Package 'IceSat2R'

March 5, 2026

Type Package

Title 'ICESat-2' Altimeter Data using R

Version 1.0.9

Date 2026-03-05

Description Programmatic connection to the 'OpenAltimetry' API <https://openaltimetry.earthdatacloud.nasa.gov/data/openapi/swagger-ui/index.html/> to download and process 'ATL03' (Global 'Geolocated' Photon Data), 'ATL06' (Land Ice Height), 'ATL07' (Sea Ice Height), 'ATL08' (Land and Vegetation Height), 'ATL10' (Sea Ice 'Freeboard'), 'ATL12' (Ocean Surface Height) and 'ATL13' (Inland Water Surface Height) 'ICESat-2' Altimeter Data. The user has the option to download the data by selecting a bounding box from a 1- or 5-degree grid globally utilizing a shiny application. The 'ICESat-2' mission collects 'altimetry' data of the Earth's surface. The sole instrument on 'ICESat-2' is the Advanced Topographic Laser Altimeter System (ATLAS) instrument that measures ice sheet elevation change and sea ice thickness, while also generating an estimate of global vegetation biomass. 'ICESat-2' continues the important observations of ice-sheet elevation change, sea-ice 'freeboard', and vegetation canopy height begun by 'ICESat' in 2003.

License GPL-3

Encoding UTF-8

URL https://github.com/mlampros/IceSat2R

BugReports https://github.com/mlampros/IceSat2R/issues

Depends R(>= 4.1.0)

Imports glue, sf, lwgeom, units, data.table, httr, utils, foreach, tools, doParallel, magrittr, leaflet, leafgl, htmltools, htmlwidgets, leafsync, miniUI, shiny, rnaturalearth, lubridate, rvest, withr

Suggests rmarkdown, knitr, DT, mapview, grDevices, stargazer, reshape2, plotly, geodist, CopernicusDEM, terra, testthat (>= 3.0.0)

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Lampros Mouselimis [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8024-1546>),
OpenAltimetry.org [cph]

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

Repository CRAN

Date/Publication 2026-03-05 08:40:02 UTC

Contents

`available_nominal_orbits` *Nominal mission orbits*

Description

This function allows the user to view the nominal orbits (all or a selection)

Usage

```
available_nominal_orbits(
  orbit_area = NULL,
  technical_specs_url = "https://icesat-2.gsfc.nasa.gov/science/specs",
  verbose = FALSE
)
```

Arguments

orbit_area	either NULL or a character string specifying the earth partition to use, it can be one of 'antarctic', 'arctic', 'western_hemisphere' and 'eastern_hemisphere'
technical_specs_url	a character string specifying the technical specs website
verbose	a boolean. If TRUE then information will be printed out in the console

Value

a list object with the available nominal orbits

References

https://icesat-2.gsfc.nasa.gov/science/specs

Examples

```
require(IceSat2R)

#.........................
# all available nominal orbits
#.........................

nomin_orb = available_nominal_orbits(verbose = TRUE)
nomin_orb

#.....................
# specific nominal orbits
#.....................

nomin_orb = available_nominal_orbits(orbit_area = 'arctic',
                                     verbose = TRUE)
nomin_orb
```

---

available_RGTs *Reference Ground Tracks (RGTs)*

Description

This function returns the url's of all Reference Ground Track (RGT) cycles. Moreover, it returns the dates that belong to each RGT cycle and the names of the RGT cycles.

Usage

```
available_RGTs(
  only_cycle_names = FALSE,
  technical_specs_url = "https://icesat-2.gsfc.nasa.gov/science/specs",
  verbose = FALSE
)
```

Arguments

only_cycle_names	a boolean. If TRUE then only the RGT (Reference Ground Track) cycle names will be returned. Otherwise all orbit files, dates and cycle names.
technical_specs_url	a character string specifying the technical specs website
verbose	a boolean. If TRUE then information will be printed out in the console

Value

a list object with the available orbit files, dates and cycle names

References

https://icesat-2.gsfc.nasa.gov/science/specs

Examples

```
## Not run:

require(IceSat2R)

#.............................................
# all available orbit files, dates and cycle names
#.............................................

avail_dat = available_RGTs(only_cycle_names = FALSE,
                           verbose = TRUE)
avail_dat

#.........................
# receive only the cycle names
#.........................

avail_cycles = available_RGTs(only_cycle_names = TRUE,
                              verbose = TRUE)
avail_cycles


## End(Not run)
```

---

degrees_to_global_grid

Create a global grid based on degrees

Description

This function allows the user to create a degrees grid based on an input bounding box

Usage

```
degrees_to_global_grid(
  minx = -180,
  maxx = 180,
  maxy = 90,
  miny = -90,
  degrees = 1,
  square_geoms = TRUE,
  crs_value = 4326,
  verbose = FALSE
)
```

Arguments

minx	the 'minx' parameter of the bounding box
maxx	the 'maxx' parameter of the bounding box
maxy	the 'maxy' parameter of the bounding box
miny	the 'miny' parameter of the bounding box
degrees	a numeric value specifying the degrees. It defaults to 1.0
square_geoms	a boolean. If FALSE then a hexagonal grid will be created
crs_value	a value. The coordinate reference system of the output grid. The crs must correspond to the EPSG of the input 'minx', 'maxx', 'miny' and 'maxy'. Moreover, the 'EPSG:4326' returns a degrees grid as output.
verbose	a boolean. If TRUE then information will be printed out in the console

Details

The default global input 'minx', 'maxx', 'miny' and 'maxy' correspond to a WKT of "POLYGON ((-180 -90, 180 -90, 180 90, -180 90, -180 -90))" When 'minx', 'maxx', 'miny' and 'maxy' EPSG is lat,long (or 'EPSG:4326') I expect the output grid to be in 'degrees' unit. Using an EPSG other than 'EPSG:4326' might return a different output unit (such as meters). See also the following 'Stackoverflow' thread for more information, https://stackoverflow.com/a/64903836/8302386

Based on 'Approximate Metric Equivalents' 1 degree is approximately 111 km (or 60 nautical miles), Reference: https://www.usna.edu/Users/oceano/pguth/md_help/html/approx_equivalents.htm

Value

a simple features (sf) object

Examples

```
## Not run:

require(IceSat2R)

#.......................................
# 'OpenAltimetry' allows 1x1 degree bounding
```

```
# box selection for the 'atl03' Product
#.....................................

gl_grid_1_d = degrees_to_global_grid(degrees = 1.0, verbose = TRUE)
gl_grid_1_d
# summary(gl_grid_1_d$area)

#.....................................................................
# 'OpenAltimetry' allows 5x5 degree bounding box selection for the following
# Products: 'atl06', 'atl07', 'atl08', 'atl10', 'atl12', 'atl13', 'level3a'
#.....................................................................

gl_grid_5_d = degrees_to_global_grid(degrees = 5.0, verbose = TRUE)
gl_grid_5_d


## End(Not run)
```

`download_file` *Customized function to download files*

Description

Customized function to download files

Usage

```
download_file(url, destfile, download_method, verbose = FALSE)
```

Arguments

`url`	a character string specifying a valid url
`destfile`	a character string specifying a valid path where the output file will be saved
`download_method`	a character string specifying the download method to use. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'. For more information see the documentation of the 'utils::download.file()' function
`verbose`	a boolean. If TRUE then information will appear in the console

References

https://github.com/mlverse/torchdatasets/blob/master/R/utils.R#L20

Examples

```
## Not run:

require(IceSat2R)

# the default timeout is 60 and we set it to 600
options(timeout = 600)

# we specify a URL and a temporary file
default_url = "https://icesat-2.gsfc.nasa.gov/sites/default/"
url_downl = glue::glue('{default_url}files/page_files/IS2RGTscycle12datetime.zip')
tmp_f = tempfile(fileext = '.zip')

# then we download the file
downl_f = download_file(url = url_downl,
                        destfile = tmp_f,
                        download_method = 'curl',
                        verbose = TRUE)

## End(Not run)
```

---

getTracks *Get the ICESAT-2 Tracks*

Description

Get a list of ICESat-2 tracks using a bounding box as input

Usage

```
getTracks(
  minx,
  miny,
  maxx,
  maxy,
  date,
  outputFormat = "csv",
  download_method = "curl",
  verbose = FALSE
)
```

