fails (e.g., wrong username/password), the user is informed and can choose to re-enter credentials or cancel.

Usage

```
ATLAS_dataDownload(
  url,
  outdir = NULL,
  overwrite = FALSE,
  buffer_size = 512,
  timeout = 10,
  retries = 3,
  backoff = 2,
  quiet = FALSE
)
```

Arguments

`url`	Character vector; URLs to ATL03/ATL08 files (e.g., returned by `ATLAS_dataFinder()`).
`outdir`	Character; output directory (default: `tempdir()`).
`overwrite`	Logical; overwrite existing files? Default `FALSE`.
`buffer_size`	Integer; chunk size in KB per `readBin` call (default 512).
`timeout`	Numeric; connection timeout (seconds) for establishing the connection (default 10).

retries	Integer; maximum attempts per file (default 3).
backoff	Numeric; exponential backoff base used as sleep = backoff^(attempt - 1) (default 2).
quiet	Logical; suppress per-file messages (default FALSE).

Value

(invisibly) a data.frame with columns:

- file: file name
- dest: full destination path
- status: one of ok, failed, skipped
- attempts: number of attempts used
- error: error message (if any)

References

Credits to Cole Krehbiel. Code adapted from: https://git.earthdata.nasa.gov/projects/LPDUR/repos/daac_data_download_r/browse/DAACDataDownload.R

Examples

```
## Not run:
urls <- c(
  "https://example.com/path/to/ATL03_001.h5",
  "https://example.com/path/to/ATL08_001.h5"
)
status <- ATLAS_dataDownload(urls,
                             outdir        = "data",
                             retries       = 5,
                             backoff       = 2)
subset(status, status == "failed")

## End(Not run)
```

ATLAS_dataFinder *ICESat-2 ATL03 and ATL08 data finder for either direct download or cloud computing*

Description

This function finds the exact granule(s) that contain ICESat-2 ATLAS data for a given region of interest and date range

Usage

```
ATLAS_dataFinder(
  short_name,
  lower_left_lon,
  lower_left_lat,
  upper_right_lon,
  upper_right_lat,
  version = "006",
  daterange = NULL,
  persist = TRUE,
  cloud_hosted = TRUE,
  cloud_computing = FALSE
)
```

Arguments

short_name	ICESat-2 ATLAS data level short_name; Options: "ATL03", "ATL08",
lower_left_lon	Numeric. Minimum longitude in (decimal degrees) for the bounding box of the area of interest.
lower_left_lat	Numeric. Minimum latitude in (decimal degrees) for the bounding box of the area of interest.
upper_right_lon	Numeric. Maximum longitude in lon (decimal degrees) for the bounding box of the area of interest.
upper_right_lat	Numeric. Maximum latitude in (decimal degrees) for the bounding box of the area of interest.
version	Character. The version of the ICESat-2 ATLAS product files to be returned (only V005 or V006). Default "006".
daterange	Vector. Date range. Specify your start and end dates using ISO 8601 [YYYY]-[MM]-[DD]T[hh]:[mm]:[ss]Z. Ex.: c("2019-07-01T00:00:00Z","2020-05-22T23:59:59Z"). If NULL (default), the date range filter will be not applied.
persist	Logical. If TRUE, it will create the .netrc
cloud_hosted	Logical. Flag to indicate use of cloud hosted collections.To be used when the cloud computing parameter is FALSE.
cloud_computing	Logical. If TRUE, it will return granules for cloud computing directly, otherwise it will return links to be passed in the function ATLAS_dataDownload for data download.

Value

Return either a vector object pointing out the path to ICESat-2 ATLAS data found within the boundary box coordinates provided for data download or a vector object containing the granules hosted on the cloud for cloud computing directly.

See Also

bbox: Defined by the upper left and lower right corner coordinates, in lat,lon ordering, for the bounding box of the area of interest (e.g. lower_left_lon,lower_left_lat,upper_right_lon,upper_right_lat)

This function relies on the existing CMR tool: https://cmr.earthdata.nasa.gov/search/site/docs/search/api.html

Examples

```
# ICESat-2 data finder is a web service provided by NASA
# usually the request takes more than 5 seconds

# Specifying bounding box coordinates
lower_left_lon <- -96.0
lower_left_lat <- 40.0
upper_right_lon <- -96.5
upper_right_lat <- 40.5

# Specifying the date range
daterange <- c("2022-05-01", "2022-05-02")

# Extracting the path to ICESat-2 ATLAS data for the specified boundary box coordinates
# for data download

# Shouldn't be tested because it relies on a web service which might be down

## Not run:
ATLAS02b_list <- ATLAS_dataFinder(
  short_name = "ATL08",
  lower_left_lon,
  lower_left_lat,
  upper_right_lon,
  upper_right_lat,
  version = "007",
  daterange = daterange
)

## End(Not run)
```

---

build_ee_forest *Convert an R randomForest model to a Google Earth Engine randomForest classifier*

Description

Given a fitted randomForest::randomForest object, this function serializes the forest using the internal Rcpp module `icesat2_module` and constructs a corresponding Google Earth Engine random forest **classifier** via `ee$Classifier$decisionTreeEnsemble(rf_strings)`.

This implementation always returns an `ee$Classifier` and does not use `ee$Regressor`.

Usage

```
build_ee_forest(rf)
```

Arguments

rf	A fitted randomForest::randomForest model object.

Value

An Earth Engine `ee$Classifier` object created with `ee$Classifier$decisionTreeEnsemble()` that can be used with `ee$Image$classify()`.

---

clip — *Clip ICESat-2 ATL03/ATL08 data (h5, attributes, or ATL03-ATL08 join) by box or geometry*

Description

Unified clipping interface for ICESat-2 ATL03/ATL08 data. The generic `clip()` dispatches on the class of `x` and internally calls appropriate `_clipBox()` or `_clipGeometry()` helpers.

Depending on the product you are working with you can access its specific clipping methods:

ATL03_h5::

- `ATL03_h5_clipBox()`
- `ATL03_h5_clipGeometry()`

ATL08_h5::

- `ATL08_h5_clipBox()`
- `ATL08_h5_clipGeometry()`

ATL08_seg_attributes_dt::

- `ATL08_seg_attributes_dt_clipBox()`
- `ATL08_seg_attributes_dt_clipGeometry()`

ATL03_ATL08_photons_attributes_dt_join::

- `ATL03_ATL08_photons_attributes_dt_clipBox()`
- `ATL03_ATL08_photons_attributes_dt_clipGeometry()`

Usage

```
clip(x, clip_obj, ...)
```

Arguments

x	ICESat-2 object (ATL03/ATL08 h5, attributes, or joined table) Can be either: • icesat2.atl03_h5 • icesat2.atl03_dt • icesat2.atl03_seg_dt • icesat2.atl08_h5 • icesat2.atl08_dt • icesat2.atl03atl08_dt
clip_obj	Clipping object. Supported for box-based clipping: • numeric bounding box: c(xmin, ymin, xmax, ymax) • terra::SpatExtent For geometry-based clipping: • sf, sfc, terra::SpatVector, etc. • Will also accept additional argument split_by: – split_by Character. The SpatVector attribute whose values are used to identify and name each clipping geometry. Defaults to id. To clip by a spatial object's extent, extract its bbox first.
...	Additional arguments passed to _clipBox() or _clipGeometry(). For HDF5-based clipping, supported arguments include: • output Character. Path to the output HDF5 file that will store the clipped dataset. • beam Character vector specifying which ICESat-2 beams to include. Defaults to: c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r"). • additional_groups Character vector of additional non-beam HDF5 groups to copy unchanged into the output file. Defaults to: c("orbit_info").

---

clip,icesat2.atl03atl08_dt,ANY-method

Clip Joined ATL03 and ATL08 by Geometry

Description

This function clips joined ATL03 and ATL08 photon attributes within a given geometry.

Usage

```
## S4 method for signature 'icesat2.atl03atl08_dt,ANY'
clip(x, clip_obj, ...)
```

Arguments

x	An S4 object of class icesat2.atl03atl08_dt containing ATL03 and ATL08 data (output of ATL03_ATL08_photons_attributes_dt_join() function).
clip_obj	An object of class terra::SpatVector. which can be loaded as an ESRI shapefile using terra::vect function.
...	split_by Optional. clip_obj ID. If defined, the data will be clipped by each clip_obj using the clip_obj ID from the attribute table.

Value

Returns an S4 object of class icesat2.atl03atl08_dt containing the clipped ATL08 attributes.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path)
atl08_h5 <- ATL08_read(atl08_path)

# Joining ATL03 and ATL08 photon attributes
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# Specifying the path to the shapefile
clip_obj_filepath <- system.file("extdata", "clip_geom.shp", package = "ICESat2VegR")

# Reading shapefile as a SpatVector object
clip_obj <- terra::vect(clip_obj_filepath)

# Clipping ATL08 terrain attributes by geometry
atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipGeometry(
  atl03_atl08_dt,
  clip_obj,
  split_by = "id"
)
head(atl03_atl08_dt_clip)

close(atl03_h5)
close(atl08_h5)
}
```

clip,icesat2.atl03atl08_dt,numeric-method

Clip joined ATL03 and ATL08 photons by bounding extent

Description

Clips joined ATL03 and ATL08 photon attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
## S4 method for signature 'icesat2.atl03atl08_dt,numeric'
clip(x, clip_obj, ...)
```

Arguments

`x`	An S4 object of class `icesat2.atl03atl08_dt` containing joined ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`clip_obj`	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees. • `SpatExtent` (package **terra**): the extent is used directly.
`...`	not used.

Details

When `clip_obj` is a `SpatExtent`, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns an S4 object of class `icesat2.atl03atl08_dt` containing a subset of the joined ATL03 and ATL08 photon attributes restricted to the clipping extent.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl08_path <- system.file("extdata", "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Joining ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# 1) Using a numeric bounding box: xmin ymin xmax ymax
bbox <- c(-103.7604, 59.4672, -103.7600, 59.4680)

atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt = atl03_atl08_dt,
  clip_obj = bbox
)
head(atl03_atl08_dt_clip)

# 2) Using a SpatExtent (example)
ext <- terra::ext(-103.7604, -103.7600, 59.4672, 59.4680)
atl03_atl08_dt_clip_ext <- ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt = atl03_atl08_dt,
  clip_obj = ext
)

close(atl03_h5)
close(atl08_h5)

}
```

`clip,icesat2.atl03atl08_dt,SpatExtent-method`

Clip joined ATL03 and ATL08 photons by bounding extent

Description

Clips joined ATL03 and ATL08 photon attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
## S4 method for signature 'icesat2.atl03atl08_dt,SpatExtent'
clip(x, clip_obj, ...)
```

Arguments

x	An S4 object of class icesat2.atl03atl08_dt containing joined ATL03 and ATL08 data (output of ATL03_ATL08_photons_attributes_dt_join() function).
clip_obj	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: c(xmin, ymin, xmax, ymax) in decimal degrees. • SpatExtent (package **terra**): the extent is used directly.
...	not used.

Details

When clip_obj is a SpatExtent, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns an S4 object of class icesat2.atl03atl08_dt containing a subset of the joined ATL03 and ATL08 photon attributes restricted to the clipping extent.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl08_path <- system.file("extdata", "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Joining ATL03 and ATL08 photons and heights
```

```
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# 1) Using a numeric bounding box: xmin ymin xmax ymax
bbox <- c(-103.7604, 59.4672, -103.7600, 59.4680)

atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt = atl03_atl08_dt,
  clip_obj = bbox
)
head(atl03_atl08_dt_clip)

# 2) Using a SpatExtent (example)
ext <- terra::ext(-103.7604, -103.7600, 59.4672, 59.4680)
atl03_atl08_dt_clip_ext <- ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt = atl03_atl08_dt,
  clip_obj = ext
)

close(atl03_h5)
close(atl08_h5)
}
```

`clip,icesat2.atl03_dt,ANY-method`

Clip ATL03 photons by Coordinates

Description

This function clips ATL03 photon attributes within given bounding coordinates.

Usage

```
## S4 method for signature 'icesat2.atl03_dt,ANY'
clip(x, clip_obj, ...)
```

Arguments

`x`	An ATL03 photon data table. An S4 object of class `icesat2.atl03_dt`.
`clip_obj`	Spatial clip_obj. An object of class `terra::SpatVector`, which can be loaded as an ESRI shapefile using the terra::vect function in the *sf* package.
`...`	The `split_by` parameter can be specified to clip_obj by attribute. If defined, GEDI data will be clipped by each clip_obj using the clip_obj id from the attribute table defined by the user.

Value

Returns an S4 object of class `icesat2.atl03_dt` containing the ATL03 photon attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Extracting ATL03 photon attributes
atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5 = atl03_h5)

# Specifying the path to shapefile
clip_obj_filepath <-
  system.file(
    "extdata",
    "clip_geom.shp",
    package = "ICESat2VegR"
  )

# Reading shapefile as sf object
clip_obj <- terra::vect(clip_obj_filepath)

# Clipping ATL03 photon attributes by Geometry
atl03_photons_dt_clip <-
  ATL03_photons_attributes_dt_clipGeometry(atl03_photons_dt, clip_obj, split_by = "id")

head(atl03_photons_dt_clip)

close(atl03_h5)
}
```

---

clip,icesat2.atl03_dt,numeric-method

Clip ATL03 photons by bounding extent

Description

Clips ATL03 photon attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
## S4 method for signature 'icesat2.atl03_dt,numeric'
clip(x, clip_obj, ...)
```

Arguments

x	An icesat2.atl03_dt object (output of ATL03_photons_attributes_dt()).
clip_obj	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: c(xmin, ymin, xmax, ymax) in decimal degrees. • SpatExtent (package **terra**): the extent is used directly.
...	not used

Details

When clip_obj is a SpatExtent, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns a subset of the original icesat2.atl03_dt object containing only photons within the bounding box.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file("extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5 = atl03_h5)

# 1) Using a numeric bbox: xmin ymin xmax ymax
bbox <- c(-106.57, 41.53, -106.5698, 41.54)
atl03_clip_bbox <- ATL03_photons_attributes_dt_clipBox(
  atl03_photons_dt = atl03_photons_dt,
  clip_obj = bbox
)

# 2) Using a SpatExtent
ext <- terra::ext(-106.57, -106.5698, 41.53, 41.54)
atl03_clip_ext <- ATL03_photons_attributes_dt_clipBox(
  atl03_photons_dt = atl03_photons_dt,
  clip_obj = ext
```

```
)

close(atl03_h5)
}
```

`clip,icesat2.atl03_dt,SpatExtent-method`
Clip ATL03 photons by bounding extent

Description

Clips ATL03 photon attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
## S4 method for signature 'icesat2.atl03_dt,SpatExtent'
clip(x, clip_obj, ...)
```

Arguments

`x`	An `icesat2.atl03_dt` object (output of `ATL03_photons_attributes_dt()`).
`clip_obj`	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees. • `SpatExtent` (package **terra**): the extent is used directly.
`...`	not used

Details

When `clip_obj` is a `SpatExtent`, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns a subset of the original `icesat2.atl03_dt` object containing only photons within the bounding box.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file("extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5 = atl03_h5)

# 1) Using a numeric bbox: xmin ymin xmax ymax
bbox <- c(-106.57, 41.53, -106.5698, 41.54)
atl03_clip_bbox <- ATL03_photons_attributes_dt_clipBox(
  atl03_photons_dt = atl03_photons_dt,
  clip_obj = bbox
)

# 2) Using a SpatExtent
ext <- terra::ext(-106.57, -106.5698, 41.53, 41.54)
atl03_clip_ext <- ATL03_photons_attributes_dt_clipBox(
  atl03_photons_dt = atl03_photons_dt,
  clip_obj = ext
)

close(atl03_h5)
}
```

---

clip,icesat2.atl03_h5,ANY-method

Clip ICESat-2 ATL03 HDF5 Data Using Geometry-Based Boundaries

---

Description

Clips an ICESat-2 ATL03 HDF5 file using one or more geometry-based clipping objects provided as a `terra::SpatVector`. Each geometry in `vect` defines an individual clipping region. The function iteratively clips the ATL03 data for each region and writes a separate output HDF5 file for every geometry. The name of each output file is automatically appended with the value of the attribute specified in `split_by`.

Only datasets within the ATL03 beam groups are spatially clipped; metadata and ancillary non-beam groups remain unchanged in the resulting files.

Usage

```
## S4 method for signature 'icesat2.atl03_h5,ANY'
clip(x, clip_obj, ...)
```

Arguments

x	An icesat2.atl03_h5 object obtained using ATL03_read(), representing the ATL03 HDF5 file to be clipped.
clip_obj	A terra::SpatVector object containing the geometric clip boundaries. Each feature (row) in the SpatVector defines an independent clipping region.
...	Additional arguments: • output Character. Path to the output filename. The final written files will append the unique value of split_by for each clipping geometry, for example: output_id1.h5, output_id2.h5, etc. • split_by Character. The SpatVector attribute whose values are used to identify and name each clipping geometry. Defaults to id. • beam Character vector specifying which ATL03 beams to include. The default includes all six beams: c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r"). • additional_groups Character vector of non-beam HDF5 groups that should be copied unchanged into each clipped file. Defaults to: c("orbit_info").