Arguments

minx	the 'minx' parameter of the bounding box
miny	the 'miny' parameter of the bounding box
maxx	the 'maxx' parameter of the bounding box
maxy	the 'maxy' parameter of the bounding box

date	a character string specifying the Data collection date in the format 'yyyy-MM-dd' (such as '2020-01-01')
outputFormat	a character string specifying the output format of the downloaded data. One of 'csv' or 'json'
download_method	a character string specifying the download method to use. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'. For more information see the documentation of the 'utils::download.file()' function
verbose	a boolean. If TRUE then information will be printed out in the console

Value

either a data.table (if outputFormat is 'csv') or a nested list (if outputFormat is 'json')

References

https://openaltimetry.earthdatacloud.nasa.gov

https://nsidc.org/data/icesat-2

Examples

```
## Not run:

require(IceSat2R)
require(magrittr)
require(sf)

sf::sf_use_s2(use_s2 = FALSE)

#.....................................................
# In case we are interested to find the ICESat-2 tracks for
#     - a specific area (or bounding box)
#     - a specific time interval (close to September 2021)
#.....................................................

#.............
# Ice sheet area
#.............

data(ne_10m_glaciated_areas)

ice_sheet = 'Kluane Ice Cap'

ice_sheet_geom = ne_10m_glaciated_areas %>%
  subset(!is.na(name)) %>%
  subset(name == ice_sheet)

ice_sheet_geom

#............
# time interval
```

```
#.............

approx_date_start = "2021-09-05"
approx_date_end = "2021-09-20"

res_rgt_many = time_specific_orbits(date_from = approx_date_start,
                                    date_to = approx_date_end,
                                    RGT_cycle = NULL,
                                    download_method = 'curl',
                                    threads = parallel::detectCores(),
                                    verbose = TRUE)

res_rgt_many

# table(as.Date(res_rgt_many$Date_time))

#........................................
# create the bounding box of the ice
# sheet so that I use the same area in the
# 'st_intersects()' and 'getTracks()' functions
# ('getTracks()' takes a bounding box as input)
#........................................

bbx_ice_sheet = sf::st_bbox(obj = ice_sheet_geom)
sf_obj_ice_sheet = sf::st_as_sfc(bbx_ice_sheet)

#....................................
# intersection of the Ice Sheet with
# the tracks for the specified time period
#....................................

res_inters = sf::st_intersects(x = sf::st_geometry(sf_obj_ice_sheet),
                               y = sf::st_geometry(res_rgt_many),
                               sparse = TRUE)

#....................................
# one or more intersected RGT's for the area
#....................................

df_inters = data.frame(res_inters)
rgt_subs = res_rgt_many[df_inters$col.id, , drop = FALSE]


#................................................
# Keep the Date and the bounding box to
# verify with the 'getTracks()' function
# an alternative is to use the "verify_RGTs()" function
#................................................

for (item in 1:nrow(rgt_subs)) {

  dat_item = rgt_subs[item, , drop = F]
  Date = as.Date(dat_item$Date_time)
```

```
  op_tra = getTracks(minx = as.numeric(bbx_ice_sheet['xmin']),
                               miny = as.numeric(bbx_ice_sheet['ymin']),
                               maxx = as.numeric(bbx_ice_sheet['xmax']),
                               maxy = as.numeric(bbx_ice_sheet['ymax']),
                               date = as.character(Date),
                               outputFormat = 'csv',
                               download_method = 'curl',
                               verbose = FALSE)

  date_obj = dat_item$Date_time
  tim_rgt = glue::glue("Date: {date_obj} Time specific RGT: '{dat_item$RGT}'")

  if (nrow(op_tra) > 0) {
    iter_op_trac = paste(op_tra$track, collapse = ', ')
    cat(glue::glue("{tim_rgt}  OpenAltimetry: '{iter_op_trac}'"), '\n')
  }
  else {
    cat(glue::glue("{tim_rgt} without an OpenAltimetry match!"), '\n')
  }
}


## End(Not run)
```

---

get_atlas_data *Get ICESat-2 ATLAS data for a specific Date*

Description

This function allows the user to download ICESat-2 ATLAS Product data for a specific date, bounding box, track and beam.

Usage

```
get_atlas_data(
  minx,
  miny,
  maxx,
  maxy,
  date,
  trackId,
  beamName = NULL,
  product = "atl03",
  client = "portal",
  photonConfidence = NULL,
  sampling = FALSE,
  outputFormat = "csv",
  file_path_zip = NULL,
```

```
  download_method = "curl",
  verbose = FALSE
)
```

Arguments

minx	the 'minx' parameter of the bounding box
miny	the 'miny' parameter of the bounding box
maxx	the 'maxx' parameter of the bounding box
maxy	the 'maxy' parameter of the bounding box
date	a character string specifying the Data collection date in the format 'yyyy-MM-dd' (such as '2020-01-01')
trackId	an integer specifying the 'Reference ground track ID' (see the examples section on how to come to the 'trackId' and to a bounding box of 1 or 5 degrees globally)
beamName	either NULL (if the 'product' parameter is not equal to 'atl03' which means data for all 6 beams will be returned) or a character string specifying the Beam Name, which can be one of the 'gt1l', 'gt1r', 'gt2l', 'gt2r', 'gt3l' or 'gt3r'
product	a character string specifying the input data product (default value is 'atl03'). It can be one of 'atl03', 'atl06', 'atl07', 'atl08', 'atl10', 'atl12' or 'atl13'
client	a character string specifying the 'Referring client'. Can be one of 'portal' or 'jupyter' (default is 'portal')
photonConfidence	either NULL or a character vector specifying the 'Confidence levels of the photons (can be one or more). If NULL (default) all photon data will be returned. The available options are 'na', 'noise', 'buffer', 'low', 'medium', 'high'
sampling	a boolean. If TRUE a sampling rate of 1/1000 will be used, otherwise all data will be returned (default is FALSE)
outputFormat	a character string specifying the output format of the downloaded data. One of 'csv', 'json' or 'zip'
file_path_zip	either NULL or a character string specifying a valid path to the output .zip file. This parameter will normally be a valid path if the 'outputFormat' parameter is set to 'zip'. If it's NULL and the 'outputFormat' parameter is 'zip' then the downloaded '.zip' file will be converted and returned as a data.table object
download_method	a character string specifying the download method to use. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'. For more information see the documentation of the 'utils::download.file()' function
verbose	a boolean. If TRUE then information will be printed out in the console

Details

'atl03', *Global Geolocated Photon Data* (Version 4, Requests are limited to 1x1 degree spatial bounding box selection

This data set *(ATL03)* contains height above the WGS 84 ellipsoid (ITRF2014 reference frame), latitude, longitude, and time for all photons downlinked by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation

Satellite-2 (ICESat-2) observatory. The ATL03 product was designed to be a single source for all photon data and ancillary information needed by higher-level ATLAS/ICESat-2 products. As such, it also includes spacecraft and instrument parameters and ancillary data not explicitly required for ATL03

'atl06', *Land Ice Height* (Version 4, Requests are limited to 5x5 degree spatial bounding box selection)
This data set *(ATL06)* provides geolocated, land-ice surface heights (above the WGS 84 ellipsoid, ITRF2014 reference frame), plus ancillary parameters that can be used to interpret and assess the quality of the height estimates. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl07', *Sea Ice Height* (Version 4, Requests are limited to 5x5 degree spatial bounding box selection)
The data set *(ATL07)* contains along-track heights for sea ice and open water leads (at varying length scales) relative to the WGS84 ellipsoid (ITRF2014 reference frame) after adjustment for geoidal and tidal variations, and inverted barometer effects. Height statistics and apparent reflectance are also provided. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl08', *Land and Vegetation Height* (Version 4, Requests are limited to 5x5 degree spatial bounding box selection)
This data set *(ATL08)* contains along-track heights above the WGS84 ellipsoid (ITRF2014 reference frame) for the ground and canopy surfaces. The canopy and ground surfaces are processed in fixed 100 m data segments, which typically contain more than 100 signal photons. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl10', *Sea Ice Freeboard* (Version 4, Requests are limited to 5x5 degree spatial bounding box selection)
This data set *(ATL10)* contains estimates of sea ice freeboard, calculated using three different approaches. Sea ice leads used to establish the reference sea surface and descriptive statistics used in the height estimates are also provided. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl12', *Ocean Surface Height* (Version 4, Requests are limited to 5x5 degree spatial bounding box selection)
This data set *(ATL12)* contains along-track sea surface heights at variable length scales over cloud-free regions. Estimates of height distributions, surface roughness, surface slope, and apparent reflectance are also provided. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl13', *Inland Water Surface Height* (Version 4, Requests are limited to 5x5 degree spatial bounding box selection)
This data set *(ATL13)* contains along-track water surface heights and descriptive statistics for inland water bodies. Water bodies include lakes, reservoirs, bays, and estuaries. Descriptive statistics include along-track surface slope (where data permit), mean and standard deviation, subsurface signal (532 nm) attenuation, wave height, and coarse depth to bottom topography

Value

if the 'file_path_zip' parameter is NULL it returns either a data.table (if outputFormat is 'csv') or a nested list (if outputFormat is 'json') else the file path where the .zip file is saved. In case that the 'outputFormat' is set to 'zip' and the 'file_path_zip' parameter to NULL then a data.table will be returned.