Value

Returns a list of clipped S4 objects of class icesat2.atl03_h5, one for each clipping geometry contained in vect.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL03 file
atl03_h5 <- ATL03_read(atl03_path)

# Output base filename
output <- tempfile(fileext = ".h5")

# Load clipping geometries
vect_path <- system.file("extdata", "clip_geom.shp",
  package = "ICESat2VegR"
)
vect <- terra::vect(vect_path)

# Clip ATL03 data using polygon geometries
atl03_clipped_list <- ATL03_h5_clipGeometry(
  atl03_h5,
  output,
  vect,
```

```
  split_by = "id"
)

close(atl03_h5)

}
```

---

clip,icesat2.atl03_h5,numeric-method

Clip ICESat-2 ATL03 HDF5 Data Using a Bounding Extent

Description

Clips an ICESat-2 ATL03 HDF5 file to a specified spatial extent. Only geolocated photon and segment datasets within the ATL03 beam groups are clipped; all metadata, orbit information, and ancillary groups are preserved unchanged. This function provides bounding-box-based clipping, complementing geometry-based clipping workflows.

Usage

```
## S4 method for signature 'icesat2.atl03_h5,numeric'
clip(x, clip_obj, ...)
```

Arguments

`x`	An `icesat2.atl03_h5` object created by `ATL03_read()`, representing the ATL03 HDF5 file to be clipped.
`clip_obj`	Bounding extent used for clipping. Supported inputs: • A numeric vector of length 4: `c(ul_lat, ul_lon, lr_lat, lr_lon)`, following the ICESat-2 CMS bounding-box convention (upper-left latitude/longitude and lower-right latitude/longitude). • A `terra::SpatExtent` object.
`...`	Additional arguments: • `output` Character. Path to the output HDF5 file that will store the clipped ATL03 dataset. • `beam` Character vector specifying which ATL03 beams to include. Defaults to: `c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")`. • `additional_groups` Character vector of additional non-beam HDF5 groups to copy unchanged into the output file. Defaults to: `c("orbit_info")`.

Details

The function performs spatial subsetting at the photon and segment level. Internally, it:

1. Copies file- and group-level attributes.

2. Extracts beam-level datasets and evaluates segment-level and photon-level masks using the supplied bounding box.
3. Applies these masks to all datasets whose dimensions correspond to segments or photons.
4. Recomputes dependent indexing datasets (e.g., `ph_index_beg`) to maintain ATL03 structural integrity.
5. Writes the clipped data into a new, valid ATL03 HDF5 file.

This function preserves the full ATL03 metadata, ancillary information, and file structure while trimming the spatial domain to the user-specified bounding extent.

Value

An S4 object of class `icesat2.atl03_h5` representing the clipped ATL03 HDF5 file saved at the `output` location.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL03 data (HDF5)
atl03_h5 <- ATL03_read(atl03_path)

# Bounding rectangle coordinates (ul_lat, ul_lon, lr_lat, lr_lon)
xmin <- -106.5723
xmax <- -106.5693
ymin <- 41.533
ymax <- 41.537

bbox <- c(ymax, xmin, ymin, xmax)

# Clip ATL03 using the bounding extent
output <- tempfile(fileext = ".h5")
atl03_clip <- ATL03_h5_clipBox(
  atl03_h5,
  output = output,
  clip_obj = bbox
)

close(atl03_h5)

}
```

`clip,icesat2.atl03_h5,SpatExtent-method`
Clip ICESat-2 ATL03 HDF5 Data Using a Bounding Extent

Description

Clips an ICESat-2 ATL03 HDF5 file to a specified spatial extent. Only geolocated photon and segment datasets within the ATL03 beam groups are clipped; all metadata, orbit information, and ancillary groups are preserved unchanged. This function provides bounding-box-based clipping, complementing geometry-based clipping workflows.

Usage

```
## S4 method for signature 'icesat2.atl03_h5,SpatExtent'
clip(x, clip_obj, ...)
```

Arguments

`x`	An `icesat2.atl03_h5` object created by `ATL03_read()`, representing the ATL03 HDF5 file to be clipped.
`clip_obj`	Bounding extent used for clipping. Supported inputs: • A numeric vector of length 4: `c(ul_lat, ul_lon, lr_lat, lr_lon)`, following the ICESat-2 CMS bounding-box convention (upper-left latitude/longitude and lower-right latitude/longitude). • A `terra::SpatExtent` object.
`...`	Additional arguments: • `output` Character. Path to the output HDF5 file that will store the clipped ATL03 dataset. • `beam` Character vector specifying which ATL03 beams to include. Defaults to: `c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")`. • `additional_groups` Character vector of additional non-beam HDF5 groups to copy unchanged into the output file. Defaults to: `c("orbit_info")`.

Details

The function performs spatial subsetting at the photon and segment level. Internally, it:

1. Copies file- and group-level attributes.
2. Extracts beam-level datasets and evaluates segment-level and photon-level masks using the supplied bounding box.
3. Applies these masks to all datasets whose dimensions correspond to segments or photons.
4. Recomputes dependent indexing datasets (e.g., `ph_index_beg`) to maintain ATL03 structural integrity.
5. Writes the clipped data into a new, valid ATL03 HDF5 file.

This function preserves the full ATL03 metadata, ancillary information, and file structure while trimming the spatial domain to the user-specified bounding extent.

Value

An S4 object of class icesat2.atl03_h5 representing the clipped ATL03 HDF5 file saved at the output location.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL03 data (HDF5)
atl03_h5 <- ATL03_read(atl03_path)

# Bounding rectangle coordinates (ul_lat, ul_lon, lr_lat, lr_lon)
xmin <- -106.5723
xmax <- -106.5693
ymin <- 41.533
ymax <- 41.537

bbox <- c(ymax, xmin, ymin, xmax)

# Clip ATL03 using the bounding extent
output <- tempfile(fileext = ".h5")
atl03_clip <- ATL03_h5_clipBox(
  atl03_h5,
  output = output,
  clip_obj = bbox
)

close(atl03_h5)


}
```

---

clip,icesat2.atl08_dt,ANY-method

Clip ATL08 Terrain and Canopy Attributes by Geometry

Description

This function clips ATL08 Terrain and Canopy Attributes within a given geometry

Usage

```
## S4 method for signature 'icesat2.atl08_dt,ANY'
clip(x, clip_obj, ...)
```

Arguments

x	An icesat2.atl08_dt object (output of ATL08_seg_attributes_dt() function).
clip_obj	clip_obj. An object of class terra::SpatVector, which can be loaded as an ESRI shapefile using terra::vect function in the *sf* package.
...	The split_by parameter can be specified to clip_obj id. If defined, ATL08 data will be clipped by each clip_obj using the clip_obj id from table of attribute defined by the user

Value

Returns an S4 object of class icesat2.atl08_dt containing the clipped ATL08 Terrain and Canopy Attributes.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08-derived terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

clip_obj_path <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

if (require(terra)) {
  polygon <- terra::vect(clip_obj_path)

  head(atl08_seg_att_dt)
  # Clipping ATL08 Terrain and Canopy Attributes by Geometry
  atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipGeometry(atl08_seg_att_dt,
    polygon,
    split_by = "id"
  )

  hasLeaflet <- require(leaflet)

  if (hasLeaflet) {
    leaflet() %>%
      addCircleMarkers(atl08_seg_att_dt_clip$longitude,
        atl08_seg_att_dt_clip$latitude,
        radius = 1,
        opacity = 1,
```

```
        color = "red"
      ) %>%
      addScaleBar(options = list(imperial = FALSE)) %>%
      addPolygons(
        data = polygon, weight = 1, col = "white",
        opacity = 1, fillOpacity = 0
      ) %>%
      addProviderTiles(providers$Esri.WorldImagery,
        options = providerTileOptions(minZoom = 3, maxZoom = 17)
      )
   }
  }
  close(atl08_h5)

 }
```

`clip,icesat2.atl08_dt,numeric-method`
Clip ATL08 terrain and canopy attributes by bounding extent

Description

Clips ATL08 terrain and canopy segment attributes within a given bounding extent, defined by a single `clip_obj` argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
## S4 method for signature 'icesat2.atl08_dt,numeric'
clip(x, clip_obj, ...)
```

Arguments

`x`	An `icesat2.atl08_dt` object (output of `ATL08_seg_attributes_dt()` function).
`clip_obj`	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: `c(xmin, ymin, xmax, ymax)` in decimal degrees. • `SpatExtent` (package **terra**): the extent is used directly.
`...`	not used.

Details

When `clip_obj` is a `SpatExtent`, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns an S4 object of class icesat2.atl08_dt containing the clipped ATL08 terrain and canopy attributes.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to the ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path)

# Extracting ATL08-derived segment attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

# 1) Using a numeric bounding box: xmin ymin xmax ymax
bbox <- c(-103.7604, 59.4672, -103.7600, 59.4680)

atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipBox(
  atl08_seg_att_dt = atl08_seg_att_dt,
  clip_obj = bbox
)

# 2) Using a SpatExtent
ext <- terra::ext(-103.7604, -103.7600, 59.4672, 59.4680)
atl08_seg_att_dt_clip_ext <- ATL08_seg_attributes_dt_clipBox(
  atl08_seg_att_dt = atl08_seg_att_dt,
  clip_obj = ext
)

close(atl08_h5)

}
```

---

clip,icesat2.atl08_dt,SpatExtent-method

Clip ATL08 terrain and canopy attributes by bounding extent

Description

Clips ATL08 terrain and canopy segment attributes within a given bounding extent, defined by a single clip_obj argument. The clipping extent can be provided as:

- (i) a numeric bounding box, or
- (ii) a spatial extent object.

Usage

```
## S4 method for signature 'icesat2.atl08_dt,SpatExtent'
clip(x, clip_obj, ...)
```

Arguments

x	An icesat2.atl08_dt object (output of ATL08_seg_attributes_dt() function).
clip_obj	Bounding extent used to perform the clipping. Supported inputs: • Numeric vector of length 4: c(xmin, ymin, xmax, ymax) in decimal degrees. • SpatExtent (package **terra**): the extent is used directly.
...	not used.

Details

When clip_obj is a SpatExtent, the package **terra** must be installed. If not found, the function stops with an informative message.

Value

Returns an S4 object of class icesat2.atl08_dt containing the clipped ATL08 terrain and canopy attributes.

---

clip,icesat2.atl08_h5,ANY-method

Clips ICESat-2 ATL08 data

Description

This function clips ATL08 HDF5 file within beam groups, but keeps metada and ancillary data the same.

Usage

```
## S4 method for signature 'icesat2.atl08_h5,ANY'
clip(x, clip_obj, ...)
```

Arguments

x	icesat2.atl08_h5 object, obtained through ATL08_read() for clipping
clip_obj	terra::SpatVector for clipping
...	Additional arguments: • output character. Path to the output h5 file.

- split_by character. The attribute name used for identifying the different clip_objs. Default is "id"
- beam character. The vector of beams to include, default all c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")
- additional_groups character. Other addional groups that should be included, default c("orbit_info")

Value

Returns a list of clipped S4 object of class icesat2.atl08_h5

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

output <- tempfile(fileext = ".h5")

vect_path <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

vect <- terra::vect(vect_path)

# Clipping ATL08 photons by boundary box extent
atl08_photons_dt_clip <- ATL08_h5_clipGeometry(
  atl08_h5,
  output,
  vect,
  split_by = "id"
)

close(atl08_h5)

}
```

---

clip,icesat2.atl08_h5,numeric-method

Clip ICESat-2 ATL08 HDF5 Data Using a Bounding Extent

Description

Clips an ICESat-2 ATL08 HDF5 file to a specified spatial extent. Only segment-level datasets within the ATL08 beam groups are clipped; all metadata, orbit information, and ancillary groups are preserved unchanged. This function is the bounding-box-based counterpart to geometry-based clipping functions.

Usage

```
## S4 method for signature 'icesat2.atl08_h5,numeric'
clip(x, clip_obj, ...)
```

Arguments

`x`	An `icesat2.atl08_h5` object obtained via `ATL08_read()`, representing the ATL08 HDF5 file to be clipped.
`clip_obj`	Bounding extent used for clipping. Supported inputs: • A numeric vector of length 4: `c(ul_lat, ul_lon, lr_lat, lr_lon)`, following the ICESat-2 CMS bounding-box convention (upper-left latitude/longitude and lower-right latitude/longitude). • A `terra::SpatExtent` object.
`...`	Additional arguments: • `output` Character path specifying where the clipped HDF5 file should be written. • `beam` Character vector specifying which ATL08 beams to include. Defaults to: `c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")`. • `additional_groups` Character vector specifying additional non-beam HDF5 groups to copy unchanged into the output file. Defaults to: `c("orbit_info")`.

Details

The function performs the following steps:

1. Copies file-level attributes and all requested non-beam groups.
2. Clips beam-level datasets based on latitude/longitude coordinates and the supplied spatial extent.
3. Reconstructs dependent indexing datasets (e.g., photon index ranges) when necessary to maintain valid ATL08 structure.

The resulting HDF5 file remains fully compliant with the ATL08 product structure, but contains only data within the specified bounding extent.

Value

An S4 object of class `icesat2.atl08_h5` representing the clipped ATL08 HDF5 file.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL08 data
atl08_h5 <- ATL08_read(atl08_path)

# Bounding rectangle coordinates (ul_lat, ul_lon, lr_lat, lr_lon)
ul_lon <- -106.5723
lr_lon <- -106.5693
lr_lat <- 41.533
ul_lat <- 41.537

# Clip ATL08 data using the bounding extent
atl08_clip <- ATL08_h5_clipBox(
  atl08_h5,
  output = tempfile(fileext = ".h5"),
  clip_obj = c(ul_lat, ul_lon, lr_lat, lr_lon)
)

close(atl08_h5)
close(atl08_clip)


}
```

---

clip,icesat2.atl08_h5,SpatExtent-method

Clip ICESat-2 ATL08 HDF5 Data Using a Bounding Extent

Description

Clips an ICESat-2 ATL08 HDF5 file to a specified spatial extent. Only segment-level datasets within the ATL08 beam groups are clipped; all metadata, orbit information, and ancillary groups are preserved unchanged. This function is the bounding-box-based counterpart to geometry-based clipping functions.

Usage

```
## S4 method for signature 'icesat2.atl08_h5,SpatExtent'
clip(x, clip_obj, ...)
```

Arguments

x	An icesat2.atl08_h5 object obtained via ATL08_read(), representing the ATL08 HDF5 file to be clipped.
clip_obj	Bounding extent used for clipping. Supported inputs: • A numeric vector of length 4: c(ul_lat, ul_lon, lr_lat, lr_lon), following the ICESat-2 CMS bounding-box convention (upper-left latitude/longitude and lower-right latitude/longitude). • A terra::SpatExtent object.
...	Additional arguments: • output Character path specifying where the clipped HDF5 file should be written. • beam Character vector specifying which ATL08 beams to include. Defaults to: c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r"). • additional_groups Character vector specifying additional non-beam HDF5 groups to copy unchanged into the output file. Defaults to: c("orbit_info").

Details

The function performs the following steps:

1. Copies file-level attributes and all requested non-beam groups.
2. Clips beam-level datasets based on latitude/longitude coordinates and the supplied spatial extent.
3. Reconstructs dependent indexing datasets (e.g., photon index ranges) when necessary to maintain valid ATL08 structure.

The resulting HDF5 file remains fully compliant with the ATL08 product structure, but contains only data within the specified bounding extent.

Value

An S4 object of class icesat2.atl08_h5 representing the clipped ATL08 HDF5 file.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Read ATL08 data
atl08_h5 <- ATL08_read(atl08_path)

# Bounding rectangle coordinates (ul_lat, ul_lon, lr_lat, lr_lon)
ul_lon <- -106.5723
```

```
lr_lon <- -106.5693
lr_lat <- 41.533
ul_lat <- 41.537

# Clip ATL08 data using the bounding extent
atl08_clip <- ATL08_h5_clipBox(
  atl08_h5,
  output = tempfile(fileext = ".h5"),
  clip_obj = c(ul_lat, ul_lon, lr_lat, lr_lon)
)

close(atl08_h5)
close(atl08_clip)


}
```

`close,icesat2.h5-method`

Safely closes the ICESat2.h5 base classes

Description

Closing files will avoid locking HDF5 ATL03 files.

Closing files will avoid locking HDF5 files.

Usage

```
## S4 method for signature 'icesat2.h5'
close(con, ...)

## S4 method for signature 'icesat2.predict_h5'
close(con, ...)
```

Arguments

con	An object of class icesat2.predict_h5
...	Inherited from base, not used

close.GDALDataset *Method to close GDALDataset*

Description

Closing ensures the data is actually flushed (saved) to the dataset also it releases the lock from the file.

Usage

```
## S3 method for class 'GDALDataset'
close(con, ...)
```

Arguments

con	GDALDataset to be closed.
...	not used, inherited from generic close function.

close.icesat2.predict_h5 *Closes the HDF5 file pointer to release resources*

Description

Closes the HDF5 file pointer to release resources

Usage

```
## S3 method for class 'icesat2.predict_h5'
close(con, ...)
```

Arguments

con	The HDF5 file pointer of class icesat2.predict_h5
...	Additional parameters inherited from generic

Value

Nothing, just closes the HDF5 file pointer

createDataset *Creates a new GDALDataset*

Description

Create a new raster file based on specified data. It will output a *.tif file.

Usage

```
createDataset(
  raster_path,
  nbands,
  datatype,
  projstring,
  lr_lat,
  ul_lat,
  ul_lon,
  lr_lon,
  res,
  nodata,
  co = c("TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512", "COMPRESSION=LZW")
)
```

Arguments

raster_path	Character. The output path for the raster data
nbands	Integer. Number of bands. Default 1.
datatype	GDALDataType. The GDALDataType to use for the raster, use (GDALDataType$) to find the options. Default GDALDataType$GDT_Float64
projstring	The projection string, either proj or WKT is accepted.
lr_lat	Numeric. The lower right latitude.
ul_lat	Numeric. The upper left latitude.
ul_lon	Numeric. The upper left longitude.
lr_lon	Numeric. The lower right longitude.
res	Numeric. The resolution of the output raster
nodata	Numeric. The no data value for the raster. Default 0.
co	CharacterVector. A CharacterVector of creation options for GDAL. Default NULL

Value

An object from GDALDataset R6 class.

Examples

```
# Parameters
raster_path <- tempfile(fileext = ".tif")
ul_lat <- -15
ul_lon <- -45
lr_lat <- -25
lr_lon <- -35
res <- c(0.01, -0.01)
datatype <- GDALDataType$GDT_Int32
nbands <- 1
projstring <- "EPSG:4326"
nodata <- -1
co <- c("TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512", "COMPRESSION=LZW")

# Create a new raster dataset
ds <- createDataset(
  raster_path = raster_path,
  nbands = nbands,
  datatype = datatype,
  projstring = projstring,
  lr_lat = lr_lat,
  ul_lat = ul_lat,
  ul_lon = ul_lon,
  lr_lon = lr_lon,
  res = res,
  nodata = nodata,
  co = co
)

# Get the GDALRasterBand for ds
band <- ds[[1]]

# Set some dummy values
band[[0, 0]] <- 1:(512 * 512)

ds$Close()
```

---

dt_to_las *Convert ICESat-2 ATL03 and ATL08 data.table objects to LAS files*

Description

Converts ICESat-2 ATL03 or ATL08 point data stored as a data.table with longitude, latitude, and height values into one or more LAS files.

Usage

```
dt_to_las(dt, output)
```

Arguments

dt	A data.table with at least three columns: longitude, latitude, and height. The first three columns are interpreted as longitude, latitude, and height.
output	Character. Output LAS file path. If the data span multiple UTM zones, the EPSG code is appended to each output file name.

Details

The input data must contain at least three columns representing longitude, latitude, and height. The function renames the first three columns to X, Y, and Z internally. The original X and Y values are assumed to be longitude and latitude in EPSG:4326.

Because LAS files should store projected coordinates, the function identifies the appropriate UTM zone for each point, splits the dataset by UTM zone, projects each subset to the corresponding UTM coordinate system, and writes one LAS file per UTM zone.

Value

Invisibly returns a character vector with the written LAS file paths.

Examples

```
library(data.table)

dt <- data.table(
  lon = runif(1000, -52.5, -52.4),
  lat = runif(1000, -15.9, -15.8),
  h = runif(1000, 0, 35)
)

dt_to_las(dt, tempfile(fileext = ".las"))
```

---

earthdata_login *Get or Create a* .netrc *File for NASA Earthdata Login*

Description

Creates (if necessary) and configures a .netrc file containing credentials for **NASA Earthdata Login** (<urs.earthdata.nasa.gov>). This file is required for authenticated downloads from Earthdata services (e.g., via earthaccess, curl, wget, or other clients).

Usage

```
earthdata_login(output_dir = tempdir())
```

Arguments

output_dir	Character. Directory where the .netrc file should be created. Defaults to tempdir() so credentials are not written to the user's home filespace unless explicitly requested; pass e.g. "~" yourself if you want the credentials to persist across sessions.

Details

The function will:

- Use an existing .netrc file if valid credentials are already present
- Otherwise prompt the user securely for credentials
- Or read credentials from environment variables:
  - EARTHDATA_USERNAME
  - EARTHDATA_PASSWORD
- Set the NETRC environment variable for the current R process

Credentials are written in standard .netrc format:

```
machine urs.earthdata.nasa.gov
  login <username>
  password <password>
```

If credentials are not found in environment variables and the package **getPass** is available, the user is prompted securely. Otherwise, an error is thrown.

The function also sets the NETRC environment variable for the current R process. Python libraries such as earthaccess inherit this setting when they are initialized.

Value

Character string. The normalized path to the .netrc file being used.

See Also

NASA Earthdata Login

---

ee *The pointer to the* earth-engine-api *python reticulate module*

Description

The pointer to the earth-engine-api python reticulate module

Usage

```
ee
```

See Also

https://developers.google.com/earth-engine/apidocs

`ee_build_AlphaEarth_embedding_terrain_stack`
Stack Alpha Earth embedding and terrain ancillary layers

Description

Creates a Google Earth Engine image stack that combines:

- Annual satellite embeddings from `GOOGLE/SATELLITE_EMBEDDING/V1/ANNUAL` (AlphaEarth).
- Terrain derivatives (elevation, slope, aspect) from USGS 3DEP or NASADEM as fallbacks.
- Optional longitude/latitude bands from `ee$Image$pixelLonLat()`.

The embedding collection is filtered to the specified year range and AOI, reduced using a pixel-wise median, and clipped to the AOI. Terrain data are reprojected to a target scale (default `30` m).

Geographic coverage limitation: The primary terrain sources (USGS 3DEP 1 m and 10 m) cover only the contiguous United States, Alaska, and Hawaii. For areas outside the US, the function falls back to NASADEM, which provides global coverage but at coarser resolution (~30 m). For best terrain accuracy, this function is recommended for use within the United States.

Authentication: This function requires an active Google Earth Engine session. Authenticate using `ICESat2VegR::ee_initialize()` before calling this function.

Usage

```
ee_build_AlphaEarth_embedding_terrain_stack(
  geom,
  start_year,
  end_year,
  mask_outside = TRUE,
  terrain_scale = 30,
  multiply_slope_aspect_by10 = TRUE,
  add_lonlat = TRUE
)
```

Arguments

`geom`	AOI geometry. Can be: • an `sf` or `sfc` object, • a `SpatVector` object, • or an Earth Engine geometry (Python object) already in geographic coordinates. The geometry is internally converted to an Earth Engine geometry via an internal helper.

`start_year`, `end_year`
: Integer (or coercible to integer). Inclusive year range used to filter the AlphaEarth annual embedding collection.

`mask_outside`
: Logical. If `TRUE` (default), pixels outside the AOI are masked out using a binary mask image clipped to `geom`.

`terrain_scale`
: Numeric. Target scale in meters used to reproject the terrain data. Default is `30`.

`multiply_slope_aspect_by10`
: Logical. If `TRUE` (default), slope and aspect are multiplied by 10 to preserve conventions used elsewhere in the package and avoid small floating-point values.

`add_lonlat`
: Logical. If `TRUE` (default), longitude and latitude bands named `lon` and `lat` are added, derived from `ee$Image$pixelLonLat()`.

Details

The terrain source is chosen in the following order:

1. USGS 3DEP 1 m (`USGS/3DEP/1m`), if available for the AOI.
2. USGS 3DEP 10 m (`USGS/3DEP/10m`).
3. NASADEM (`NASA/NASADEM_HGT/001`) as a global fallback.

All terrain layers are clipped to the AOI and reprojected to `EPSG:4326` with the requested `terrain_scale`.

Value

A Python `ee$Image` object containing:

- The median annual AlphaEarth embedding bands.
- `elevation`, `slope`, and `aspect` bands.
- Optional `lon` and `lat` bands.

Examples

```
## Not run:
  # Requires Google Earth Engine authentication
  # Note: USGS 3DEP terrain data covers the United States only.
  # Outside the US, NASADEM is used as a global fallback.
  Sys.setenv(EE_PROJECT = "your-ee-project-id")
  ICESat2VegR::ee_initialize()

  library(sf)
  aoi <- st_as_sfc(st_bbox(c(
    xmin = -82.4, xmax = -82.2,
    ymin =  29.6, ymax =  29.8
  ), crs = 4326))

  ee_geom <- ICESat2VegR::.as_ee_geom(aoi)

  # Build the embedding + terrain stack for 2025
  predictors <- ee_build_AlphaEarth_embedding_terrain_stack(
```

```
  geom       = ee_geom,
  start_year = 2025,
  end_year   = 2025
)

# Visualize RGB embedding bands using leaflet
predictors_rgb <- predictors$select(c("A00", "A20", "A40"))

centroid_lon <- mean(terra::ext(terra::vect(aoi))[c(1, 2)])
centroid_lat <- mean(terra::ext(terra::vect(aoi))[c(3, 4)])

leaflet::leaflet() |>
  ICESat2VegR::addEEImage(
    predictors_rgb,
    bands = c("A00", "A20", "A40"),
    group = "RGB Embedding",
    min   = -0.2,
    max   =  0.2
  ) |>
  leaflet::setView(
    lng  = centroid_lon,
    lat  = centroid_lat,
    zoom = 15
  ) |>
  leaflet::addLayersControl(
    overlayGroups = "RGB Embedding",
    options = leaflet::layersControlOptions(collapsed = FALSE)
  )

## End(Not run)
```

`ee_build_hls_s1c_terrain_stack` *Build HLS, Sentinel-1C and terrain ancillary stack in Earth Engine*

Description

Constructs a multi-source ancillary stack in Google Earth Engine composed of:

1. Optical features from the Harmonized Landsat and Sentinel-2 (HLS) product, including spectral bands, vegetation indices, and spectral unmixing fractions.
2. C-band SAR backscatter and radar-based indices from Sentinel-1C GRD.
3. Terrain variables (elevation, slope, aspect) from a NASA DEM.
4. Neighborhood statistics (mean, min, max, stdDev) computed over a 3x3 kernel for HLS and terrain bands.
5. GLCM texture metrics computed from scaled HLS reflectance bands.

The final stack contains 244 bands covering spectral, radar, terrain, neighborhood, and texture features.

Authentication: This function requires an active Google Earth Engine session. Authenticate using `ICESat2VegR::ee_initialize()` before calling this function.

Usage

```
ee_build_hls_s1c_terrain_stack(
  x,
  start_date,
  end_date,
  cloud_max = 10,
  buffer_m = 30,
  hls_collection_id = c("NASA/HLS/HLSS30/v002", "NASA/HLS/HLSL30/v002")
)
```

Arguments

`x`	Spatial input defining the area of interest. Can be: • a character path to a vector file readable by `vect` (e.g., SHP, GPKG), • a `SpatVector`, • a `SpatRaster` (its extent will be used), • an `sf` or `sfc` object, • a `SpatExtent` object, • a numeric vector of length 4: `c(xmin, xmax, ymin, ymax)`.
`start_date`	character. Start date of the temporal filter in `YYYY-MM-DD` format. **Required.**
`end_date`	character. End date of the temporal filter in `YYYY-MM-DD` format. **Required.**
`cloud_max`	numeric. Maximum allowed cloud coverage percentage for HLS scenes. Default is `10`.
`buffer_m`	numeric. Buffer distance in meters applied to the AOI bounding box before filtering and clipping. Default is `30`.
`hls_collection_id`	character vector. HLS collection IDs to try in order until one has data for the AOI and date range. Default tries Sentinel-2 HLS first, then Landsat HLS as fallback: `c("NASA/HLS/HLSS30/v002", "NASA/HLS/HLSL30/v002")`.

Details

The HLS collection is selected automatically from `hls_collection_id` by trying each ID in order and using the first one that contains scenes for the specified AOI and date range. By default, Sentinel-2 HLS (`NASA/HLS/HLSS30/v002`) is tried first, followed by Landsat HLS (`NASA/HLS/HLSL30/v002`).

Cloud masks are applied using the `Fmask` band before computing the median mosaic. Sentinel-1C is filtered by VV/VH polarization and IW instrument mode, sorted by acquisition time, and reduced using `ee$Reducer$firstNonNull()`.

Neighborhood statistics (mean, min, max, stdDev) are computed over a fixed 3x3 kernel for HLS and terrain bands. GLCM texture metrics are computed from scaled (x10000) HLS reflectance bands using a window size of 3.

Value

Returns an ee$Image object containing 244 bands:

- HLS reflectance bands and vegetation indices (blue, green, red, nir, swir1, swir2, ndvi, kndvi, evi, savi, msavi, f_soil, f_veg, f_water, sri, ndwi, gci, wdrvi, gvmi, cvi, cmr).
- Sentinel-1C SAR bands and radar indices (vv, vh, rvi, copol, copol2, copol3).
- Terrain bands (elevation, slope, aspect).
- Neighborhood statistics (mean, min, max, stdDev) for HLS and terrain bands.
- GLCM texture metrics for HLS reflectance bands.

Examples

```
## Not run:
  # Requires Google Earth Engine authentication
  Sys.setenv(EE_PROJECT = "your-ee-project-id")
  ICESat2VegR::ee_initialize()

  library(sf)
  library(terra)

  # AOI in Ocala National Forest, Florida
  aoi <- st_as_sfc(st_bbox(c(
    xmin = -81.8, xmax = -81.6,
    ymin =  29.1, ymax =  29.3
  ), crs = 4326))

  # Build the full ancillary stack (244 bands)
  stack <- ee_build_hls_s1c_terrain_stack(
    x          = aoi,
    start_date = "2024-03-01",
    end_date   = "2024-06-01",
    cloud_max  = 10
  )

  # Check available bands
  names(stack)

  # Compute AOI centroid for map view
  centroid_lon <- mean(terra::ext(terra::vect(aoi))[c(1, 2)])
  centroid_lat <- mean(terra::ext(terra::vect(aoi))[c(3, 4)])

  # Visualize true color RGB
  leaflet::leaflet() |>
    leaflet::addProviderTiles("Esri.WorldImagery") |>
    ICESat2VegR::addEEImage(
      stack$select(c("red", "green", "blue")),
      bands = c("red", "green", "blue"),
      group = "True Color",
      min   = 0,
      max   = 0.3
    ) |>
```

```
  leaflet::addControl(
    html     = "<b>True Color</b><br>red / green / blue",
    position = "bottomleft"
  ) |>
  leaflet::setView(lng = centroid_lon, lat = centroid_lat, zoom = 11) |>
  leaflet::addLayersControl(
    overlayGroups = "True Color",
    options = leaflet::layersControlOptions(collapsed = FALSE)
  )

# Visualize false color (nir, red, green)
leaflet::leaflet() |>
  leaflet::addProviderTiles("Esri.WorldImagery") |>
  ICESat2VegR::addEEImage(
    stack$select(c("nir", "red", "green")),
    bands = c("nir", "red", "green"),
    group = "False Color",
    min   = 0,
    max   = 0.3
  ) |>
  leaflet::addControl(
    html     = "<b>False Color</b><br>nir / red / green",
    position = "bottomleft"
  ) |>
  leaflet::setView(lng = centroid_lon, lat = centroid_lat, zoom = 11) |>
  leaflet::addLayersControl(
    overlayGroups = "False Color",
    options = leaflet::layersControlOptions(collapsed = FALSE)
  )

# Visualize NDVI
leaflet::leaflet() |>
  leaflet::addProviderTiles("Esri.WorldImagery") |>
  ICESat2VegR::addEEImage(
    stack$select("ndvi"),
    bands = "ndvi",
    group = "NDVI",
    min   = 0,
    max   = 0.8
  ) |>
  leaflet::addControl(
    html     = "<b>NDVI</b>",
    position = "bottomleft"
  ) |>
  leaflet::setView(lng = centroid_lon, lat = centroid_lat, zoom = 11) |>
  leaflet::addLayersControl(
    overlayGroups = "NDVI",
    options = leaflet::layersControlOptions(collapsed = FALSE)
  )

# Force Landsat HLS only
stack_landsat <- ee_build_hls_s1c_terrain_stack(
  x                = aoi,
```

```
  start_date        = "2024-03-01",
  end_date          = "2024-06-01",
  hls_collection_id = "NASA/HLS/HLSL30/v002"
)
names(stack_landsat)

## End(Not run)
```

`ee_check_task_status` *One-shot task status check*

Description

One-shot task status check

Usage

```
ee_check_task_status(task, quiet = TRUE)
```

Arguments

`task`	EE `Task`.
`quiet`	Logical; if `TRUE`, suppress printing.

Value

The task status (list).

`ee_initialize` *Initializes the Google Earth Engine API Initialize Earth Engine for this R session*

Description

Initializes the Google Earth Engine API Initialize Earth Engine for this R session

Usage