References

https://openaltimetry.earthdatacloud.nasa.gov

https://nsidc.org/data/icesat-2

Examples

```
## Not run:

require(IceSat2R)

#.........................................................
# observe the available countries and continents using the
# 'rnaturalearth' R package to perform a query using map-edit
#.........................................................

cntr_cnt = rnaturalearth::ne_countries(scale = 110,
                                       type = 'countries',
                                       returnclass = 'sf')

cntr_cnt = cntr_cnt[, c('sovereignt', 'continent')]

# sort(cntr_cnt$sovereignt)
# sort(unique(cntr_cnt$continent))


#.........................
# Select a 'continent' as AOI   (5-degree query)
#.........................

init = select_aoi_global_grid$new(area_of_interest = 'Oceania',
                                  verbose = TRUE)

init$draw_edit_aoi(degrees = 5.0, square_geoms = TRUE)
sf_obj = init$selected_areas_global_grid(plot_data = FALSE)
sf_obj

#.................................................
# I drew a bounding box close to 'Mount Hagen Papua'
# inside one of the 5-degreee cells (after zooming-in)
#.................................................

# mapview::mapview(sf_obj, legend = F)
# sf::st_as_text(sf::st_geometry(sf_obj))

#.........................................
# to reproduce the results without selecting
# an area based on the 'draw_edit_aoi()' function
#.........................................

# plg = "POLYGON ((140 -6.641235, 145 -6.641235, 145 -1.641235, 140 -1.641235, 140 -6.641235))"
# sf_obj = sf::st_as_sfc(plg, crs = 4326)
```

```
#.................................................
# first we find available ICESat-2 track-ID's
# and Dates (time interval) from a specific RGT cycle
#.................................................

approx_date_start = "2021-02-01"
approx_date_end = "2021-02-15"
# 'RGT_cycle_10'

res_rgt_many = time_specific_orbits(date_from = approx_date_start,
                                    date_to = approx_date_end,
                                    RGT_cycle = NULL,
                                    download_method = 'curl',
                                    threads = parallel::detectCores(),
                                    verbose = TRUE)
res_rgt_many

#.....................................................
# then we create the bounding box of the selected area
# and proceed to the intersection with the computed RGT's
# ( include the bounding box for reproducibility )
#.....................................................

bbx_aoi = sf::st_bbox(obj = sf_obj)
# c(xmin = 140, ymin = -6.641235, xmax = 145, ymax = -1.641235)

sf_obj_bbx = sf::st_as_sfc(bbx_aoi)

res_inters = sf::st_intersects(x = sf_obj_bbx,
                               y = sf::st_geometry(res_rgt_many),
                               sparse = TRUE)
#.....................
# matched (RGT) tracks
#.....................

df_inters = data.frame(res_inters)

if (nrow(df_inters) == 0) {
  stop("There is no intersection between the specified AOI and the RGTs!")
}


rgt_subs = res_rgt_many[df_inters$col.id, , drop = FALSE]
rgt_subs


#.....................................
# find out which of the time specific RGT's
# match the OpenAltimetry Track-ID's
#.....................................

dtbl_rgts = verify_RGTs(nsidc_rgts = rgt_subs,
```

```
                        bbx_aoi = bbx_aoi,
                        verbose = TRUE)

  #................................
  # we will iterate over the available:
  #    - Dates
  #    - trackId's
  #    - product's
  # to gather the up to 5-degree data
  #................................

  dates_iters = unique(dtbl_rgts$Date_time)
  RGTs_iters = unique(dtbl_rgts$RGT_NSIDC)
  prods_5_degrs = c('atl06', 'atl07', 'atl08', 'atl10', 'atl12', 'atl13')

  dat_out = logs_out = list()

  for (idx in seq_along(dates_iters)) {

    date_i = dates_iters[idx]
    rgt_i = RGTs_iters[idx]

    for (prod_i in prods_5_degrs) {

      name_iter = glue::glue("{date_i}_{rgt_i}_{prod_i}")
      cat(glue::glue("Date: '{date_i}'  RGT: '{rgt_i}'  Product: '{prod_i}'"), '\n')

      iter_dat = get_atlas_data(minx = as.numeric(bbx_aoi['xmin']),
                                miny = as.numeric(bbx_aoi['ymin']),
                                maxx = as.numeric(bbx_aoi['xmax']),
                                maxy = as.numeric(bbx_aoi['ymax']),
                                date = date_i,
                                trackId = rgt_i,
                                product = prod_i,
                                client = 'portal',
                                outputFormat = 'csv',
                                verbose = FALSE)

      iter_logs = list(Date = date_i,
                       RGT = rgt_i,
                       Product = prod_i,
                       N_rows = nrow(iter_dat))

      logs_out[[name_iter]] = data.table::setDT(iter_logs)
      dat_out[[name_iter]] = iter_dat
    }
  }

  #....................................
  # each sublist corresponds to a different
  # parameter setting (Date, Track, Product)
  #....................................
```

```
dat_out

#.....
# Logs  (including the number of rows for each parameter setting)
#.....

dtbl_logs = data.table::rbindlist(logs_out)
dtbl_logs = subset(dtbl_logs, N_rows > 0)
dtbl_logs = dtbl_logs[order(dtbl_logs$N_rows, decreasing = T), ]
dtbl_logs

#.........................................
# The Products 'atl08' and 'atl13' have data
# for the Dates and RGT's of the selected area
#.........................................

unique(dtbl_logs$Product)
# c("atl06" "atl08" "atl13" "atl12")

#.............
# RGT's with data
#.............

unique(dtbl_logs$RGT)
# c(627, 756, 688, 619, 817)

#.............
# Dates with Data
#.............

unique(dtbl_logs$Date)
# c("2021-02-03", "2021-02-11", "2021-02-07", "2021-02-02", "2021-02-15")


#============
# ATL03 Data
#============

.........................................
# the default timeout is 60 and we set it to 600
#.........................................

options(timeout = 600)
print(getOption('timeout'))


# we skip the interactive selection of the 1 degree grid
plg = "POLYGON ((142 -34.64124, 143 -34.64124, 143 -33.64124, 142 -33.64124, 142 -34.64124))"
sf_obj_atl03 = sf::st_as_sfc(plg, crs = 4326)

# we update the bounding box based on the AOI
bbx_aoi = sf::st_bbox(obj = sf_obj_atl03)
# c(xmin = 142, ymin = -34.64124, xmax = 143, ymax = -33.64124)
```

```
sf_obj_bbx = sf::st_as_sfc(bbx_aoi)

res_inters = sf::st_intersects(x = sf_obj_bbx,
                               y = sf::st_geometry(res_rgt_many),
                               sparse = TRUE)

df_inters = data.frame(res_inters)

if (nrow(df_inters) == 0) {
  stop("There is no intersection between the specified AOI and the RGTs!")
}


rgt_subs = res_rgt_many[df_inters$col.id, , drop = FALSE]
rgt_subs

dtbl_rgts = verify_RGTs(nsidc_rgts = rgt_subs,
                        bbx_aoi = bbx_aoi,
                        verbose = TRUE)

comp_cas = complete.cases(dtbl_rgts)
dtbl_rgts = dtbl_rgts[comp_cas, , drop = F]


dates_iters = unique(dtbl_rgts$Date_time)
RGTs_iters = unique(dtbl_rgts$RGT_NSIDC)
prods_1_degrs = c('atl03')

#.........................
# use all beams with 'atl03'
#.........................

dat_out = logs_out = list()

for (idx in seq_along(dates_iters)) {

  date_i = dates_iters[idx]
  rgt_i = RGTs_iters[idx]

  name_iter = glue::glue("{date_i}_{rgt_i}")
  cat(glue::glue("Date: '{date_i}'  RGT: '{rgt_i}'"), '\n')

  iter_dat = get_atlas_data(minx = as.numeric(bbx_aoi['xmin']),
                            miny = as.numeric(bbx_aoi['ymin']),
                            maxx = as.numeric(bbx_aoi['xmax']),
                            maxy = as.numeric(bbx_aoi['ymax']),
                            date = date_i,
                            beamName = NULL,
                            trackId = rgt_i,
                            product = prods_1_degrs,
                            client = 'portal',
                            outputFormat = 'csv',
```

```
                            verbose = FALSE)

  iter_logs = list(Date = date_i,
                   RGT = rgt_i,
                   Product = prods_1_degrs,
                   N_rows = nrow(iter_dat))

  logs_out[[name_iter]] = data.table::setDT(iter_logs)
  dat_out[[name_iter]] = iter_dat
}

dat_out
unique(dtbl_logs$Product)
unique(dat_out[[1]]$beam)
unique(dat_out[[1]]$`confidence code`)

#.............................................
# use 2 beams with 'atl03' (a vector of characters)
#.............................................

dat_out = logs_out = list()

for (idx in seq_along(dates_iters)) {

  date_i = dates_iters[idx]
  rgt_i = RGTs_iters[idx]

  name_iter = glue::glue("{date_i}_{rgt_i}")
  cat(glue::glue("Date: '{date_i}'  RGT: '{rgt_i}'"), '\n')

  iter_dat = get_atlas_data(minx = as.numeric(bbx_aoi['xmin']),
                            miny = as.numeric(bbx_aoi['ymin']),
                            maxx = as.numeric(bbx_aoi['xmax']),
                            maxy = as.numeric(bbx_aoi['ymax']),
                            date = date_i,
                            beamName = c("gt1l", "gt1r"),
                            trackId = rgt_i,
                            product = prods_1_degrs,
                            client = 'portal',
                            outputFormat = 'csv',
                            verbose = FALSE)

  iter_logs = list(Date = date_i,
                   RGT = rgt_i,
                   Product = prods_1_degrs,
                   N_rows = nrow(iter_dat))

  logs_out[[name_iter]] = data.table::setDT(iter_logs)
  dat_out[[name_iter]] = iter_dat
}

dat_out
unique(dtbl_logs$Product)
```

```
unique(dat_out[[1]]$beam)


#.........................................
# use 1 beam with 'atl03' (a character string)
#.........................................

dat_out = logs_out = list()

for (idx in seq_along(dates_iters)) {

  date_i = dates_iters[idx]
  rgt_i = RGTs_iters[idx]

  name_iter = glue::glue("{date_i}_{rgt_i}")
  cat(glue::glue("Date: '{date_i}'  RGT: '{rgt_i}'"), '\n')

  iter_dat = get_atlas_data(minx = as.numeric(bbx_aoi['xmin']),
                            miny = as.numeric(bbx_aoi['ymin']),
                            maxx = as.numeric(bbx_aoi['xmax']),
                            maxy = as.numeric(bbx_aoi['ymax']),
                            date = date_i,
                            beamName = "gt1l",
                            trackId = rgt_i,
                            product = prods_1_degrs,
                            client = 'portal',
                            outputFormat = 'csv',
                            verbose = FALSE)

  iter_logs = list(Date = date_i,
                   RGT = rgt_i,
                   Product = prods_1_degrs,
                   N_rows = nrow(iter_dat))

  logs_out[[name_iter]] = data.table::setDT(iter_logs)
  dat_out[[name_iter]] = iter_dat
}

dat_out
unique(dtbl_logs$Product)
unique(dat_out[[1]]$beam)


#.........................................
# use all beams with 'atl03' & photonConfidence
#.........................................

dat_out = logs_out = list()

for (idx in seq_along(dates_iters)) {

  date_i = dates_iters[idx]
  rgt_i = RGTs_iters[idx]
```

```
  name_iter = glue::glue("{date_i}_{rgt_i}")
  cat(glue::glue("Date: '{date_i}'  RGT: '{rgt_i}'"), '\n')

  iter_dat = get_atlas_data(minx = as.numeric(bbx_aoi['xmin']),
                            miny = as.numeric(bbx_aoi['ymin']),
                            maxx = as.numeric(bbx_aoi['xmax']),
                            maxy = as.numeric(bbx_aoi['ymax']),
                            date = date_i,
                            beamName = NULL,
                            photonConfidence = c('buffer', 'low'),
                            trackId = rgt_i,
                            product = prods_1_degrs,
                            client = 'portal',
                            outputFormat = 'csv',
                            verbose = FALSE)

  iter_logs = list(Date = date_i,
                   RGT = rgt_i,
                   Product = prods_1_degrs,
                   N_rows = nrow(iter_dat))

  logs_out[[name_iter]] = data.table::setDT(iter_logs)
  dat_out[[name_iter]] = iter_dat
}

dat_out
unique(dtbl_logs$Product)
unique(dat_out[[1]]$beam)
unique(dat_out[[1]]$`confidence code`)


## End(Not run)
```