```
ee_initialize(
  project = Sys.getenv("EE_PROJECT", unset = NA),
  service_account = NULL,
  keyfile = NULL,
  quiet = FALSE,
  force_auth = FALSE
)
```

Arguments

project	Character. **GCP Project ID** (e.g., "ice-map-2025") or a numeric project **number**. If NULL/NA, falls back to Sys.getenv("EE_PROJECT").
service_account	Optional service account email (use with keyfile).
keyfile	Path to service-account JSON key (required if service_account is set).
quiet	Logical. Suppress messages.
force_auth	Logical. If TRUE, perform OAuth before Initialize().

Value

TRUE on success; FALSE otherwise (invisibly).

---

ee_number *Creates an* Earth Engine *server number*

Description

Creates an Earth Engine server number

Usage

```
ee_number(x)
```

Arguments

x	the number to convert to Earth Engine's number.

Value

The Earth Engine number

See Also

https://developers.google.com/earth-engine/apidocs/ee-number

ee_rect_to_sf *Convert an EE Rectangle to an sf polygon (EPSG:4326)*

Description

Convert an EE Rectangle to an sf polygon (EPSG:4326)

Usage

```
ee_rect_to_sf(aoi)
```

Arguments

aoi	An ee$Geometry$Rectangle.

Value

An sf polygon with CRS EPSG:4326.

extract *Given a geometry with point samples and images from Earth Engine retrieve the point geometry with values for the images*

Description

Given a geometry with point samples and images from Earth Engine retrieve the point geometry with values for the images

Usage

```
extract(stack, geom, scale)
```

Arguments

stack	A single image or a vector/list of images from Earth Engine.
geom	A geometry from terra::SpatVector read with terra::vect().
scale	The scale in meters for the extraction (image resolution).

Value

An ee.FeatureCollection with the properties extracted from the stack of images from ee.

`ext_to_ee` *Convert a terra extent or spatial object to an Earth Engine Rectangle*

Description

Converts a terra::ext, terra::SpatVector, or terra::SpatRaster into a Google Earth Engine geometry of type `ee$Geometry$Rectangle`. The resulting rectangle is always expressed in geographic coordinates (EPSG:4326), regardless of the input object's projection.

This function is mainly used internally to standardize spatial inputs before Earth Engine requests (e.g., filtering, clipping, or exporting data).

Usage

```
ext_to_ee(x)
```

Arguments

`x`	A spatial object of class terra::ext, terra::SpatVector, or terra::SpatRaster. Its extent is extracted and converted into an Earth Engine bounding box.

Value

A Python object representing `ee$Geometry$Rectangle`, suitable for use with Earth Engine Python API functions accessed via `reticulate`.

Examples

```
## Not run:
library(terra)

# Create an extent over Gainesville, FL
bb <- ext(c(-82.4, -82.2, 29.6, 29.8))

# Convert to EE geometry
aoi <- ext_to_ee(bb)

## End(Not run)
```

`fit_metrics` *Model fit metrics*

Description

Computes RMSE (absolute and relative), MAE (absolute and relative), bias (absolute and relative), Pearson correlation (r), and adjusted R^2 from a linear fit between predicted and observed values. Optionally draws a 1:1 plot with the regression line.

Usage

```
fit_metrics(
  observed,
  predicted,
  plotstat = FALSE,
  legend = "topleft",
  unit = "",
  ...
)
```

Arguments

`observed`	Numeric vector of observed values.
`predicted`	Numeric vector of predicted values (same length/order as `observed`).
`plotstat`	Logical; if TRUE, draws a 1:1 plot with the regression line. Default: FALSE.
`legend`	Character position for the plot legend (e.g., "topleft"). Default: "topleft".
`unit`	Character indicating the unit for RMSE/MAE/Bias (e.g., "Mg/ha"). Default: "".
`...`	Additional arguments passed to `graphics::plot()`.

Value

A data.frame with rows for `rmse`, `rmseR` (%), `mae`, `maeR` (%), `bias`, `biasR` (%), `r`, and `adj_r2`.

Examples

```
observed  <- c(178, 33, 60, 80, 104, 204, 146)
predicted <- c(184, 28.5, 55, 85, 105, 210, 155)
fit_metrics(observed, predicted,
            plotstat = TRUE, legend = "topleft", unit = "Mg/ha",
            xlab = "Observed AGBD (Mg/ha)", ylab = "Predicted AGBD (Mg/ha)", pch = 16)
```

`fit_model` *Fit a Random Forest with optional resampling, tuning, and progress bars*

Description

`fit_model()` trains a Random Forest regression model and optionally evaluates it via LOOCV, K-fold CV, a train/test split, or bootstrap OOB estimation. It can also tune core RF hyperparameters and shows progress bars for long-running loops.

Usage

```
fit_model(
  x,
  y,
  rf_args = list(ntree = 500, mtry = NULL, nodesize = 5, sampsize = NULL),
  test = list(method = "none", k = 5, test_size = 0.3, seed = NULL, folds = NULL,
    iterations = 200, correction = FALSE),
 tune = list(enable = FALSE, search = "grid", grid = NULL, n_random = 20, seed = NULL),
  verbose = TRUE,
  list_test_models = TRUE
)
```

Arguments

`x`	A data.frame of predictors (rows = samples, cols = features).
`y`	A numeric vector of responses, `length(y)` == `nrow(x)`.
`rf_args`	A named list of base Random Forest arguments used for all fits (full-data fit and each resample). Elements: • `ntree` (integer, default `500`): number of trees. • `mtry` (integer or `NULL`): number of variables sampled at each split. If `NULL`, a safe default `floor(sqrt(p))` is used. • `nodesize` (integer, default `5`): minimum terminal node size. • `sampsize` (integer, fraction in `(0,1]`, or `NULL`): sample size per tree. If a single fraction, it is multiplied by the current training size and rounded. If `NULL`, the package default is used.
`test`	A named list configuring evaluation: • `method` (character): one of `none`, `loocv`, `k-fold` (also accepts `kfold`/`k fold`), `split`, or `bootstrap`. • `k` (integer, default `5`): number of folds for K-fold CV. • `folds` (list or `NULL`): custom index list of test-fold indices. If supplied, overrides `k`. • `test_size` (numeric in `(0,1)`, default `0.3`): test fraction for train/test split. • `iterations` (integer, default `200`): number of bootstrap resamples.

- `correction` (logical, default `FALSE`): if `TRUE`, apply the .632 correction to the RMSE for the bootstrap estimate.
- `seed` (integer or `NULL`): RNG seed for reproducibility (folds, split, random tuning).

`tune` A named list configuring hyperparameter search on the *training* subset for each fit:

- `enable` (logical, default `FALSE`): turn tuning on/off.
- `search` (character, default `grid`): `grid` or `random`.
- `grid` (data.frame or `NULL`): explicit grid with columns `mtry`, `ntree`, `nodesize`, `sampsize`. If `NULL`, a sensible default grid is generated from `p = ncol(x)`.
- `n_random` (integer, default `20`): number of random draws from the grid when `search = "random"`.
- `seed` (integer or `NULL`): RNG seed for the tuning subset draw order.

`verbose` Logical (default `TRUE`): show progress bars/messages for tuning, LOOCV, K-fold, and bootstrap loops. Set `FALSE` to silence.

`list_test_models` Logical (default `TRUE`): if `TRUE`, save the *per-resample* fitted model objects as a list in `models_test`. This can be large, especially for `bootstrap` with many iterations.

Details

Tuning minimizes OOB MSE for each candidate configuration on the current training subset and then refits the model using the best settings. When `test$method = "bootstrap"` and `correction = TRUE`, the .632 corrected RMSE is reported while keeping other OOB statistics unchanged.

Value

A list with:

method `rf`

rf_args Final RF arguments used for the full-data fit (after tuning).

tune_table (data.frame or `NULL`) tuning results sorted by OOB MSE.

test Echo of `test` configuration (with resolved values).

model `randomForest` object fit on the full dataset (or train subset for `split`).

fitted_full Numeric vector of in-sample predictions from the full-data fit.

stats_train data.frame of training statistics (RMSE, Bias, %RMSE, %Bias, r, r2).

stats_test (data.frame or `NULL`) test-set or OOF statistics depending on method.

loocv_pred (numeric) LOOCV predictions (for `method = "loocv"`).

cv_pred (numeric) K-fold OOF predictions (for `method = "k-fold"`).

train_index,test_index (integer) indices for `method = "split"`.

pred_train,pred_test (numeric) predictions for train/test in `split`.

oob_pred (numeric) OOB mean prediction per observation in `bootstrap`.

models_test (list or `NULL`) models fitted per resample/fold/iteration when `list_test_models=TRUE`.

Workflow

1. Optionally tune RF hyperparameters on the *current training subset* (or on full data when `test$method = "none"`).
2. Always fit a model on the full dataset (`model` and `fitted_full`).
3. If a testing method is requested, refit on resampled training folds and compute out-of-fold predictions to summarize generalization performance.

Progress Bars

Uses `utils::txtProgressBar()`; disable with `verbose = FALSE`. The internal helper is lightweight and has no external dependencies.

See Also

`randomForest::randomForest()`, `stats::lm()`, `stats::cor()`

Examples

```
set.seed(42)
n <- 200
x <- data.frame(NDVI = runif(n, 0.2, 0.9),
                EVI  = runif(n, 0.1, 0.8),
                NBR  = runif(n, -0.5, 0.9),
                SLP  = runif(n, 0, 30))
y <- with(x, 5 + 20*NDVI + 10*EVI^1.5 - 0.05*SLP + rnorm(n, 0, 2))

# Full-data fit (no resampling)
fit_none <- fit_model(x, y, rf_args = list(ntree = 400, mtry = 2))
fit_none$stats_train

# LOOCV

fit_loocv <- fit_model(x, y, test = list(method = "loocv"))
fit_loocv$stats_test

# 5-fold CV
fit_k_fold <- fit_model(x, y, test = list(method = "k-fold", k = 5, seed = 42))
fit_k_fold$stats_test

# Train/Test split
fit_split <- fit_model(x, y, test = list(method = "split", test_size = 0.25, seed = 42))
fit_split$stats_test

# Bootstrap (with tuning and .632 correction)
ntree <- 400
iterations <- 300

fit_boot <- fit_model(
  x, y,
  rf_args = list(ntree = ntree),
  test    = list(method = "bootstrap", iterations = iterations, correction = TRUE),
```

```
  tune    = list(enable = TRUE, search = "random", n_random = 12)
)

# K-fold CV with saved models
fit_k <- fit_model(x, y, test = list(method = "k-fold", k = 5, seed = 42), list_test_models = TRUE)
length(fit_k$models_test)  # one model per fold
```

`formulaCalculate` *Calculate raster values based on a formula*

Description

Calculate raster values based on a formula

Usage

```
formulaCalculate(formula, data, updateBand)
```

Arguments

`formula`	Formula. A formula to apply to the RasterBands from `data`
`data`	List. A named list with the used variables in the textual formula
`updateBand`	GDALRasterBand. The GDALRasterBand which will be updated with the calculated values.

Value

Nothing, it just updates the band of interest.

Examples

```
# Parameters
raster_path <- file.path(tempdir(), "output.tif")
ul_lat <- -15
ul_lon <- -45
lr_lat <- -25
lr_lon <- -35
res <- c(0.01, -0.01)
datatype <- GDALDataType$GDT_Int32
nbands <- 2
projstring <- "EPSG:4326"
nodata <- -1
co <- c("TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512", "COMPRESSION=LZW")

# Create a new raster dataset
ds <- createDataset(
  raster_path = raster_path,
  nbands = nbands,
```

```
  datatype = datatype,
  projstring = projstring,
  lr_lat = lr_lat,
  ul_lat = ul_lat,
  ul_lon = ul_lon,
  lr_lon = lr_lon,
  res = res,
  nodata = nodata,
  co = co
)

# Get the GDALRasterBand for ds
band <- ds[[1]]

# The updateBand can be the same
# using a different one just for testing
updateBand <- ds[[2]]

# Set some dummy values
band[[0, 0]] <- 1:(512 * 512)

# Calculate the double - 10
formulaCalculate(
  formula = ~ x * 2 - 10,
  data = list(x = band),
  updateBand = updateBand
)

ds$Close()
```

`GDALDataset` *R6 Class GDALDataset wrapping*

Description

Wrapping class for GDALDataset C++ API exporting GetRasterBand, GetRasterXSize, GetRasterYSize

Methods

Public methods:

- `GDALDataset$new()`
- `GDALDataset$GetRasterBand()`
- `GDALDataset$GetRasterXSize()`
- `GDALDataset$GetRasterYSize()`
- `GDALDataset$Close()`
- `GDALDataset$clone()`

`GDALDataset$new()`: Create a new raster file based on specified data. It will output a *.tif file.

Usage:

```
GDALDataset$new(ds)
```

Arguments:

`ds` GDALDatasetR pointer. Should not be used

`datatype` GDALDataType. The GDALDataType to use for the raster, use (GDALDataType$) to find the options. Default GDALDataType$GDT_Float64

Returns: An object from GDALDataset R6 class.

`GDALDataset$GetRasterBand()`: Function to retrieve the GDALRasterBand R6 Object.

Usage:

```
GDALDataset$GetRasterBand(x)
```

Arguments:

`x` Integer. The band index, starting from 1 to number of bands.

Returns: An object of GDALRasterBand R6 class.

`GDALDataset$GetRasterXSize()`: Get the width for the raster

Usage:

```
GDALDataset$GetRasterXSize()
```

Returns: An integer indicating the raster width

`GDALDataset$GetRasterYSize()`: Get the height for the raster

Usage:

```
GDALDataset$GetRasterYSize()
```

Returns: An integer indicating the raster height

`GDALDataset$Close()`: Closes the GDALDataset

Usage:

```
GDALDataset$Close()
```

Returns: An integer indicating the raster width

`GDALDataset$clone()`: The objects of this class are cloneable with this method.

Usage:

```
GDALDataset$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

GDALDataType *List of datatypes supported by the GDALDataset R6 class*

Description

List of datatypes supported by the GDALDataset R6 class

Usage

```
GDALDataType
```

GDALOpen *Open GDAL raster*

Description

Function to open GDAL Dataset

Usage

```
GDALOpen(filename, readonly = TRUE)
```

Arguments

filename	Character. The path to a GDAL dataset.
readonly	Logical. Flag to open a read only GDALDataset with GA_ReadOnly or GA_Update. Default TRUE.

Value

An R6 object of GDALDataset class.

Examples

```
# Create a small raster to open
ds_path <- tempfile(fileext = ".tif")
ds <- createDataset(
  raster_path = ds_path,
  nbands = 1,
  datatype = GDALDataType$GDT_Int32,
  projstring = "EPSG:4326",
  ul_lat = -15, ul_lon = -45, lr_lat = -25, lr_lon = -35,
  res = c(0.01, -0.01),
  nodata = -1
)
ds$Close()
```

```
# Reopen the raster read-only
ds2 <- GDALOpen(ds_path)
ds2$Close()
```

GDALRasterBand *R6 Class GDALRasterBand wrapping*

Description

Wrapping class for GDALRasterBand C++ API exporting GetBlockXSize, GetBlockYSize, ReadBlock, WriteBlock for better IO speed.

Methods

Public methods:

- GDALRasterBand$new()
- GDALRasterBand$ReadBlock()
- GDALRasterBand$WriteBlock()
- GDALRasterBand$GetBlockXSize()
- GDALRasterBand$GetBlockYSize()
- GDALRasterBand$GetXSize()
- GDALRasterBand$CalculateStatistics()
- GDALRasterBand$GetYSize()
- GDALRasterBand$GetNoDataValue()
- GDALRasterBand$GetRasterDataType()
- GDALRasterBand$clone()

`GDALRasterBand$new()`: Creates a new GDALRasterBand

Usage:

```
GDALRasterBand$new(band)
```

Arguments:

`band` The C++ pointer to the GDALRasterBandR object.

`datatype` The GDALDataType for this band

Returns: An object of GDALRasterBand R6 class

`GDALRasterBand$ReadBlock()`: Efficiently reads a raster block

Usage:

```
GDALRasterBand$ReadBlock(iXBlock, iYBlock)
```

Arguments:

`iXBlock` Integer. The i-th column block to access. The `iXBlock` will be offset $BLOCKXSIZE \times iXBlock$ from the origin.

`iYBlock` Integer. The i-th row block to access. The `iYBlock` will be offset $BLOCKYSIZE \times iYBlock$ from the origin.

Details: The returned Vector will be single dimensional with the length $BLOCKXSIZE \times BLOCKYSIZE$. If you use matrix(, ncol=BLOCKXSIZE) the matrix returned will actually be transposed. You should either transpose it or you can calculate the indices using $y \cdot xsize + x$

Returns: RawVector for GDALDataType$GDT_Byte, IntegerVector for int types and NumericVector for floating point types.

`GDALRasterBand$WriteBlock()`: Efficiently writes a raster block

Usage:

```
GDALRasterBand$WriteBlock(iXBlock, iYBlock, buffer)
```

Arguments:

`iXBlock` Integer. The i-th column block to write. The `iXBlock` will be offset $BLOCKXSIZE \times iXBlock$ from the origin.

`iYBlock` Integer. The i-th row block to write. The `iYBlock` will be offset $BLOCKYSIZE \times iYBlock$ from the origin.