`get_level3a_data` *Get IceSat-2 ATLAS 'Level-3A' data for a time interval (up to 1 year)*

Description

This function allows the user to download IceSat-2 ATLAS 'Level-3A' data for a specific time interval, bounding box, track and beam.

Usage

```
get_level3a_data(
  minx,
  miny,
  maxx,
  maxy,
  startDate,
```

```
  endDate,
  trackId,
  beamName = NULL,
  product = "atl08",
  client = "portal",
  outputFormat = "csv",
  file_path_zip = NULL,
  download_method = "curl",
  verbose = FALSE
)
```

Arguments

minx	the 'minx' parameter of the bounding box
miny	the 'miny' parameter of the bounding box
maxx	the 'maxx' parameter of the bounding box
maxy	the 'maxy' parameter of the bounding box
startDate	a character string specifying the Data collection of the *start* Date in the format 'yyyy-MM-dd' (such as '2020-01-01')
endDate	a character string specifying the Data collection of the *end* Date in the format 'yyyy-MM-dd' (such as '2020-01-01')
trackId	an integer specifying the 'Reference ground track ID' (see the examples section on how to come to the 'trackId' and to a bounding box of 1 or 5 degrees globally)
beamName	either NULL (data for all 6 beams will be returned) or a character vector specifying the Beam Name(s), It can be one or more of the 'gt1l', 'gt1r', 'gt2l', 'gt2r', 'gt3l' or 'gt3r'
product	a character string specifying the input data product (default value is 'atl08'). It can be one of 'atl06', 'atl07', 'atl08', 'atl10', 'atl12' or 'atl13'
client	a character string specifying the 'Referring client'. Can be one of 'portal' or 'jupyter' (default is 'portal')
outputFormat	a character string specifying the output format of the downloaded data. One of 'csv', 'json' or 'zip'
file_path_zip	either NULL or a character string specifying a valid path to the output .zip file. This parameter will normally be a valid path if the 'outputFormat' parameter is set to 'zip'. If it's NULL and the 'outputFormat' parameter is 'zip' then the downloaded '.zip' file will be converted and returned as a data.table object
download_method	a character string specifying the download method to use. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'. For more information see the documentation of the 'utils::download.file()' function
verbose	a boolean. If TRUE then information will be printed out in the console

Details

Up to 1 year worth of ICESat-2 *Level-3A* product data can be downloaded. *Note:* requests are limited to 5x5 degree spatial bounding box selection

'atl06', *Land Ice Height* (Version 4) This data set *(ATL06)* provides geolocated, land-ice surface heights (above the WGS 84 ellipsoid, ITRF2014 reference frame), plus ancillary parameters that can be used to interpret and assess the quality of the height estimates. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl07', *Sea Ice Height* (Version 4) The data set *(ATL07)* contains along-track heights for sea ice and open water leads (at varying length scales) relative to the WGS84 ellipsoid (ITRF2014 reference frame) after adjustment for geoidal and tidal variations, and inverted barometer effects. Height statistics and apparent reflectance are also provided. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl08', *Land and Vegetation Height* (Version 4) This data set *(ATL08)* contains along-track heights above the WGS84 ellipsoid (ITRF2014 reference frame) for the ground and canopy surfaces. The canopy and ground surfaces are processed in fixed 100 m data segments, which typically contain more than 100 signal photons. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl10', *Sea Ice Freeboard* (Version 4) This data set *(ATL10)* contains estimates of sea ice freeboard, calculated using three different approaches. Sea ice leads used to establish the reference sea surface and descriptive statistics used in the height estimates are also provided. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl12', *Ocean Surface Height* (Version 4) This data set *(ATL12)* contains along-track sea surface heights at variable length scales over cloud-free regions. Estimates of height distributions, surface roughness, surface slope, and apparent reflectance are also provided. The data were acquired by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory

'atl13', *Inland Water Surface Height* (Version 4) This data set *(ATL13)* contains along-track water surface heights and descriptive statistics for inland water bodies. Water bodies include lakes, reservoirs, bays, and estuaries. Descriptive statistics include along-track surface slope (where data permit), mean and standard deviation, subsurface signal (532 nm) attenuation, wave height, and coarse depth to bottom topography

Value

if the 'file_path_zip' parameter is NULL it returns either a data.table (if outputFormat is 'csv') or a nested list (if outputFormat is 'json') else the file path where the .zip file is saved. In case that the 'outputFormat' is set to 'zip' and the 'file_path_zip' parameter to NULL then a data.table will be returned.

References

https://openaltimetry.earthdatacloud.nasa.gov

https://nsidc.org/data/icesat-2

Examples

```
## Not run:
```

```
require(IceSat2R)

#.....................................................
# parameter setting based on the output results from the
# 'examples' section of the 'get_atlas_data()' function
# so that we can iterate over the RGT's and Products for
# the specified time interval
#.....................................................

bbx = c(xmin = 140, ymin = -6.641235, xmax = 145, ymax = -1.641235)
start_date = "2021-02-03"
end_date = "2021-02-15"
RGTs = c(627, 756, 688, 619, 817)
Products = c('atl08', 'atl13')

#.....................................................
# loop over the specified parameters and save the results
#.....................................................

dat_out = logs_out = list()

for (prod_i in Products) {
  for (track_i in RGTs) {

    name_iter = glue::glue("{track_i}_{prod_i}")
    cat(glue::glue("RGT: '{track_i}'  Product: '{prod_i}'"), '\n')

    iter_dat = get_level3a_data(minx = as.numeric(bbx['xmin']),
                                miny = as.numeric(bbx['ymin']),
                                maxx = as.numeric(bbx['xmax']),
                                maxy = as.numeric(bbx['ymax']),
                                startDate = start_date,
                                endDate = end_date,
                                trackId = track_i,
                                beamName = NULL,        # return data of all 6 beams
                                product = prod_i,
                                client = 'portal',
                                outputFormat = 'csv',
                                verbose = FALSE)

    iter_logs = list(RGT = track_i,
                     Product = prod_i,
                     N_rows = nrow(iter_dat))

    logs_out[[name_iter]] = data.table::setDT(iter_logs)
    dat_out[[name_iter]] = iter_dat
  }
}


#....................................
# each sublist corresponds to a different
```

```
# parameter setting (Track, Product)
#.....................................

dat_out

#.....
# Logs  (including the number of rows for each parameter setting)
#.....

dtbl_logs = data.table::rbindlist(logs_out)
dtbl_logs = subset(dtbl_logs, N_rows > 0)
dtbl_logs = dtbl_logs[order(dtbl_logs$N_rows, decreasing = T), ]
dtbl_logs


## End(Not run)
```

IceSat2R *The "IceSat2R" package*

Description

An important aspect of the "IceSat2R" (https://cran.r-project.org/web/packages/IceSat2R/index.html) package is that it includes the code, documentation, and examples so that users can retrieve, process, and analyze data based on specific workflows.

Details

For instance,

- A user can select an "area of interest" (AOI) either programmatically or interactively
- If the "Reference Ground Track" (RGT) is not known, the user has the option to utilize either
  1. one of the "overall_mission_orbits()" or "time_specific_orbits()" to compute the RGT(s) for a pre-specified global area or for a time period, or
  2. one of the "vsi_nominal_orbits_wkt()" or "vsi_time_specific_orbits_wkt()" to compute the RGT(s) for a specific AOI
- Once the RGT is computed it can be verified with the "getTracks()" function of the "OpenAltimetry Web API" (https://openaltimetry.earthdatacloud.nasa.gov)
- Finally the user can utilize one of the "get_atlas_data()" or "get_level3a_data()" functions to retrieve the data for specific product(s), Date(s) and Beam(s)

This work-flow is illustrated also in a diagram

`latest_orbits` *Extraction of the url from the Technical Specification Website*

Description

This function allows the user to view the latest 'Nominal' and 'Time Specfic' orbit metadata (Url, Reference Ground Track Names, Dates and Types)

Usage

```
latest_orbits(
  technical_specs_url = "https://icesat-2.gsfc.nasa.gov/science/specs",
  verbose = FALSE
)
```

Arguments

`technical_specs_url`	a character string specifying the technical specs website
`verbose`	a boolean. If TRUE then information will be printed out in the console

Value

a 'data.table' object

References

https://icesat-2.gsfc.nasa.gov/science/specs

Examples

```
## Not run:

require(IceSat2R)

orbs = latest_orbits(verbose = TRUE)
orbs


## End(Not run)
```

ne_10m_glaciated_areas

Natural Earth 10m Glaciated Areas (1:10 million scale)

Description

Polygons derived from DCW (Digital Chart of the World), except for Antarctica derived from MOA. Includes name attributes for major polar glaciers. Filtering has aggregated some minor glaciers and eliminated others from the original DCW data.

Usage

```
data(ne_10m_glaciated_areas)
```

Format

An object of class `sf` (inherits from `data.frame`) with 68 rows and 6 columns.

Details

Issues: Needs scale rank attributes

Note: The original DCW data was collected decades ago. Since then many mountain glaciers have either disappeared or diminished in size. This data theme is deliberately called "glaciated areas" instead of "glaciers" to reflect the changeable character of glacier extents.

References

https://www.naturalearthdata.com/downloads/10m-physical-vectors/10m-glaciated-areas/

Examples

```
require(IceSat2R)
require(sf)

data(ne_10m_glaciated_areas)
```

overall_mission_orbits

Overall Mission Orbits

Description

This function allows the user to view information of the nominal mission orbits and beam locations: "The processed files have 7 tracks per orbit: one for each of the six beams of ICESat-2, and the seventh for the Reference Ground Track (RGT). The RGT is an imaginary line through the six-beam pattern that is handy for getting a sense of where the orbits fall on Earth, and which the mission uses to point the observatory. However, the six tracks for the six beams are our best estimate of where the beams will actually fall on Earth's surface."

Usage

```
overall_mission_orbits(
  orbit_area,
  download_method = "curl",
  threads = 1,
  verbose = FALSE
)
```

Arguments

orbit_area	a character string specifying the nominal mission orbits and beam locations. It can be one of 'antarctic', 'arctic', 'western_hemisphere' or 'eastern_hemisphere'
download_method	a character string specifying the download method. Corresponds to the 'method' parameter of the 'utils::download.file()' function. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'
threads	an integer that specifies the number of threads to use in parallel when processing the data
verbose	a boolean. If TRUE then information will be printed out in the console

Value

an 'sf' object of multiple tracks (see the 'LAYER' column of the output object)

References

https://icesat-2.gsfc.nasa.gov/science/specs

Examples

```
## Not run:

require(IceSat2R)

res_orb  = overall_mission_orbits(orbit_area = 'antarctic',
                                  download_method = 'curl',
                                  threads = 1,
                                  verbose = TRUE)
str(res_orb)

## End(Not run)
```

`revisit_time_RGTs` *Revisit Time Reference Ground Tracks and Dates*

Description

This function shows the information of the 'available_RGTs' function and additionally it returns the .zip (kmz files) of all laser tracks over each 91-day repeat period (revisit time). Note that the locations and times are estimates, but should be correct to within a few minutes in time and better than 100m in the predicted locations.