`buffer` RawVector/IntegerVector/NumericVector depending on the GDALDataType. This should be a 1D vector with size equal to raster $BLOCKXSIZE \times BLOCKYSIZE$.

Details: The returned Vector will be single dimensional with the length $BLOCKXSIZE \times BLOCKYSIZE$. If you use matrix(, ncol=BLOCKXSIZE) the matrix returned will actually be transposed. You should either transpose it or you can calculate the indices using $y \cdot xsize + x$.

Returns: Nothing

`GDALRasterBand$GetBlockXSize()`: Get the block width

Usage:

```
GDALRasterBand$GetBlockXSize()
```

Returns: An integer indicating block width

`GDALRasterBand$GetBlockYSize()`: Get the block height

Usage:

```
GDALRasterBand$GetBlockYSize()
```

Returns: An integer indicating block height

`GDALRasterBand$GetXSize()`: Get the band width

Usage:

```
GDALRasterBand$GetXSize()
```

Returns: An integer indicating band width

`GDALRasterBand$CalculateStatistics()`: Calculate statistics for the `GDALRasterBand`

Usage:

```
GDALRasterBand$CalculateStatistics()
```

Returns: nothing

`GDALRasterBand$GetYSize()`: Get the band height

Usage:

`GDALRasterBand$GetYSize()`

Returns: An integer indicating band height

`GDALRasterBand$GetNoDataValue()`: Get band no data value

Usage:

`GDALRasterBand$GetNoDataValue()`

Returns: Numeric indicating no data value

`GDALRasterBand$GetRasterDataType()`: Get band datatype

Usage:

`GDALRasterBand$GetRasterDataType()`

Returns: Numeric indicating the datatype

`GDALRasterBand$clone()`: The objects of this class are cloneable with this method.

Usage:

```
GDALRasterBand$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

This constructor must not be called at all, this is automatically called from GDALDataset$GetRasterBand function.

`geomSampling` *Get observations sampled within polygon features*

Description

Get observations sampled within polygon features

Usage

```
geomSampling(size, geom, split_id = NULL, chainSampling = NULL)
```

Arguments

`size`	numeric. The sample size per polygon feature. Either an integer or a value between `(0, 1)` for a percentage.
`geom`	A `SpatVector` polygon object opened with `vect`.
`split_id`	character. The attribute name in `geom` to use as the grouping factor for sampling.
`chainSampling`	optional. Chain with another sampling method.

Value

A icesat2_sampling_method-class object for use in sample.

See Also

sample, randomSampling

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")
atl08_h5   <- ATL08_read(atl08_path = atl08_path)
atl08_dt   <- ATL08_seg_attributes_dt(atl08_h5)

# Load polygon shapefile
polygon_filepath <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)
polygon <- terra::vect(polygon_filepath)

# Sample 2 observations per polygon feature
set.seed(1)
sampled <- ICESat2VegR::sample(
  atl08_dt,
  method = geomSampling(size = 2, geom = polygon, split_id = "id")
)
head(sampled)

close(atl08_h5)
}
```

---

getTileUrl *Retrieve Google Earth Engine's tile url for an Image or ImageCollection*

Description

Retrieve Google Earth Engine's tile url for an Image or ImageCollection

Usage

```
getTileUrl(img)
```

Arguments

img	The Image or ImageCollection to retrieve the URL

Value

The url for the tile service.

`get_catalog_id` *Retrieve the Google Earth Engine image catalog id*

Description

Retrieve the Google Earth Engine image catalog id

Usage

```
get_catalog_id(id)
```

Arguments

`id`	character. The id retrieved from the data.table resulting from `search_datasets()`.

Value

The catalog id to open within Google Earth Engine.

`glcmTexture` *Maps to* `ee.Image.glcmTexture`

Description

Computes texture metrics from the Gray Level Co-occurrence Matrix around each pixel of every band. The GLCM is a tabulation of how often different combinations of pixel brightness values (grey levels) occur in an image. It counts the number of times a pixel of value X lies next to a pixel of value Y, in a particular direction and distance. and then derives statistics from this tabulation.

Usage

```
glcmTexture(x, size = 1, kernel = NULL, average = TRUE)
```

Arguments

`x`	The input image to calculate the texture on.
`size`	integer, default 1. The size of the neighborhood to include in each GLCM.
`kernel`	default NULL. A kernel specifying the x and y offsets over which to compute the GLCMs. A GLCM is computed for each pixel in the kernel that is non-zero, except the center pixel and as long as a GLCM hasn't already been computed for the same direction and distance. For example, if either or both of the east and west pixels are set, only 1 (horizontal) GLCM is computed. Kernels are scanned from left to right and top to bottom. The default is a 3x3 square, resulting in 4 GLCMs with the offsets (-1, -1), (0, -1), (1, -1) and (-1, 0).
`average`	logical, default TRUE. If true, the directional bands for each metric are averaged.

Details

This implementation computes the 14 GLCM metrics proposed by Haralick, and 4 additional metrics from Conners. Inputs are required to be integer valued.

The output consists of 18 bands per input band if directional averaging is on and 18 bands per directional pair in the kernel, if not:

1. ASM: f1, Angular Second Moment; measures the number of repeated pairs
2. CONTRAST: f2, Contrast; measures the local contrast of an image
3. CORR: f3, Correlation; measures the correlation between pairs of pixels
4. VAR: f4, Variance; measures how spread out the distribution of gray-levels is
5. IDM: f5, Inverse Difference Moment; measures the homogeneity
6. SAVG: f6, Sum Average
7. SVAR: f7, Sum Variance
8. SENT: f8, Sum Entropy
9. ENT: f9, Entropy. Measures the randomness of a gray-level distribution
10. DVAR: f10, Difference variance
11. DENT: f11, Difference entropy
12. IMCORR1: f12, Information Measure of Corr. 1
13. IMCORR2: f13, Information Measure of Corr. 2
14. MAXCORR: f14, Max Corr. Coefficient. (not computed)
15. DISS: Dissimilarity
16. INERTIA: Inertia
17. SHADE: Cluster Shade
18. PROM: Cluster prominence

Value

Another Earth Engine image with bands described on details section.

See Also

More information can be found in the two papers: Haralick et. al, 'Textural Features for Image Classification', http://doi.org/10.1109/TSMC.1973.4309314 and Conners, et al, Segmentation of a high-resolution urban scene using texture operators', http://doi.org/10.1016/0734-189X(84)90197-X.

https://developers.google.com/earth-engine/apidocs/ee-image-glcmtexture

gridSampling *Get samples stratified by grid cells of specified size*

Description

Get samples stratified by grid cells of specified size

Usage

```
gridSampling(size, grid_size, chainSampling = NULL)
```

Arguments

size	numeric. The sample size per grid cell. Either an integer or a value between (0, 1) for a percentage.
grid_size	numeric. The grid cell size in decimal degrees.
chainSampling	optional. Chain with another sampling method such as randomSampling, spacedSampling, or stratifiedSampling.

Value

A icesat2_sampling_method-class object for use in sample.

See Also

sample, randomSampling

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")
atl08_h5   <- ATL08_read(atl08_path = atl08_path)
atl08_dt   <- ATL08_seg_attributes_dt(atl08_h5)

# Sample 2 observations per 0.01 degree grid cell
set.seed(1)
sampled <- ICESat2VegR::sample(
  atl08_dt,
  method = gridSampling(size = 2, grid_size = 0.01)
)
head(sampled)

# Chain grid sampling with random sampling within each cell
set.seed(1)
sampled_chain <- ICESat2VegR::sample(
  atl08_dt,
  method = gridSampling(
    size          = 2,
    grid_size     = 0.01,
```

```
    chainSampling = randomSampling(2)
  )
)
head(sampled_chain)

close(atl08_h5)
}
```

`icesat2.atl03atl08_dt-class` *Class for joined ATL03 and ATL08 attributes*

Description

Class for joined ATL03 and ATL08 attributes

See Also

`data.table` in the `data.table` package and https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

---

`icesat2.atl03_atl08_seg_dt-class` *Class that represent custom segments created from ATL03 and ATL08 joined data*

Description

Class that represent custom segments created from ATL03 and ATL08 joined data

Details

This class is actually just a wrap around the `data.table`, but it indicates the output from `ATL03_ATL08_segment_create()`, which means the dataset will contain the needed structure for computing value for the computing the stats with `ATL03_ATL08_compute_seg_attributes_dt_segStat()`

`icesat2.atl03_dt-class` *Class for ATL03 attributes*

Description

Class for ATL03 attributes

See Also

`data.table` in the `data.table` package and https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

icesat2.atl03_h5-class

Class for ICESat-2 ATL03

Description

Class for ICESat-2 ATL03

Slots

h5 Object of class H5File from hdf5r package containing the ICESat-2 Global Geolocated Photon Data (ATL03)

See Also

H5File in the hdf5r package and https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

icesat2.atl03_seg_dt-class

Class for ATL03 segment attributes

Description

Class for ATL03 segment attributes

See Also

data.table in the data.table package and https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

icesat2.atl08_dt-class

Class for ATL08 attributes

Description

Class for ATL08 attributes

See Also

data.table in the data.table package and https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

icesat2.atl08_h5-class *Class for ICESat-2 ATL08*

Description

Class for ICESat-2 ATL08

Slots

`h5` Object of class `H5File` from `hdf5r` package containing the ICESat-2 Land and Vegetation Along-Track Products (ATL08)

See Also

`H5File` in the `hdf5r` package and https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

icesat2.h5-class *Base class for all ICESat2VegR package's H5 files for generic functions that can be run on any H5*

Description

Base class for all ICESat2VegR package's H5 files for generic functions that can be run on any H5

ICESat2.h5ds_cloud *Class representing dataset opened from the cloud using h5py*

Description

This class should not be instanced by the user, instead it is automatically handled when the user opens a dataset using `[[]]` operator from a h5 file.

Public fields

`ds` Dataset pointer

`dims` Dimensions of the dataset

`chunk_dims` Chunk dimensions for the dataset

Methods

Public methods:

- ICESat2.h5ds_cloud$new()
- ICESat2.h5ds_cloud$get_type()
- ICESat2.h5ds_cloud$get_fill_value()
- ICESat2.h5ds_cloud$get_creation_property_list()
- ICESat2.h5ds_cloud$clone()

`ICESat2.h5ds_cloud$new()`: Constructor using a dataset pointer

Usage:

```
ICESat2.h5ds_cloud$new(ds)
```

Arguments:

`ds` Dataset pointer

`ICESat2.h5ds_cloud$get_type()`: Get the type of data for this dataset

Usage:

```
ICESat2.h5ds_cloud$get_type()
```

`ICESat2.h5ds_cloud$get_fill_value()`: Get the fill_value used by the dataset

Usage:

```
ICESat2.h5ds_cloud$get_fill_value()
```

`ICESat2.h5ds_cloud$get_creation_property_list()`: Get creation property list for the dataset (plist)

Usage:

```
ICESat2.h5ds_cloud$get_creation_property_list()
```

`ICESat2.h5ds_cloud$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ICESat2.h5ds_cloud$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

ICESat2.h5ds_local *Class representing dataset opened from locally using hdf5r*

Description

This class should not be instanced by the user, instead it is automatically handled when the user opens a dataset using [[]] operator from a h5 file.

Public fields

ds Dataset pointer

dims Dimensions of the dataset

chunk_dims Chunk dimensions for the dataset

Methods

Public methods:

- ICESat2.h5ds_local$new()
- ICESat2.h5ds_local$get_type()
- ICESat2.h5ds_local$get_fill_value()
- ICESat2.h5ds_local$get_creation_property_list()
- ICESat2.h5ds_local$clone()

ICESat2.h5ds_local$new(): Constructor using a dataset pointer

Usage:

```
ICESat2.h5ds_local$new(ds)
```

Arguments:

ds Dataset pointer

ICESat2.h5ds_local$get_type(): Get the type of data for this dataset

Usage:

```
ICESat2.h5ds_local$get_type()
```

ICESat2.h5ds_local$get_fill_value(): Get the fill_value used by the dataset

Usage:

```
ICESat2.h5ds_local$get_fill_value()
```

ICESat2.h5ds_local$get_creation_property_list(): Get creation property list for the dataset (plist)

Usage:

```
ICESat2.h5ds_local$get_creation_property_list()
```

ICESat2.h5ds_local$clone(): The objects of this class are cloneable with this method.

Usage:

```
ICESat2.h5ds_local$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`ICESat2.h5_cloud` *The class representing the h5 file opened from the cloud for cloud computing*

Description

Besides representing h5 files, it is also used to represent groups opened with `[[]]`.

The variants `_cloud` and `_local` allows all the other functions to use generic calls using the same interface, with each class implementation is provided accordingly.

The regular usage does not require the user to work with those classes as most other provided functions will actually give access to the most common necessities for working with ICESat-2 data.

Public fields

`h5` A pointer to `h5py` in case the user wants to access features not implemented yet

`beams` The `character` vector of beams available for the granule.

`strong_beams` The `character` vector of strong beams calculated using orbit_info

`weak_beams` The `character` vector of weak beams calculated using orbit_info

Methods

Public methods:

- `ICESat2.h5_cloud$new()`
- `ICESat2.h5_cloud$ls()`
- `ICESat2.h5_cloud$ls_groups()`
- `ICESat2.h5_cloud$ls_attrs()`
- `ICESat2.h5_cloud$dt_datasets()`
- `ICESat2.h5_cloud$exists()`
- `ICESat2.h5_cloud$attr()`
- `ICESat2.h5_cloud$close_all()`
- `ICESat2.h5_cloud$print()`
- `ICESat2.h5_cloud$clone()`

`ICESat2.h5_cloud$new()`: Direct initialization should not be used, it is handled by `ATL0X_read()`

Usage:

```
ICESat2.h5_cloud$new(h5)
```

Arguments:

`h5` the result of the `ATLAS_dataFinder()` with `cloud_computing = TRUE`

Returns: The class object

`ICESat2.h5_cloud$ls()`: Lists the groups and datasets that are within current group

Usage:

`ICESat2.h5_cloud$ls()`

Returns: List the groups and datasets within the current path

`ICESat2.h5_cloud$ls_groups()`: Lists all grouops recursively

Usage:

```
ICESat2.h5_cloud$ls_groups(recursive = FALSE)
```

Arguments:

`recursive` `logical`, default FALSE. If TRUE it will list groups recursively and return the full path.

Returns: The `character` representing

`ICESat2.h5_cloud$ls_attrs()`: Lists the available attributes

Usage:

```
ICESat2.h5_cloud$ls_attrs()
```

Returns: `character` vector of attributes available

`ICESat2.h5_cloud$dt_datasets()`: Get datasets as data.table with columns (name, dataset.dims, rank)

Usage:

```
ICESat2.h5_cloud$dt_datasets(recursive = FALSE)
```

Arguments:

`recursive` `logical`, default FALSE. If TRUE recursively searches and returns the full path.

Returns: A `data.table::data.table` with the columns (name, dataset.dims, rank)

`ICESat2.h5_cloud$exists()`: Checks if a supplied group/dataset exist within the H5

Usage:

```
ICESat2.h5_cloud$exists(path)
```

Arguments:

`path` `character` with the path to test

Returns: `logical` TRUE or FALSE if it exists or not

`ICESat2.h5_cloud$attr()`: Read an attribute from h5

Usage:

```
ICESat2.h5_cloud$attr(attribute)
```

Arguments:

`attribute` `character` the address of the attribute to open

`ICESat2.h5_cloud$close_all()`: Safely closes the h5 file pointer

Usage:

```
ICESat2.h5_cloud$close_all()
```

Returns: Nothing, just closes the file

`ICESat2.h5_cloud$print()`: Prints the data in a friendly manner

Usage:

```
ICESat2.h5_cloud$print(...)
```

Arguments:

`...` Added for compatibility purposes with generic signature

Returns: Outputs information about object

`ICESat2.h5_cloud$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ICESat2.h5_cloud$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ICESat2.h5_local` *The class representing the h5 file opened from local files*

Description

The variants `_cloud` and `_local` allows all the other functions to use generic calls using the same interface, with each class implementation is provided accordingly.

The regular usage does not require the user to work with those classes as most other provided functions will actually give access to the most common necessities for working with ICESat-2 data.

Public fields

`h5` A pointer to `hdf5r::H5File` in case the user wants to access features not implemented yet

`beams` The `character` vector of beams available for the granule.

`strong_beams` The `character` vector of strong beams calculated using orbit_info

`weak_beams` The `character` vector of weak beams calculated using orbit_info

`isOpen` A flag to indicate if the file pointer has already been closed

Methods

Public methods:

- `ICESat2.h5_local$new()`
- `ICESat2.h5_local$ls()`
- `ICESat2.h5_local$ls_groups()`
- `ICESat2.h5_local$ls_attrs()`
- `ICESat2.h5_local$dt_datasets()`
- `ICESat2.h5_local$exists()`
- `ICESat2.h5_local$attr()`
- `ICESat2.h5_local$close_all()`

- ICESat2.h5_local$print()
- ICESat2.h5_local$clone()

`ICESat2.h5_local$new()`: Direct initialization should not be used, it is handled by `ATL0X_read()`

Usage:

```
ICESat2.h5_local$new(h5)
```

Arguments:

`h5` the result of the `ATLAS_dataFinder()`

Returns: The class object

`ICESat2.h5_local$ls()`: Lists the groups and datasets that are within current group

Usage:

```
ICESat2.h5_local$ls()
```

Returns: List the groups and datasets within the current path

`ICESat2.h5_local$ls_groups()`: Lists all grouops recursively

Usage:

```
ICESat2.h5_local$ls_groups(recursive = FALSE)
```

Arguments:

`recursive` `logical`, default FALSE. If TRUE it will list groups recursively and return the full path.