Usage

```
revisit_time_RGTs(RGT_cycle = NULL, complete_date_sequence = FALSE)
```

Arguments

`RGT_cycle`	either NULL or a character string specifying a single RGT (Reference Ground Track) as determined by the output of the 'available_RGTs(only_cycle_names = TRUE)' function. If NULL then all available Data will be returned.
`complete_date_sequence`	a boolean. If TRUE a complete sequence of Dates will be returned, otherwise only the 'minimum' and 'maximum' Dates.

Details

ICESat-2 was in safe-hold from June 26 through July 9, 2019. ATLAS was off during this time, so data was not collected or pointed to the reference ground track.

Value

a list object with the available orbit files, dates and date sequence lengths

References

https://icesat-2.gsfc.nasa.gov/science/specs

Examples

```
require(IceSat2R)

#.....................................................
# receive all orbit files, dates and length of sequences
#.....................................................

rev_all = revisit_time_RGTs(RGT_cycle = NULL, complete_date_sequence = TRUE)
rev_all

#.............................................
```

```
# observe and choose one of the available RGT-cycles
#.............................................

avail_cycles = available_RGTs(only_cycle_names = TRUE,
                              verbose = TRUE)
avail_cycles

#.................................
# receive results for a specific cycle
#.................................

rev_cycle = revisit_time_RGTs(RGT_cycle = 'RGT_cycle_1', complete_date_sequence = FALSE)
rev_cycle
```

RGT_cycle_14 *Reference Ground Tracks (RGTs) for IceSat-2 Cycle 14*

Description

The data includes the following columns: "longitude", "latitude", "day_of_year", "Date", "hour", "minute", "second" and "RGT". The "RGT" column consists of 1387 Reference Ground Tracks (RGTs) for the IceSat-2 Cycle 14 (from 'December 22, 2021' to 'March 23, 2022')

Usage

```
data(RGT_cycle_14)
```

Format

An object of class `data.table` (inherits from `data.frame`) with 131765 rows and 8 columns.

Details

The following code snippet shows how to come to the "RGT_cycle_14" data. The same can be done with any of the available RGT Cycles. For the following code I utilized 8 threads to speed up the pre-processing of the downloaded .kml files (the code takes approximately 15 minutes on my Linux Personal Computer),

```
require(IceSat2R)
require(magrittr)
require(sf)


avail_cycles = available_RGTs(only_cycle_names = TRUE)
avail_cycles


idx_cycle = 14
```

```
choose_cycle = avail_cycles[idx_cycle]
choose_cycle

res_rgt_many = time_specific_orbits(RGT_cycle = choose_cycle, download_method = 'curl',
threads = parallel::detectCores(), verbose = TRUE)

RGT_cycle_14 = sf::st_coordinates(res_rgt_many)
colnames(RGT_cycle_14) = c('longitude', 'latitude')
RGT_cycle_14 = data.table::data.table(RGT_cycle_14)
RGT_cycle_14$day_of_year = res_rgt_many$day_of_year
RGT_cycle_14$Date = as.Date(res_rgt_many$Date_time)
RGT_cycle_14$hour = lubridate::hour(res_rgt_many$Date_time)
RGT_cycle_14$minute = lubridate::minute(res_rgt_many$Date_time)
RGT_cycle_14$second = lubridate::second(res_rgt_many$Date_time)
RGT_cycle_14$RGT = res_rgt_many$RGT
RGT_cycle_14
```

References

https://icesat-2.gsfc.nasa.gov/science/specs

Examples

```
require(IceSat2R)
require(data.table)

data(RGT_cycle_14)
```

`select_aoi_global_grid`

R6 Class to Select an Area of Interest (AOI) from a Global Grid

---

Description

The 'select_aoi_global_grid' class allows the user to select an Area of Interest (AOI) (see the examples section for all available options)

Usage

```
# init <- select_aoi_global_grid$new()
```

Public fields

`area_of_interest` an R object (character string, vector)

`leaflet_provider_base` a leaflet provider object

`leaflet_provider_secondary` a leaflet provider object

`crs_value` a numeric value

`use_s2` a boolean

`verbose` a boolean

Methods

Public methods:

- `select_aoi_global_grid$new()`
- `select_aoi_global_grid$draw_edit_internal()`
- `select_aoi_global_grid$draw_edit_aoi()`
- `select_aoi_global_grid$selected_areas_global_grid()`
- `select_aoi_global_grid$selected_aoi_sf()`
- `select_aoi_global_grid$clone()`

Method `new()`**:** Initialization method for the 'select_aoi_global_grid' R6 class

Usage:

```
select_aoi_global_grid$new(
  area_of_interest = NULL,
  leaflet_provider_base = leaflet::providers$CartoDB.Positron,
  leaflet_provider_secondary = leaflet::providers$Esri.WorldImagery,
  crs_value = 4326,
  use_s2 = FALSE,
  verbose = FALSE
)
```

Arguments:

`area_of_interest` either NULL (which allows the user to draw the area of interest on the map) or a character string (i.e. a 'country' or a 'continent') or a named bounding box vector (such as c(xmin = 16.1, xmax = 16.6, ymax = 48.6, ymin = 47.9)). The 'countries' and 'continents' can be extracted from the "rnaturalearth::ne_countries(scale = 110, type = 'countries', returnclass = 'sf')" function and specifically the columns: 'sovereignt' and 'continent'

`leaflet_provider_base` a leaflet provider object

`leaflet_provider_secondary` a leaflet provider object

`crs_value` a value. The coordinate reference system. The default value of the crs is 4326

`use_s2` a boolean. If TRUE, use the s2 spherical geometry package for geographical coordinate operations (see the documentation of the 'sf::sf_use_s2()' function for more information)

`verbose` a boolean. If TRUE then information will be printed in the console

Method `draw_edit_internal()`**:** Internal Shiny application to visualize the selected area

Usage:

```
select_aoi_global_grid$draw_edit_internal(lft_map)
```

Arguments:

`lft_map` a leaflet map

Method `draw_edit_aoi()`**:** Allows to view the Global Grid on the map including a popup that shows the Area of each grid rectangle (or hexagon). The user can then select an Area of Interest (AOI)

Usage:

```
select_aoi_global_grid$draw_edit_aoi(degrees = 1, square_geoms = TRUE)
```

Arguments:

`degrees` a numeric value. This can be either 1.0 or 5.0 to allow queries to the 'OpenAltimetry' API

`square_geoms` a boolean. If FALSE then a hexagonal grid will be created

Method `selected_areas_global_grid()`**:** Takes the selected Area(s) of Interest (AOI) from the draw_edit_aoi() method and returns a simple features object with the corresponding n-degree grid cells

Usage:

```
select_aoi_global_grid$selected_areas_global_grid(plot_data = FALSE)
```

Arguments:

`plot_data` a boolean specifying if the selected from the user AOI's and the corresponding grid cells should be plotted side by side. If FALSE then only the simple features object will be returned. If TRUE and the initial 'area_of_interest' parameter is NULL then a single plot will be displayed.

Returns: either an 'sfc' object (if the initial 'area_of_interest' parameter is NULL) or an 'sf' object consisting of the n-degree grid cells

Method `selected_aoi_sf()`**:** Returns the selected area of interest (AOI) by the user in form of an 'sfc' object

Usage:

```
select_aoi_global_grid$selected_aoi_sf()
```

Returns: an 'sfc' object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
select_aoi_global_grid$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://github.com/r-spatial/mapedit/blob/master/experiments/gadget_draw2.R