Returns: The `character` representing

`ICESat2.h5_local$ls_attrs()`: Lists the available attributes

Usage:

```
ICESat2.h5_local$ls_attrs()
```

Returns: `character` vector of attributes available

`ICESat2.h5_local$dt_datasets()`: Get datasets as data.table with columns (name, dataset.dims, rank)

Usage:

```
ICESat2.h5_local$dt_datasets(recursive = FALSE)
```

Arguments:

`recursive` `logical`, default FALSE. If TRUE recursively searches and returns the full path.

Returns: A `data.table::data.table` with the columns (name, dataset.dims, rank)

`ICESat2.h5_local$exists()`: Checks if a supplied group/dataset exist within the H5

Usage:

```
ICESat2.h5_local$exists(path)
```

Arguments:

`path` `character` with the path to test

Returns: `logical` TRUE or FALSE if it exists or not

`ICESat2.h5_local$attr()`: Read an attribute from h5

Usage:

```
ICESat2.h5_local$attr(attribute)
```

Arguments:

`attribute` character the address of the attribute to open

`ICESat2.h5_local$close_all()`: Safely closes the h5 file pointer

Usage:

```
ICESat2.h5_local$close_all(silent = TRUE)
```

Arguments:

`silent` logical, default TRUE. Will cast warning messages if silent = FALSE.

Returns: Nothing, just closes the file

`ICESat2.h5_local$print()`: Prints the data in a friendly manner

Usage:

```
ICESat2.h5_local$print(...)
```

Arguments:

`...` Added for compatibility purposes with generic signature

Returns: Outputs information about object

`ICESat2.h5_local$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ICESat2.h5_local$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

ICESat2VegR_configure *Configure Python environment for ICESat2VegR cloud features*

Description

This function sets up a robust, persistent Python environment for the cloud-related features of **ICESat2VegR**, including:

Usage

```
ICESat2VegR_configure(
  envname = "icesat2-env",
  py_ver = "3.10",
  prefer_conda = TRUE,
  force_conda_forge = TRUE,
  allow_session_installs = TRUE,
  manage_aiobotocore = TRUE,
  auto_restart = TRUE,
  verbose = TRUE,
  quick_probe = c("h5py", "earthaccess", "ee")
)
```

Arguments

envname	Character. Name of the conda/virtualenv environment to use or create. Default is "icesat2-env". If this is the default value and the environment variable CONDA_DEFAULT_ENV is set, that value is used instead (useful when running inside an existing conda environment).
py_ver	Character. Python version to request when creating a new env (default "3.10").
prefer_conda	Logical. If TRUE (default), use conda/Miniconda. If FALSE, a virtualenv is used instead.
force_conda_forge	Logical. If TRUE (default), configure conda to use conda-forge with strict channel priority (recommended).
allow_session_installs	Logical. If TRUE (default) and Python is already initialized in the current R session, attempt session-only installs first via reticulate::py_require() and, if needed, a pip call from the active interpreter. If that fails, a persistent env is prepared instead.
manage_aiobotocore	Logical. If TRUE (default), attempt to detect and resolve common aiobotocore/botocore/boto3 version conflicts by pinning compatible versions and optionally uninstalling aiobotocore (which is not required for ICESat2VegR).
auto_restart	Logical. If TRUE (default), attempts to restart the R session in RStudio via rstudioapi::restartSession() after preparing the conda environment, so that the new Python binding is cleanly established.
verbose	Logical. If TRUE (default), print progress messages.
quick_probe	Character vector of Python import names to test for a fast early exit (default: c("h5py", "earthaccess", "ee")).

Details

- Downloading ICESat-2 data via earthaccess
- Interfacing with Google Earth Engine via earthengine-api
- Handling S3 access via boto3/botocore

The function:

- Detects whether Python is already initialized in the R session.
- Performs a fast "quick probe" to see if required modules are present (h5py, earthaccess, ee by default).
- If everything is already available, it returns immediately.
- Otherwise, it:
  - Ensures Miniconda is installed (if prefer_conda = TRUE).
  - Creates or reuses a conda environment (default: "icesat2-env").
  - Installs required Python modules using conda-forge (and pip fallback).
  - Optionally tries in-session installs if Python is already initialized.

- Optionally resolves common `aiobotocore`/`botocore` conflicts.
- Optionally restarts the R session in RStudio to bind the new env.

Typical usage is simply:

```
ICESat2VegR_configure()
```

You usually only need to run this:

- Once per machine, or
- After major Python / Miniconda changes, or
- If the package reports missing Python modules.

Value

Logical `TRUE` on success, `FALSE` otherwise.

Checked/installed Python modules

The function checks for the following import names and installs the corresponding packages when missing:

Import name	Install name
numpy	numpy
h5py	h5py
packaging	packaging
ee	earthengine-api
earthaccess	earthaccess
googleapiclient	google-api-python-client
boto3	boto3
botocore	botocore
s3transfer	s3transfer

Examples

```
## Not run:
  # Basic configuration using defaults
  ICESat2VegR_configure()

  # Use existing conda env named "icesat2"
  ICESat2VegR_configure(envname = "icesat2")

## End(Not run)
```

length,ee.imagecollection.ImageCollection-method

Returns the number of images in an ImageCollection

Description

Returns the number of images in an ImageCollection

Usage

```
## S4 method for signature 'ee.imagecollection.ImageCollection'
length(x)
```

Arguments

x	The ImageCollection to get the length of

Value

The number of images in the ImageCollection

map_create

Create a prediction map in Google Earth Engine using a fitted Random Forest model

Description

Applies a fitted Random Forest model (from R's randomForest package) to a Google Earth Engine image stack and returns an ee$Image containing the predicted values. The model is converted to a GEE Classifier via build_ee_forest, which serializes the R forest through the ICESat2VegR C++ module and constructs an EE estimator using ee$Classifier$decisionTreeEnsemble().

Authentication: This function requires an active Google Earth Engine session. Authenticate using ICESat2VegR::ee_initialize() before calling this function.

Usage

```
map_create(
  model,
  stack,
  aoi = NULL,
  reducer = c("mosaic", "median"),
  mode = c("auto", "classifier"),
  to_float = TRUE
)
```

Arguments

model	A fitted randomForest model, or a wrapper object containing an element named model. The predictor variable names in the model must match the band names in stack.
stack	An ee$Image or ee$ImageCollection providing the predictor variables (bands) matching those used to train the model. Typically the output of ee_build_AlphaEarth_embedding_terr or ee_build_hls_s1c_terrain_stack.
aoi	Optional. An EE geometry (ee$Geometry) defining the area of interest. If provided, the output prediction image is clipped to this geometry. Default is NULL.
reducer	character. Aggregation method when stack is an ee$ImageCollection. One of "mosaic" (default) or "median".
mode	character. Controls how the estimator is applied. One of: • "auto" (default) — applies the classifier using img$classify(). • "classifier" — explicitly applies img$classify().
to_float	logical. If TRUE (default), converts the output prediction band to 32-bit float.

Details

The function works for both:

- ee$Image: predictors already merged into one raster.
- ee$ImageCollection: predictions computed for each image and reduced using mosaic or median.

The output band is always named prediction_layer. Properties from a zero-pixel placeholder image are copied to preserve band metadata.

The predictor variable names used to train the model must exactly match the band names in stack. Use names(stack) to verify the available bands before training the model.

Value

An ee$Image containing one band named "prediction_layer" with the model predictions at each pixel.

See Also

build_ee_forest, ee_build_AlphaEarth_embedding_terrain_stack, seg_ancillary_extract

Examples

```
## Not run:
  Sys.setenv(EE_PROJECT = "your-ee-project-id")
  ICESat2VegR::ee_initialize()

  library(sf)
  library(randomForest)
```

```
  # -- AOI as simple bounding box --------------------------------
  aoi <- sf::st_as_sfc(sf::st_bbox(c(
    xmin = -82.4, xmax = -82.2,
    ymin =  29.6, ymax =  29.8
  ), crs = 4326))
  ee_geom <- ICESat2VegR::.as_ee_geom(aoi)

  # -- Build embedding + terrain stack ---------------------------
  stack <- ee_build_AlphaEarth_embedding_terrain_stack(
    geom       = ee_geom,
    start_year = 2025,
    end_year   = 2025
  )

  # -- Synthetic training data using stack band names ------------
  # In practice, use real ICESat-2 extracted values
  set.seed(42)
  n              <- 200
  predictor_cols <- head(names(stack), 10)

  X <- as.data.frame(matrix(
    rnorm(n * length(predictor_cols)),
    nrow      = n,
    ncol      = length(predictor_cols),
    dimnames  = list(NULL, predictor_cols)
  ))

  # Synthetic canopy height response
  y <- 5 + 2 * X[[1]] + 3 * X[[2]] + rnorm(n, sd = 1)

  # -- Fit Random Forest model -----------------------------------
  rf_model <- randomForest(x = X, y = y, ntree = 100)
  rf_model

  # -- Create prediction map in GEE ------------------------------
  pred <- map_create(
    model = rf_model,
    stack = stack,
    aoi   = ee_geom,
    mode  = "auto"
  )

  # pred is an ee$Image with one band: "prediction_layer"
  class(pred)
  names(pred)
  pred

## End(Not run)
```

map_download *map_download: create task -> start -> (monitor) -> download/return id*

Description

One function to export an EE image to Drive, Cloud Storage, or Asset; optionally monitor the task; and, for Drive/GCS, download the result locally. Returns a local file (or `SpatRaster` if `terra` is available) for Drive/GCS, or the `asset_id` (invisible) for Asset exports.

Usage

```
map_download(
  ee_image,
  method = c("drive", "gcs", "asset"),
  region,
  scale = NULL,
  file_name_prefix,
  dsn = NULL,
  drive_folder = NULL,
  gcs_bucket = NULL,
  asset_id = NULL,
  monitor = TRUE,
  task_time = 5,
  ...
)
```

Arguments

`ee_image`	An `ee$Image` to export.
`method`	One of `drive`, `gcs`, or `asset`.
`region`	EE geometry/feature collection defining the export region.
`scale`	Numeric pixel size in meters.
`file_name_prefix`	Export file prefix (used to search/download).
`dsn`	Destination path on disk (Drive/GCS methods).
`drive_folder`	Drive folder name for Drive exports.
`gcs_bucket`	Cloud Storage bucket name for GCS exports.
`asset_id`	Destination asset id for Asset exports.
`monitor`	Logical; if `TRUE`, poll the task until completion.
`task_time`	Seconds between polls when monitoring.
`...`	Additional arguments forwarded to the corresponding export helper (`ee_image_to_drive()`, `ee_image_to_gcs()`, or `ee_image_to_asset()`), e.g., CRS, pyramiding policy, `maxPixels`, etc.

Value

For drive/gcs, a local path or a SpatRaster if terra is installed. For asset, returns invisible(asset_id).

Examples

```
## Not run:
out <- map_download(
  ee_image = img, method = "drive", region = aoi, scale = 10,
  file_name_prefix = "my_pred", dsn = "pred.tif",
  drive_folder = "EE_Exports", monitor = TRUE
)

## End(Not run)
```

---

map_view *Compose a leaflet map from multiple layers (EE rasters and/or vectors)*

Description

High-level map composer capable of adding Earth Engine raster tiles (optionally tiled by AOI grid) and vector overlays (sf/SpatVector), with layer control, legends, and per-group opacity sliders.

Usage

```
map_view(
  layers,
  base_tiles = c("OSM", "Carto.Light", "Carto.Dark"),
  add_layers_control = TRUE,
  add_opacity_controls = TRUE,
  fit_to = NULL
)
```

Arguments

layers	A list of layer specs. For rasters: list(type="ee_image", x, bands, aoi=NULL, group=NULL, min_va For vectors: list(type="vector", vect, group=NULL, border_color, border_weight, fill, fill_color, fill_opacity, color_field, palette, legend_title).
base_tiles	One of OSM, Carto.Light, Carto.Dark.
add_layers_control	Logical; add layers control panel.
add_opacity_controls	Logical; add per-group opacity sliders.
fit_to	Optional EE Rectangle to fit the initial view.

Value

A leaflet htmlwidget.

Examples

```
## Not run:
m <- map_view(list(
  list(type="ee_image", x=img, bands="prediction_layer", aoi=aoi,
       group="Prediction", min_value=0, max_value=30,
       legend=list(title="Height (m)", auto="quantile")),
  list(type="vector", vect=polys_sf, group="Sites", color_field="site")
))

## End(Not run)
```

`plot,icesat2.atl03atl08_dt,missing-method`

Plot photons from ATL03 and ATL08 joined products

Description

This function plots ATL03 and ATL08 joined data along track

This function plots ATL08 attributes along track

This function plots ATL03 photons along track

Usage

```
## S4 method for signature 'icesat2.atl03atl08_dt,missing'
plot(x, y, ...)

## S4 method for signature 'icesat2.atl03atl08_dt,character'
plot(
  x,
  y,
  beam = NULL,
  col = c("gray", "#bd8421", "forestgreen", "green"),
  ...
)

## S4 method for signature 'icesat2.atl03_seg_dt,missing'
plot(x, col = "gray", ...)
```

Arguments

x	An object of class icesat2.atl03_dt
y	The attribute name for y axis
...	will be passed to the main plot
beam	Character vector indicating only one beam to process ("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"). Default is "gt1r"
col	Color for plotting the photons. Default is "gray"

Value

No return value

No return value

No return value

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

plot(
  atl03_atl08_dt,
  "ph_h",
  pch = 16, cex = 0.5,
  beam = "gt1r",
  colors = c("gray", "#bd8421", "forestgreen", "green")
)

close(atl03_h5)
close(atl08_h5)
}
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
```

```
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

plot(
  atl03_atl08_dt,
  "h_ph",
  colors = c("gray", "#bd8421", "forestgreen", "green"),
  pch = 16, cex = 0.5
)

close(atl03_h5)
close(atl08_h5)
}
if (requireNamespace("hdf5r", quietly = TRUE)) {
# Specifying the path to ATL03 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Extracting atl03 and atl03 photons and heights
atl03_photons_dt <- ATL03_seg_metadata_dt(
  atl03_h5 = atl03_h5,
  attributes = c(
    "reference_photon_lon",
    "reference_photon_lat",
    "segment_dist_x",
    "h_ph"
  )
)

plot(
  atl03_photons_dt,
  col = "gray",
  pch = 16,
  cex = 0.5
)

close(atl03_h5)
}
```

`plot.varSel` *Plot variable importance for* `varSel` *objects*

Description

Plot scaled variable importance for a `varSel` object using a horizontal barplot. Variables are ordered so that the most important metrics appear at the **top** (largest bars).

When `which = "importance"` all variables are shown, and those selected by `varSel()` are highlighted in a different color. An optional color palette (e.g., `viridis::inferno`) can be supplied to color the bars by importance.

Usage

```
## S3 method for class 'varSel'
plot(
  x,
  which = c("sel.importance", "importance"),
  main = "Random Forest variable importance",
  xlab = "Scaled importance",
  col.selected = "steelblue",
  col.other = "grey80",
  palette = NULL,
  legend.loc = "bottomright",
  ...
)
```

Arguments

`x`	An object of class `varSel`.
`which`	Character; either `sel.importance` (default) to plot the importance of selected variables only, or `importance` to plot the full-model importance (all predictors).
`main`	Plot title.
`xlab`	Label for the x-axis (importance scale).
`col.selected`	Base color for selected variables when no palette is given.
`col.other`	Base color for non-selected variables when no palette is given.
`palette`	Optional color palette. Can be: - a function `f(n)` returning `n` colors (e.g., `viridis::inferno`), - or a character vector of colors used to build a gradient via grDevices::colorRampPalette. The palette is applied to all bars and then the color of selected variables is overridden by `col.selected` when `which = "importance"`.
`legend.loc`	Legend location, e.g. "bottomright"
`...`	Additional graphical arguments passed to graphics::barplot.

Examples

```
require(randomForest)

data(airquality)
airquality <- na.omit(airquality)

xdata <- airquality[, 2:6]
ydata <- airquality[, 1]

ntree = 200

vs <- varSel(xdata, ydata, ntree = ntree, min.imp = 0.2)

## Selected variables only
plot(vs, which = "sel.importance")

## All variables, highlighting selected ones, with inferno palette
if (requireNamespace("viridis", quietly = TRUE)) {
  plot(vs, which = "importance", palette = viridis::inferno)
}
```

---

plot_icesat2_orbit_animation

Plot ICESat-2 orbital tracks as a 3D globe animation

Description

Creates an interactive HTML animation of ICESat-2 orbital tracks from one or more KML files or from a `terra::SpatVector` returned by `rgt_extract()`.

Usage

```
plot_icesat2_orbit_animation(
  kml_files = NULL,
  rgt = NULL,
  output_dir = tempdir(),
  output_file = "ICESat2_orbit_animation.html",
  track_speed = 2,
  earth_rotation_speed = 2,
  launch = TRUE
)
```

Arguments

`kml_files`	Character vector. Path to one or more KML files. If `NULL` and `rgt = NULL`, all example KML files available in `inst/extdata` are used.

rgt	A terra::SpatVector returned by rgt_extract(). If provided, this object is used instead of kml_files.
output_dir	Character. Folder where the HTML animation and required assets will be written. Default is tempdir().
output_file	Character. Name of the output HTML file.
track_speed	Numeric. Initial animation speed. Default is 2.
earth_rotation_speed	Numeric. Initial Earth rotation speed. Default is 2.
launch	Logical. If TRUE, starts a local server and opens the HTML.

Value

Invisibly returns the path to the generated HTML file.

Examples

```
## Not run:
plot_icesat2_orbit_animation()

atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
rgt_line <- rgt_extract(h5 = atl03_h5, line = TRUE)

plot_icesat2_orbit_animation(
  rgt = rgt_line,
  launch = TRUE
)

close(atl03_h5)

## End(Not run)
```

---

predict_h5 *Model prediction over data.tables using HDF5 file as output*

Description

Model prediction over a data.table from ATL03 or ATL08 data containing geolocation data. It can both append results to an existing HDF5 file or create a new file, allowing to incrementally add predictions to the file to avoid memory issues.

Usage

```
predict_h5(model, dt, output)
```

Arguments

model	The trained model object
dt	The input data.table to run the model
output	The output HDF5 file path

Value

An icesat2.predict_h5, which is an h5 file with latitude, longitude and prediction datasets.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file(
  "extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl03_seg_dt <- ATL03_seg_metadata_dt(atl03_h5)

linear_model <- stats::lm(h_ph ~ segment_ph_cnt, data = atl03_seg_dt)
output_h5 <- tempfile(fileext = ".h5")
predicted_h5 <- predict_h5(linear_model, atl03_seg_dt, output_h5)

# List datasets
predicted_h5$ls()$name

# See predicted values
head(predicted_h5[["prediction"]][])

# Close the file
close(predicted_h5)

}
```

---

predict_h5,ANY,icesat2.atl03_seg_dt,character-method

S4 method for predicting using HDF5 file as output

Description

This method is used to predict using a trained model and save the results in HDF5 file.

Usage

```
## S4 method for signature 'ANY,icesat2.atl03_seg_dt,character'
predict_h5(model, dt, output)
```

Arguments

model	The trained model object
dt	The input data.table to run the model
output	The output file path

Value

An icesat2.predict_h5, which is an h5 file with latitude, longitude and prediction

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file(
  "extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl03_seg_dt <- ATL03_seg_metadata_dt(atl03_h5)

linear_model <- stats::lm(h_ph ~ segment_ph_cnt, data = atl03_seg_dt)
output_h5 <- tempfile(fileext = ".h5")

predicted_h5 <- predict_h5(linear_model, atl03_seg_dt, output_h5)

# List datasets
predicted_h5$ls()$name

# See predicted values
head(predicted_h5[["prediction"]][])

# Close the file
close(predicted_h5)
}
```

---

predict_h5,ANY,icesat2.atl08_dt,character-method

S4 method for predicting using HDF5 file as output

Description

This method is used to predict using a trained model and save the results in HDF5 file.

Usage

```
## S4 method for signature 'ANY,icesat2.atl08_dt,character'
predict_h5(model, dt, output)
```

Arguments

model	The trained model object
dt	The input data.table to run the model
output	The output file path

Details

This method is used to predict using a trained model and save the results in an HDF5 file.

Value

An icesat2.predict_h5, which is an h5 file with latitude, longitude and prediction datasets.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file(
  "extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)
atl08_dt <- ATL08_seg_attributes_dt(atl08_h5)
linear_model <- stats::lm(h_canopy ~ canopy_openness, data = atl08_dt)
output_h5 <- tempfile(fileext = ".h5")
predicted_h5 <- predict_h5(linear_model, atl08_dt, output_h5)

# List datasets
predicted_h5$ls()$name

# See predicted values
head(predicted_h5[["prediction"]][])

# Close the file
close(predicted_h5)

}
```

---

prepend_class *Prepend a class to an object's list of classes*

Description

Prepend a class to an object's list of classes

Usage

```
prepend_class(obj, className)
```

Arguments

obj	The object to which prepend the class.
className	character with the name of the class to prepend.

Value

Nothing, it replaces the class attribute in place

---

randomSampling *Pure random sampling method*

Description

Pure random sampling method

Usage

```
randomSampling(size)
```

Arguments

size	numeric. The sample size. Either an integer for absolute number of samples, or a value between (0, 1) to sample a percentage relative to the number of available observations.

Value

A icesat2_sampling_method-class object for use in sample.

See Also

sample, gridSampling, stratifiedSampling, spacedSampling, geomSampling, rasterSampling

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")
atl08_h5   <- ATL08_read(atl08_path = atl08_path)
atl08_dt   <- ATL08_seg_attributes_dt(atl08_h5)

# Sample 5 random observations
set.seed(1)
sampled <- ICESat2VegR::sample(atl08_dt, method = randomSampling(5))
head(sampled)

# Sample 50% of observations
set.seed(1)
sampled_pct <- ICESat2VegR::sample(atl08_dt, method = randomSampling(0.5))
```

```
nrow(sampled_pct)

close(atl08_h5)
}
```

`rasterize_h5` *Rasterizes the model prediction saved in the HDF5 file*

Description

This is used after running the prediction using `predict_h5()` function to rasterize and aggregate the prediction within raster cells. By default it will calculate (n, mean, variance * (n - 1), min, max, sd) in a single raster file with those 6 bands in that order. You can modify this behavior by changing the `agg_function`, `agg_join` and `finalizer` parameters, see details section.

Usage

```
rasterize_h5(h5_input, output, bbox, res, ...)
```

Arguments

`h5_input`	The input HDF5 file path
`output`	The output raster file path
`bbox`	The bounding box of the raster
`res`	The resolution of the raster
`...`	Additional parameters (see details section)

Details

This function will create five different aggregate statistics (n, mean, variance, min, max).

Within `...` additional parameters we can use:

`agg_function`: is a formula which return a data.table with the aggregate function to perform over the data. The default is:

```
~data.table(
    n = length(x),
    mean = mean(x,na.rm = TRUE),
    variance = var(x) * (length(x) - 1),
    min = min(x, na.rm=T),
    max = max(x, na.rm=T)
  )
```

`agg_join`: is a function to merge two data.table aggregates from the `agg_function`. Since the h5 files will be aggregated in chunks to avoid memory overflow, the statistics from the different chunks should have a function to merge them.

The default function is:

```
function(x1, x2) {
    combined = data.table()
    x1$n[is.na(x1$n)] = 0
    x1$mean[is.na(x1$mean)] = 0
    x1$variance[is.na(x1$variance)] = 0
    x1$max[is.na(x1$max)] = -Inf
    x1$min[is.na(x1$min)] = Inf

    combined$n = x1$n + x2$n

    delta = x2$mean - x1$mean
    delta2 = delta * delta

    combined$mean = (x1$n * x1$mean + x2$n * x2$mean) / combined$n
    combined$variance = x1$variance + x2$variance +
      delta2 * x1$n * x2$n / combined$n

    combined$min = pmin(x1$min, x2$min, na.rm=F)
    combined$max = pmax(x1$max, x2$max, na.rm=F)
    return(combined)
}
```

`finalizer`: is a list of formulas to generate the final rasters based on the intermediate statistics from the previous functions. The default `finalizer` will calculate the sd, based on the `variance` and `n` values. It is defined as:

```
list(
  sd = ~sqrt(variance/(n - 1)),
)
```

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {

atl08_path <- system.file(
  "extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

atl08_h5 <- ATL08_read(atl08_path = atl08_path)
atl08_dt <- ATL08_seg_attributes_dt(atl08_h5)

xmin <- min(atl08_dt$longitude)
xmax <- max(atl08_dt$longitude)
ymin <- min(atl08_dt$latitude)
ymax <- max(atl08_dt$latitude)

linear_model <- stats::lm(h_canopy ~ canopy_openness, data = atl08_dt)
output_h5    <- tempfile(fileext = ".h5")
```

```
predicted_h5 <- predict_h5(linear_model, atl08_dt, output_h5)
output_raster <- tempfile(fileext = ".tif")

rasterize_h5(
  predicted_h5,
  output = output_raster,
  bbox   = terra::ext(xmin, xmax, ymin, ymax),
  res    = 0.003
)

# Load and plot the raster
r <- terra::rast(output_raster)
terra::plot(r[[1]])

close(atl08_h5)

}
```

`rasterize_h5,icesat2.predict_h5,character,SpatExtent,numeric-method`
Rasterizes the model prediction saved in the HDF5 file

Description

Rasterizes the model prediction saved in the HDF5 file

Usage

```
## S4 method for signature 'icesat2.predict_h5,character,SpatExtent,numeric'
rasterize_h5(
  h5_input,
  output,
  bbox,
  res,
  chunk_size = 512 * 512,
  agg_function = agg_function_default,
  agg_join = agg_join_default,
  finalizer = finalizer_default
)
```

Arguments

`h5_input`	The input HDF5 file path
`output`	The output raster file path
`bbox`	The bounding box of the raster
`res`	The resolution of the raster
`chunk_size`	The chunk size to read the HDF5 file

agg_function	The function to aggregate the data
agg_join	The function to join the aggregated data
finalizer	The function to finalize the raster

`rasterSampling` *Get observations sampled by raster class*

Description

Get observations sampled by raster class

Usage

```
rasterSampling(size, raster, chainSampling = NULL)
```

Arguments

size	numeric. The sample size per raster class. Either an integer or a value between `(0, 1)` for a percentage.
raster	A SpatRaster object opened with rast.
chainSampling	optional. Chain with another sampling method.

Value

A icesat2_sampling_method-class object for use in sample.

See Also

sample, randomSampling

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")
atl08_h5   <- ATL08_read(atl08_path = atl08_path)
atl08_dt   <- ATL08_seg_attributes_dt(atl08_h5)

# Create a raster over the data extent
r <- terra::rast(
  xmin       = min(atl08_dt$longitude),
  xmax       = max(atl08_dt$longitude),
  ymin       = min(atl08_dt$latitude),
  ymax       = max(atl08_dt$latitude),
  resolution = 0.005,
  crs        = "EPSG:4326"
)
terra::values(r) <- sample(1:3, terra::ncell(r), replace = TRUE)
```

```
# Sample 2 observations per raster class
set.seed(1)
sampled <- ICESat2VegR::sample(
  atl08_dt,
  method = rasterSampling(size = 2, raster = r)
)
head(sampled)

close(atl08_h5)
}
```

`rbindlist2` *Row-bind a list of objects while preserving class*

Description

Convenience wrapper around `data.table::rbindlist()` that **row-binds a list of homogeneous objects and then restores their class**. This is especially useful for custom S3/S4-like classes used in ICESat2VegR (e.g. `icesat2.atl03_dt`, `icesat2.atl08_dt`), where a plain `data.table::rbindlist()` call would drop the original class attribute.

Usage

```
rbindlist2(l, ...)
```

Arguments

`l`	A list whose elements are all of the *same* class, typically `data.table`, `data.frame`, or a custom subclass (e.g. `icesat2.*_dt`). All elements must share identical `class()`; otherwise an error is raised.
`...`	Additional arguments passed directly to `data.table::rbindlist()`, such as `use.names` or `fill`.

Value

A single `data.table` (the result of `data.table::rbindlist()`), with its `class` attribute reset to match the class of the input elements.

rgt_extract *Extract Reference Ground Track from ICESat-2 ATL03 or ATL08 Data*

Description

Extracts the reference ground track from ICESat-2 ATL03 or ATL08 data and returns it as a `terra::SpatVector`. Optionally, the extracted ground track can also be written to disk, for example as a KML file.

Usage

```
rgt_extract(h5, line = TRUE, output = NULL)
```

Arguments

`h5`	An ICESat-2 ATL03 or ATL08 object, usually the output of `ATL03_read()` or `ATL08_read()`.
`line`	Logical. If `TRUE`, returns the reference ground track as a line. If `FALSE`, returns the bounding polygon. Default is `TRUE`.
`output`	Character or `NULL`. Optional output vector file path. If provided, the extracted ground track is written using `terra::writeVector()`. Use a file extension supported by GDAL, such as `.kml`, `.gpkg`, or `.shp`.

Details

This function uses ICESat-2 orbit information stored in the `orbit_info` group to derive the reference ground track. When `line = TRUE`, the function estimates the centerline from the bounding polygon coordinates.

The returned object can be passed directly to `plot_icesat2_orbit_animation()` using the `rgt` argument.

Value

A `terra::SpatVector` containing the extracted reference ground track.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)
```

```
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

rgt_line <- rgt_extract(h5 = atl03_h5, line = TRUE)

rgt_extract(
  h5 = atl03_h5,
  line = TRUE,
  output = file.path(tempdir(), "atl03_rgt.kml")
)

close(atl03_h5)

}
```

`sample` *Sample method for applying multiple sampling methods*

Description

Sample method for applying multiple sampling methods

Usage

```
sample(x, ..., method)
```

Arguments

`x`	the generic input data to be sampled
`...`	generic params to pass ahead to the specific sampling function
`method`	the sampling method to use.

Details

It is expected that the user pass a `method` parameter within `...`

See Also

`randomSampling()`, `spacedSampling()`, `gridSampling()`, `stratifiedSampling()`, `geomSampling()`, `rasterSampling()`

`sample_ATL_granules_by_year` *Sample ICESat-2 ATL granule URLs by year*

Description

Given a vector (or first column of a matrix/data frame) of ICESat-2 ATL03/ATL08 granule URLs or file paths, this function:

- Attempts to parse the acquisition year from each URL/path.
- Groups granules by year.
- Samples up to `n_per_year` unique URLs per year.

This is useful for creating manageable subsets of ATL granules for testing, model fitting, or cross-validation.

Usage

```
sample_ATL_granules_by_year(urls, n_per_year = 5, seed = NULL)
```

Arguments

`urls`	Character vector, matrix, or data frame containing granule URLs or file paths. If a matrix or data frame is provided, the first column is used.
`n_per_year`	Integer. Maximum number of URLs to sample per year (default `5`).
`seed`	Optional integer seed passed to `set.seed()` to make the sampling reproducible. If `NULL` (default), the random state is left unchanged.

Details

The year is extracted using the following heuristics:

1. A path component of the form `/YYYY/` (e.g., `.../2020/...`).
2. A filename pattern of the form `ATL0[38]_YYYYMMDD...`.

URLs for which no year can be detected are dropped with a warning.

Value

A data frame with columns:

- `year`: integer acquisition year.
- `url`: the sampled URL or file path.

Rows are ordered by `year` and then `url`.

Examples

```
urls <- c(
  "https://example.org/ATL03_20200101000000_001.h5",
  "https://example.org/ATL03_20200102000000_002.h5",
  "https://example.org/ATL03_20210101000000_003.h5"
)

sample_df <- sample_ATL_granules_by_year(urls, n_per_year = 1, seed = 42)
sample_df
```

`search_datasets` *Search for Google Earth Engine datasets*

Description

Search for Google Earth Engine datasets

Usage

```
search_datasets(
  ...,
  title = TRUE,
  description = TRUE,
  operator = "and",
  refresh_cache = FALSE
)
```

Arguments

`...`	character arguments to search for within title and/or description
`title`	logical. Whether should search within the title, default TRUE.
`description`	logical. Whether should search within the description of the dataset, default TRUE.
`operator`	character. Should be either "OR" or "AND" to tell if the search needs to include all the queries "AND" or any of the queries "OR". Default "AND".
`refresh_cache`	flag indicating if the results cache should be refreshed, default FALSE.

Value

A `data.table` containing the id, title and description of the datasets that matched the supplied query ordered by relevance.

seg_ancillary_extract *Given a stack image raster from GEE retrieve the point geometry with values for the images*

Description

Given a stack image raster from GEE retrieve the point geometry with values for the images

Usage

```
seg_ancillary_extract(stack, geom, scale = 10, chunk_size = 1000)
```

Arguments

stack	A single image or a vector/list of images from Earth Engine.
geom	A geometry from terra::SpatVector read with terra::vect.
scale	The scale in meters for the extraction (image resolution).
chunk_size	If the number of observations is greater than 1000, it is recommended to chunk the results for not running out of memory within GEE server, default is chunk by 1000.

Value

A data.table::data.table with the properties extracted from the ee images.

slope *Compute terrain slope (degrees) from a DEM image*

Description

Computes the terrain **slope** in degrees for each pixel of an Earth Engine ee$Image representing a digital elevation model (DEM).

This method is a thin wrapper around ee$Terrain$slope() and returns the same output structure. Slope is computed using Earth Engine's internal gradient operator, which relies on the 4-connected neighborhood around each pixel. As a result, edge pixels may contain missing values depending on the input DEM.