Examples

```
## Not run:

require(IceSat2R)
require(magrittr)
```

```
#.....................................
# 1st. Option: Select an AOI from the map
#.....................................

init = select_aoi_global_grid$new(area_of_interest = NULL,
                                  verbose = TRUE)
init$draw_edit_aoi()
sf_obj = init$selected_areas_global_grid(plot_data = TRUE)
sf_obj


#.........................................
# observe the available countries and continents
# using the 'rnaturalearth' R package
#.........................................

cntr_cnt = rnaturalearth::ne_countries(scale = 110,
                                       type = 'countries',
                                       returnclass = 'sf')

cntr_cnt = cntr_cnt[, c('sovereignt', 'continent')]

# sort(cntr_cnt$sovereignt)
# sort(unique(cntr_cnt$continent))


#.....................................
# 2nd. Option: Select a 'country' as AOI   (5-degrees query)
#.....................................

init = select_aoi_global_grid$new(area_of_interest = 'Antarctica',
                                  verbose = TRUE)

init$draw_edit_aoi(degrees = 5.0, square_geoms = TRUE)
sf_obj = init$selected_areas_global_grid(plot_data = TRUE)
sf_obj


#.....................................
# 3rd. Option: Select a 'continent' as AOI   (1-degree query)
#.....................................

init = select_aoi_global_grid$new(area_of_interest = 'North America',
                                  verbose = TRUE)

init$draw_edit_aoi(degrees = 1.0, square_geoms = TRUE)
sf_obj = init$selected_areas_global_grid(plot_data = TRUE)
sf_obj


#.....................................
# 4th. Option: Use a bounding box as input  ('Greenland Ice Sheet')
```

```
#.....................................

data(ne_10m_glaciated_areas)

dat_bbx = ne_10m_glaciated_areas %>%
  subset(!is.na(name)) %>%
  subset(name == "Greenland Ice Sheet") %>%
  sf::st_bbox(crs = 4326)

dat_bbx

init = select_aoi_global_grid$new(area_of_interest = dat_bbx,
                                  verbose = TRUE)

init$draw_edit_aoi(degrees = 1.0, square_geoms = TRUE)
sf_obj = init$selected_areas_global_grid(plot_data = TRUE)
sf_obj


#.................................................
# 5th. Option: Create a global hexagonal 5-degrees grid
#.................................................

bbx_global = c(xmin = -180, xmax = 180, ymin = -90, ymax = 90)

init = select_aoi_global_grid$new(area_of_interest = bbx_global,
                                  verbose = TRUE)

init$draw_edit_aoi(degrees = 5.0, square_geoms = FALSE)
sf_obj = init$selected_areas_global_grid(plot_data = TRUE)
sf_obj


## End(Not run)
```

`time_specific_orbits` *Time Specific Orbits*

Description

This function shows the reference ground track time and locations for specific date ranges. "Updated KML files have been posted to the 'tech-specs' website (see the 'references' section for more details) containing individual files for each Reference Ground Track (RGT) with a date and time stamp posted every 420 kilometers along-track (roughly 1 minute of flight time in between each point). The first RGT is 234; this is where the time series begins. The date of each RGT is in the file name, so the user can easily ascertain where and when ICESat-2 will be on a particular day."

Usage

```
time_specific_orbits(
```

```
  date_from = NULL,
  date_to = NULL,
  RGT_cycle = NULL,
  download_method = "curl",
  threads = 1,
  verbose = FALSE
)
```

Arguments

date_from	either NULL or a character string specifying the start date in the format 'yyyy-MM-dd' (such as '2020-01-01'). If this parameter is NULL then the 'RGT_cycle' parameter must be specified
date_to	either NULL or a character string specifying the end date in the format 'yyyy-MM-dd' (such as '2020-01-01'). If this parameter is NULL then the 'RGT_cycle' parameter must be specified
RGT_cycle	a character vector specifying the RGT (Reference Ground Track) cycle(s) (the specific revisit times of ICESAT-2). This parameter can be greater or equal to 1 with a maximum 'RGT-cycle' names as determined by the output of the 'available_RGTs(only_cycle_names = TRUE)' function. The computation time of a single 'RGT-cycle' might take approximately 15 minutes utilizing 8 threads (in parallel) and require approximately 2 GB of memory.
download_method	a character string specifying the download method. Corresponds to the 'method' parameter of the 'utils::download.file()' function. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'
threads	an integer that specifies the number of threads to use in parallel when processing the data
verbose	a boolean. If TRUE then information will be printed out in the console

Value

an 'sf' object that will include one or more Reference Ground Tracks (see the 'RGT' column of the output object)

References

https://icesat-2.gsfc.nasa.gov/science/specs

Examples

```
## Not run:

require(IceSat2R)

#..........................................
# RGTs (Reference Ground Tracks) for a single day
#..........................................
```

```
res_rgt_one = time_specific_orbits(date_from = '2019-06-01',
                                   date_to = '2019-06-01',
                                   download_method = 'curl',
                                   threads = 1,
                                   verbose = TRUE)
str(res_rgt_one)

#.....................................................
# RGTs (Reference Ground Tracks) for a specific time period
#.....................................................

res_rgt_many = time_specific_orbits(date_from = '2019-06-01',
                                    date_to = '2019-06-03',
                                    download_method = 'curl',
                                    threads = 1,
                                    verbose = TRUE)
str(res_rgt_many)


#.....................................................
# processing more than one RGTs for a specified date range
#.....................................................

res_rgt_inters = time_specific_orbits(date_from = '2021-03-23',
                                      date_to = '2021-03-26',
                                      download_method = 'curl',
                                      threads = 1,
                                      verbose = TRUE)
str(res_rgt_inters)

table(res_rgt_inters$cycle)
table(res_rgt_inters$day_of_year)
table(res_rgt_inters$RGT)

#.........................................................
# RGTs (Reference Ground Tracks) for a selected 'cycle'
# Observe the available RGT-cycles and use all available threads
#.........................................................

avail_cycles = available_RGTs(only_cycle_names = TRUE,
                              verbose = TRUE)
avail_cycles

choose_cycle = avail_cycles[3]

res_rgt_many = time_specific_orbits(RGT_cycle = choose_cycle,
                                    download_method = 'curl',
                                    threads = parallel::detectCores(),
                                    verbose = TRUE)

## End(Not run)
```

verify_RGTs *Verification of the Reference Ground Tracks (RGTs)*

Description

This function allows the user to verify the NSIDC extracted RGTs with the corresponding OpenAltimetry using the same Dates

Usage

```
verify_RGTs(nsidc_rgts, bbx_aoi, verbose = FALSE, ...)
```

Arguments

nsidc_rgts	a data.frame, data.table or tibble object that includes the columns 'Date_time' and 'RGT'
bbx_aoi	a named numeric vector or an sf-bbox object with names 'xmin', 'ymin', 'xmax', 'ymax'
verbose	a boolean. If TRUE then information will be printed out in the console
...	further parameters for the getTracks function

Value

a 'data.table' object where it is possible that the number of the OpenAltimetry RGTs is higher compared to the NSIDC RGTs

Examples

```
## Not run:

require(IceSat2R)

rgts = data.table::setDT(list(RGT = c(1251L, 1252L, 1260L, 1267L, 1275L),
                              Date_time = c("2020-12-15", "2020-12-15",
                              "2020-12-15", "2020-12-16", "2020-12-16")))
bbx = c(xmin = -53.108876, ymin = 60.119614, xmax = -19.203521, ymax = 80.793117)

dtbl = verify_RGTs(nsidc_rgts = rgts, bbx_aoi = bbx, verbose = TRUE)
dtbl

# split by Date to observe RGTs by date

split(dtbl, by = 'Date_time')


## End(Not run)
```

vsi_kml_from_zip *Utilizing Virtual File Systems (vsi) to extract the .kml from the .zip file*

Description

This function returns the '.kml' and '.kmz' files in form of virtual file paths. Moreover, the user has the option to download these files.