Usage

```
slope(x)
```

Arguments

x	An ee$Image representing a DEM from which slope will be derived.

Value

An ee$Image with one band named slope containing terrain slope values in degrees.

Examples

```
## Not run:
  ee <- reticulate::import("ee")
  dem <- ee$Image("NASA/NASADEM_HGT/001")
  slp <- slope(dem)

## End(Not run)
```

---

spacedSampling *Get observations with a minimum radius distance between samples*

Description

Get observations with a minimum radius distance between samples

Usage

```
spacedSampling(size, radius, spatialIndex = NULL, chainSampling = NULL)
```

Arguments

size	numeric. The sample size. Either an integer or a value between (0, 1) for a percentage.
radius	numeric. The minimum radius in decimal degrees between samples.
spatialIndex	optional. A pre-built ANNIndex spatial index for accelerating the search. Created automatically if not provided.
chainSampling	optional. Chain with another sampling method.

Value

A icesat2_sampling_method-class object for use in sample.

See Also

sample, randomSampling

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")
atl08_h5   <- ATL08_read(atl08_path = atl08_path)
atl08_dt   <- ATL08_seg_attributes_dt(atl08_h5)

# Sample up to 5 observations with minimum 0.01 degree spacing
set.seed(1)
sampled <- ICESat2VegR::sample(
  atl08_dt,
  method = spacedSampling(size = 5, radius = 0.01)
)
head(sampled)

close(atl08_h5)
}
```

---

stratifiedSampling *Get samples stratified by a variable binning histogram*

Description

Get samples stratified by a variable binning histogram

Usage

```
stratifiedSampling(size, variable, chainSampling = NULL, ...)
```

Arguments

`size`	numeric. The sample size per stratum. Either an integer or a value between `(0, 1)` for a percentage.
`variable`	character. Variable name used for the stratification.
`chainSampling`	optional. Chain with another sampling method.
`...`	Additional arguments forwarded to `hist`, where you can manually define the breaks.

Value

A `icesat2_sampling_method-class` object for use in `sample`.

See Also

`sample`, `randomSampling`

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")
atl08_h5   <- ATL08_read(atl08_path = atl08_path)
atl08_dt   <- ATL08_seg_attributes_dt(atl08_h5)

# Sample 2 observations per h_canopy stratum
set.seed(1)
sampled <- ICESat2VegR::sample(
  atl08_dt,
  method = stratifiedSampling(size = 2, variable = "h_canopy")
)
head(sampled)

close(atl08_h5)
}
```

---

to_vect *Convert ICESat-2 classes, data.frame/data.table, and sf to terra::SpatVector*

Description

Generic function to convert ICESat-2 objects, plain tables (data.frame / data.table), and `sf` objects to a `terra::SpatVector`.

Usage

```
to_vect(x, ...)

## S4 method for signature 'data.frame'
to_vect(x, lon = NULL, lat = NULL, crs = "EPSG:4326", ...)

## S4 method for signature 'data.table'
to_vect(x, lon = NULL, lat = NULL, crs = "EPSG:4326", ...)

## S4 method for signature 'icesat2.atl03_seg_dt'
to_vect(x, ...)

## S4 method for signature 'icesat2.atl08_dt'
to_vect(x, ...)

## S4 method for signature 'icesat2.atl03atl08_dt'
to_vect(x, ...)

## S4 method for signature 'icesat2.atl03_dt'
to_vect(x, ...)
```

```
## S4 method for signature 'icesat2.atl03_atl08_seg_dt'
to_vect(x, ...)

## S4 method for signature 'sf'
to_vect(x, target_crs = NULL, ...)
```

Arguments

x	An sf object (POINT geometry recommended).
...	Ignored.
lon, lat	For data.frame/data.table, the column names to use as coordinates.
crs	CRS string for the resulting SpatVector. Default is "EPSG:4326".
target_crs	Optional CRS string (e.g., "EPSG:4326") to reproject the output.

Details

For plain tables, you can either:

1. Provide lon= and lat= column names, or
2. Rely on defaults (longitude / latitude), or
3. Use common ICESat-2 column names (lon_ph/lat_ph, reference_photon_lon/reference_photon_lat).

Value

A terra::SpatVector object.

A terra::SpatVector

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
# ICESat2VegR examples (package objects keep their classes):
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl03_segment_dt <- ATL03_seg_metadata_dt(atl03_h5 = atl03_h5)
atl03_segment_vect <- to_vect(atl03_segment_dt)
terra::plot(atl03_segment_vect, col = atl03_segment_vect$segment_ph_cnt)
close(atl03_h5)

# Plain data.frame / data.table usage:
df <- data.frame(longitude = c(-84, -84.1), latitude = c(29.6, 29.7), z = 1:2)
v  <- to_vect(df)
v2 <- to_vect(df, lon = "longitude", lat = "latitude", crs = "EPSG:4326")

}
```

varSel *Random Forest Variable Selection (Breiman-only)*

Description

Implements the Random Forest model selection approach of Murphy et al. (2010), using Breiman's original **randomForest** implementation. This is a simplified adaptation of `rfUtilities::rf.modelSel`, restricted to the Breiman implementation and modified for the ICESat2VegR package.

It returns the selected variables and importance metrics, but does not fit a final model.

Usage

```
varSel(
  xdata,
  ydata,
  imp.scale = c("mir", "se"),
  r = c(0.25, 0.5, 0.75),
  min.imp = NULL,
  seed = NULL,
  parsimony = NULL,
  kappa = FALSE,
  ...
)
```

Arguments

`xdata`	Matrix or data.frame of predictor variables (columns = predictors).
`ydata`	Response vector. For classification, `ydata` must be a factor; otherwise the model is fit in regression mode.
`imp.scale`	Character; type of scaling for importance values, either `mir` or `se`. Default is `mir`.
`r`	Numeric vector of importance percentiles to test, e.g., `c(0.25, 0.50, 0.75)`. These percentiles are used to define thresholds for building nested models during selection.
`min.imp`	Optional numeric in $[0, 1]$ used as a minimum scaled importance cutoff (in MIR or SE scale) to filter the final set of variables. Variables whose scaled importance is below this threshold are dropped from the selected set `selvars`. If `NULL` (default), no cutoff is applied after the Murphy-style model selection.
`seed`	Optional integer; sets the random seed in the global R environment. This is strongly recommended for reproducibility.
`parsimony`	Numeric in (0,1); threshold for selecting among competing models. If specified, models whose errors are within `parsimony` of the best model (OOB / class error for classification, variance explained / MSE for regression) are considered, and the one with the fewest parameters is chosen.

kappa	Logical; use the chance-corrected κ statistic as the primary optimization criterion for classification instead of percent correctly classified (PCC). This corrects PCC for random agreement.
...	Additional arguments passed to randomForest::randomForest(), e.g. ntree = 1000, replace = TRUE, proximity = TRUE.

Details

For classification, ensure that ydata is a factor; otherwise the model is fit in regression mode.

Selection strategy:

- For classification, candidate models are compared using OOB PCC (or κ, if kappa = TRUE) and maximum within-class error, with preference for more parsimonious models.
- For regression, candidate models are compared using percent variance explained and MSE, with preference for more parsimonious models.
- After the best model is chosen, the optional min.imp cutoff is applied to the scaled importance (MIR/SE) from the full model to remove weak predictors from the final set selvars.

Typical choices for min.imp:

- MIR: min.imp in $[0.1, 0.3]$ (e.g., keep variables with at least 20\
- SE: min.imp in $[0.01, 0.05]$, remembering that SE-scaled importance values sum to 1.

Value

An object of class varSel (a list) with components:

- selvars Character vector of selected variable names (after applying min.imp, if provided).
- test Data.frame of model-selection diagnostics, containing error metrics, threshold, number of parameters, and the variables used in each candidate model (for inspection only).
- importance Data.frame of scaled importance values for all variables in the full model (columns parameter, importance).
- sel.importance Data.frame of scaled importance values for the selected variables.
- parameters List of variables used in each candidate model.
- scaling Character; the importance scaling used (mir or se).

See Also

randomForest::randomForest() and plot.varSel().

Examples

```
require(randomForest)

data(airquality)
airquality <- na.omit(airquality)

xdata <- airquality[, 2:6]
```

```
ydata <- airquality[, 1]
ntree <- 500

## Regression example with MIR scaling and importance cutoff
vs.regress <- varSel(
  xdata     = xdata,
  ydata     = ydata,
  imp.scale = "mir",
  ntree     = ntree,
  min.imp   = 0.2
)

vs.regress$selvars

## Plot all variables, highlighting selected ones
plot(vs.regress, which = "importance")
```

`vect_as_ee` *Convert vector data to Google Earth Engine FeatureCollection (no rgee)*

Description

`vect_as_ee()` converts vector data to a Google Earth Engine `ee$FeatureCollection` using **reticulate** to call the official Python Earth Engine API-**without** relying on **rgee**. It accepts `sf`/`sfc`/`sfg` objects or `terra::SpatVector`, validates and reprojects geometries, and either returns an in-memory FeatureCollection or exports to an EE Asset.

Usage

```
vect_as_ee(
  x,
  via = c("getInfo", "getInfo_to_asset"),
  assetId = NULL,
  overwrite = TRUE,
  proj = "EPSG:4326",
  make_valid = TRUE,
  quiet = FALSE,
  monitoring = TRUE,
  poll_interval_sec = 5,
  poll_timeout_sec = 3600
)
```

Arguments

`x`	An input vector object: an `sf`, `sfc`, or `sfg` object, or a `terra::SpatVector`. For `sfg`/`sfc`, a simple point/line/polygon geometry is accepted; for `SpatVector`, it will be converted to `sf`.

via	Character; one of c("getInfo","getInfo_to_asset").
	• getInfo returns an in-memory ee$FeatureCollection built from the GeoJSON of x.
	• getInfo_to_asset creates an EE export task to save the collection to an Earth Engine **Asset** (requires assetId).
assetId	Character; EE asset id (e.g., users/you/my_fc). **Required** when via = "getInfo_to_asset".
overwrite	Logical; if TRUE, attempt to delete an existing EE asset at assetId before export.
proj	Character CRS string (e.g., EPSG:4326). Input will be transformed to this CRS before conversion. Default is EPSG:4326.
make_valid	Logical; if TRUE, attempt to repair invalid geometries (sf::st_make_valid() / terra::makeValid()).
quiet	Logical; if FALSE, print progress messages (e.g., task state).
monitoring	Logical; when exporting to asset, if TRUE poll the task until completion or timeout, otherwise return immediately after starting.
poll_interval_sec	Numeric; seconds to wait between task status checks when monitoring = TRUE. Default 5.
poll_timeout_sec	Numeric; maximum seconds to keep polling before timing out. Default 3600 (1 hour).

Details

- The function converts x to sf, enforces a defined CRS, optionally fixes invalid geometries, and transforms to proj (default WGS84).
- Properties are carried alongside geometry; however, Earth Engine does not accept POSIX* timestamp columns or property names containing '.'. The function stops with a clear error if such columns are found-convert them to character or rename before calling.
- GeoJSON is built in-memory (via **geojsonsf** when available) or via a temporary .geojson file as a fallback, then parsed to a list for ee$FeatureCollection.
- Requires a properly initialized EE Python environment (ee.Initialize() in the active Python session used by **reticulate**).

Value

If via = "getInfo", returns an in-memory ee$FeatureCollection object (reticulate Python object). If via = "getInfo_to_asset", returns an ee$FeatureCollection **referencing** assetId (after successful export), or returns immediately if monitoring = FALSE.

Errors & Constraints

- Undefined CRS: the function stops if x has no CRS set.
- Unsupported columns: the function stops if any property is POSIX* or if names contain dots (.).
- Export failures: if via = "getInfo_to_asset" and the EE task fails or is cancelled, an error is thrown when monitoring = TRUE.

See Also

- Earth Engine Python API docs: `ee$FeatureCollection`
- Geometry repair: `sf::st_make_valid()`, `terra::makeValid()`
- GeoJSON helpers: **geojsonsf**, `sf::st_write()`

Examples

```
## Not run:
# -- Prerequisites (Python side):
# import ee; ee.Initialize()
#
# Example with sf POINTS
library(sf)
pts <- st_as_sf(data.frame(
  id = 1:3,
  x  = c(-84.02171, -84.02025, -84.02026),
  y  = c(31.29891, 31.31945, 31.31938)
), coords = c("x","y"), crs = "EPSG:4326")

# In-memory FeatureCollection:
fc <- vect_as_ee(pts, via = "getInfo")

# Export to an EE asset (monitor until done):
# fc_asset <- vect_as_ee(
#   pts,
#   via = "getInfo_to_asset",
#   assetId = "users/you/demo_points",
#   overwrite = TRUE,
#   monitoring = TRUE
# )

## End(Not run)
```

`write_geojson` *Safely write a GeoJSON file from a lon/lat table*

Description

Converts a data frame or similar tabular object with longitude/latitude columns into a point layer and writes it to disk as a GeoJSON file. The function first attempts to use **terra** via `ICESat2VegR::to_vect()`, and falls back to **sf** if available.

Usage

```
write_geojson(
  dt,
  path,
```

```
  xcol = "lon",
  ycol = "lat",
  crs = "EPSG:4326",
  overwrite = TRUE
)
```

Arguments

dt	A data frame, data.table, or similar object containing at least two numeric columns for coordinates.
path	Character. Path to the output GeoJSON file.
xcol, ycol	Character. Names of the longitude and latitude columns in dt. Defaults are lon and lat.
crs	Character. Coordinate reference system of the input coordinates. Default is EPSG:4326 (longitude/latitude in WGS84).
overwrite	Logical. If TRUE (default), allow overwriting an existing file.

Value

Invisibly returns TRUE on success. An error is thrown if both the **terra**-based and **sf**-based write attempts fail.

Examples

```
pts <- data.frame(
  id  = 1:3,
  lon = c(-82.35, -82.34, -82.33),
  lat = c( 29.65,  29.66,  29.67)
)

write_geojson(pts, tempfile(fileext = ".geojson"))
```

[,icesat2.granules_cloud,ANY,ANY,ANY-method

Subset Granules

Description

This method subsets the granules slot of an icesat2.granules_cloud object.

Usage

```
## S4 method for signature 'icesat2.granules_cloud,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

x	An object of class icesat2.granules_cloud.
i	Subset index.
j	Unused, just to match the generic signature.
...	Additional arguments (not used).
drop	Unused, just to match the generic signature.

Value

An object of class icesat2.granule_cloud.

[[,icesat2.granules_cloud,ANY,ANY-method
Extract Granules

Description

This method extracts a single element from the granules slot of an icesat2.granules_cloud object.

Usage

```
## S4 method for signature 'icesat2.granules_cloud,ANY,ANY'
x[[i, j, ...]]
```

Arguments

x	An object of class icesat2.granules_cloud.
i	Extraction index.
j	Unused, just to match the generic signature.
...	Additional arguments (not used).

Value

An object of class icesat2.granule_cloud.

Examples

```
## Not run:
lower_left_lon <- -83.26
lower_left_lat <- 31.95
upper_right_lon <- -83.11
upper_right_lat <- 32.46

daterange <- c("2019-04-12", "2019-05-03")
```

```
ATL08_granules_cloud <- ATLAS_dataFinder(
  short_name = "ATL08",
  lower_left_lon,
  lower_left_lat,
  upper_right_lon,
  upper_right_lat,
  version = "007",
  daterange = daterange,
  persist = TRUE,
  cloud_computing = TRUE
)

ATL08_h5_cloud <- ATL08_read(ATL08_granules_cloud[1])

ATL08_h5_cloud[['gt1l']]
close(ATL08_h5_cloud)

## End(Not run)
```

`[[.GDALDataset` *GDALDataset [[]] accessor*

Description

This function gives access to the GDALRasterBand using [[i]], where i is the band index to return.

Usage

```
## S3 method for class 'GDALDataset'
x[[slice]]
```

Arguments

`x`	GDALDatset. Automatically obtained from GDALDataset[[]] call.
`slice`	Integer. The index for the band to access.

Value

An object of GDALRasterBand R6 class.

[[.GDALRasterBand *GDALRasterBand [[]] getter*

Description

This function gives access to the GDALRasterBand using [[i]], where i is the band index to return.

Usage

```
## S3 method for class 'GDALRasterBand'
x[[blockX, blockY]]
```

Arguments

x	GDALRasterBand. Automatically obtained from GDALDataset[[]] call.
blockX	Integer. The x index for block to access.
blockY	Integer. The y index for block to access.

Value

Nothing, this is a setter

[[<-.GDALRasterBand *GDALRasterBand [[]]= setter*

Description

This function gives access to the GDALRasterBand using [[i]], where i is the band index to return.

Usage

```
## S3 replacement method for class 'GDALRasterBand'
x[[blockX, blockY]] <- value
```

Arguments

x	GDALRasterBand. Automatically obtained from GDALDataset[[]] call.
blockX	Integer. The x index for block to access.
blockY	Integer. The y index for block to access.
value	Integer. The value buffer to write

Value

Nothing, this is a setter

Index