Usage

```
vsi_kml_from_zip(
  icesat_rgt_url,
  download_zip = FALSE,
  download_method = "curl",
  verbose = FALSE
)
```

Arguments

icesat_rgt_url	a character string specifying the input .zip URL
download_zip	a boolean. If TRUE the .zip file will be first downloaded and then the .kml files will be returned, otherwise the 'gdalinfo' function will be used as input to the R 'system2()' function to read the .kml files without downloading the .zip file. The 'gdalinfo' command requires that the user has configured GDAL properly.
download_method	a character string specifying the download method. Corresponds to the 'method' parameter of the 'utils::download.file()' function. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'
verbose	a boolean. If TRUE then information will be printed out in the console

Value

an one column data.table of the output files

References

https://icesat-2.gsfc.nasa.gov/science/specs

https://gdal.org/user/virtual_file_systems.html

Examples

```
## Not run:

require(IceSat2R)

URL = 'https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/arcticallorbits.zip'
```

```
#.................................
# without downloading the .zip file
#.................................

res_out = vsi_kml_from_zip(icesat_rgt_url = URL,
                           download_zip = FALSE,
                           download_method = 'curl',
                           verbose = TRUE)
str(res_out)


#.............................
# by downloading the .zip file
#.............................

res_out = vsi_kml_from_zip(icesat_rgt_url = URL,
                           download_zip = TRUE,
                           download_method = 'curl',
                           verbose = TRUE)
str(res_out)


## End(Not run)
```

---

vsi_nominal_orbits_wkt *Utilizing Virtual File Systems (vsi) and Well Known Text (WKT) to access the 'nominal orbits'*

Description

Utilizing Virtual File Systems (vsi) and Well Known Text (WKT) to access the 'nominal orbits'

Usage

```
vsi_nominal_orbits_wkt(
  orbit_area,
  track = "GT7",
  rgt_repeat = 1,
  wkt_filter = NULL,
  download_method = "curl",
  download_zip = FALSE,
  verbose = FALSE
)
```

Arguments

orbit_area	a character string specifying the earth partition to use, it can be one of 'antarctic', 'arctic', 'western_hemisphere' and 'eastern_hemisphere'

track — a character string specifying the orbit track. Can be one of 'GT1L','GT1R','GT2L','GT2R','GT3L','GT3R or 'GT7'

rgt_repeat — an integer specifying the orbit repeat. This parameter defaults to 1 and it is relevant if a user chooses one of the 'western_hemisphere' or 'eastern_hemisphere' where there are 8 orbit repeats, whereas for the 'antarctic' and 'arctic' there is only 1 repeat

wkt_filter — either NULL, or a Well Known Text (WKT) character string to allow a user to restrict to an area of interest rather than processing all data (this parameter will be used as input to the 'sf::st_read()' function)

download_method — a character string specifying the download method. Corresponds to the 'method' parameter of the 'utils::download.file()' function. Can be one of 'internal', 'wininet' (Windows only), 'libcurl', 'wget', 'curl' or 'auto'

download_zip — a boolean. If TRUE the .zip file will be first downloaded and then the .kml files will be returned, otherwise the 'gdalinfo' function will be used as input to the R 'system2()' function to read the .kml files without downloading the .zip file. The 'gdalinfo' command requires that the user has configured GDAL properly. Set the parameter 'download_zip' to TRUE if GDAL is not (properly) installed.

verbose — a boolean. If TRUE then information will be printed out in the console

Value

an 'sf' object

References

https://icesat-2.gsfc.nasa.gov/science/specs

https://gdal.org/user/virtual_file_systems.html

Examples

```
## Not run:

require(IceSat2R)
require(magrittr)

#.....................................
# processing all data of the orbit area
#.....................................

dat_rgt = vsi_nominal_orbits_wkt(orbit_area = 'eastern_hemisphere',
                                 track = 'GT7',
                                 rgt_repeat = 1,
                                 wkt_filter = NULL,
                                 download_method = 'curl',
                                 download_zip = FALSE,
                                 verbose = TRUE)
str(dat_rgt)
```

```
#.....................................
# extracting nominal orbits only for the WKT
#.....................................

WKT = 'POLYGON ((-14.765 18.979, -11.25 18.979, -11.25 21.943, -14.765 21.943, -14.765 18.979))'

dat_rgt = vsi_nominal_orbits_wkt(orbit_area = 'western_hemisphere',
                                 track = 'GT3R',
                                 rgt_repeat = 8,
                                 wkt_filter = WKT,
                                 download_method = 'curl',
                                 download_zip = FALSE,
                                 verbose = TRUE)
str(dat_rgt)
dat_rgt[[1]]$RGT                 # Reference Ground Tracks of input WKT

#........................
# Visualize the results
# (first compute the centroid)
#........................

wkt_sf = sf::st_as_sfc(WKT, crs = 4326)
centr_wkt = sf::st_coordinates(sf::st_centroid(wkt_sf))

RGTs = mapview::mapview(dat_rgt, legend = F)
AOI_wkt = mapview::mapview(wkt_sf, legend = F)

lft = RGTs + AOI_wkt
lft@map %>% leaflet::setView(lng = as.numeric(centr_wkt[, 'X']),
                             lat = as.numeric(centr_wkt[, 'Y']),
                             zoom = 7)

## End(Not run)
```

`vsi_time_specific_orbits_wkt` *Utilizing Virtual File Systems (vsi) and Well Known Text (WKT) to access the 'time specific orbits'*

Description

Utilizing Virtual File Systems (vsi) and Well Known Text (WKT) to access the 'time specific orbits'

Usage

```
vsi_time_specific_orbits_wkt(
  date_from,
  date_to,
  RGTs,
```

```
  wkt_filter = NULL,
  download_zip = FALSE,
  verbose = FALSE
)
```

Arguments

date_from	a character string specifying the 'start' date in the format 'yyyy-MM-dd' (such as '2020-01-01')
date_to	a character string specifying the 'end' date in the format 'yyyy-MM-dd' (such as '2020-01-01')
RGTs	a character vector (consisting of one or more) Reference Ground Track (RGT). See the Examples section on how to come to these RGTs based on the "vsi_nominal_orbits_wkt()" function
wkt_filter	either NULL, or a Well Known Text (WKT) character string to allow a user to restrict to an area of interest rather than processing all data. It is possible that the WKT won't intersect with any of the available time specific orbits due to the sparsity of the coordinates (the output in that case will be an empty list)
download_zip	a boolean. If TRUE the .zip file will be first downloaded and then the .kml files will be returned, otherwise the 'gdalinfo' function will be used as input to the R 'system2()' function to read the .kml files without downloading the .zip file. The 'gdalinfo' command requires that the user has configured GDAL properly. Set the parameter 'download_zip' to TRUE if GDAL is not (properly) installed.
verbose	a boolean. If TRUE then information will be printed out in the console

Details

In case that this function does not return any results (empty list object) for a specified 'wkt_filter' parameter, then use a bigger Well Known Text (WKT) area. This is required because the 'time specific orbits' (points) are quite sparse.

Moreover, set the parameter 'download_zip' to TRUE if the 'gdalinfo' function returns internally an empty character string. In that case also a warning will be shown in the R session.

Value

a list of 'sf' objects where each sublist will represent a different RGT cycle

References

https://icesat-2.gsfc.nasa.gov/science/specs

https://gdal.org/user/virtual_file_systems.html

Examples

```
## Not run:

require(IceSat2R)
require(magrittr)
```

```
#.....................................
# extracting nominal orbits only for the WKT
#.....................................

WKT = 'POLYGON ((-14.765 18.979, -11.25 18.979, -11.25 21.943, -14.765 21.943, -14.765 18.979))'

dat_rgt = vsi_nominal_orbits_wkt(orbit_area = 'western_hemisphere',
                                 track = 'GT3R',
                                 rgt_repeat = 8,
                                 wkt_filter = WKT,
                                 download_method = 'curl',
                                 download_zip = FALSE,
                                 verbose = TRUE)
str(dat_rgt)

out_rgt = dat_rgt[[1]]$RGT

#.....................................
# time specific RGTs (for a time interval)
# request using a single RGT cycle
#.....................................

date_start = '2020-01-01'
date_end = '2020-02-01'

orb_cyc_single = vsi_time_specific_orbits_wkt(date_from = date_start,
                                              date_to = date_end,
                                              RGTs = out_rgt,
                                              wkt_filter = WKT,
                                              verbose = TRUE)
str(orb_cyc_single)

#.....................................
# time specific RGTs (for a time interval)
# request using more than one RGT cycles
#.....................................

date_start = '2019-11-01'
date_end = '2020-01-01'

orb_cyc_multi = vsi_time_specific_orbits_wkt(date_from = date_start,
                                             date_to = date_end,
                                             RGTs = out_rgt,
                                             wkt_filter = WKT,
                                             verbose = TRUE)
str(orb_cyc_multi)
table(orb_cyc_multi$cycle)


#.................................................
# visualization of the output cycles (including the WKT)
#.................................................
```

```
orb_cyc_multi$cycle = as.factor(orb_cyc_multi$cycle)
mp_orb = mapview::mapview(orb_cyc_multi, legend = TRUE, zcol = 'cycle')

sf_aoi = sf::st_as_sfc(WKT, crs = 4326)
mp_aoi = mapview::mapview(sf_aoi, alpha.regions = 0.3, legend = F)

mp_orb + mp_aoi


## End(Not run)
```

Index