

Package: ocean3d (via r-universe)

October 15, 2026

Title Three-Dimensional Marine Spatial Analysis

Version 0.1.1.9009

Description Facilitates the analysis of marine habitat in 3D, enabling descriptions of habitat by depth and area. Provides tools for volume calculation, environmental extraction, and spatial overlap analysis between species ranges, fisheries, and depth-stratified environmental data.

URL <https://github.com/Marine-Biodiversity-Conservation-Lab/ocean3d>,
<https://marine-biodiversity-conservation-lab.github.io/ocean3d/>

BugReports <https://github.com/Marine-Biodiversity-Conservation-Lab/ocean3d/issues>

License GPL-3

Imports curl, magrittr, methods, rlang, sf ($\geq 1.1.0$), stringr, terra,
rredlist, ncdf4

Encoding UTF-8

Roxygen list(markdown = TRUE)

Suggests dplyr, gfwr (≥ 3.0), ggplot2, here, knitr, rmarkdown,
testthat ($\geq 3.0.0$), tibble, tidyterra, vcr

Config/testthat/edition 3

VignetteBuilder knitr

Remotes GlobalFishingWatch/gfwr

Config/roxygen2/version 8.1.0

Config/Needs/website rmarkdown

Config/testthat/parallel true

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libicu-dev libnetcdf-
dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository <https://ropensci-staging.r-universe.dev>

Date/Publication 2026-10-15 12:20:53 UTC

RemoteUrl <https://github.com/Marine-Biodiversity-Conservation-Lab/ocean3d>

RemoteRef HEAD

RemoteSha 239d4f75b3b6836e9030eefca4844251184a810b

Contents

as_envelope	2
as_voxel	4
calc_volume_overlap	5
create_study_raster	6
depths	7
envelope_to_voxel	8
extract_to_area	11
extract_to_point	12
extract2d	15
extract3d_all	16
extract3d_bottom	17
extract3d_nearest	18
extract3d_surface	19
fetch_species_assessments	20
fill_missing_depths	21
gfw_effort_to_raster	22
intersect_3d	24
intersects_3d	26
load_gebco_bathymetry	28
mask-3d	29
occupied	31
SpatEnvelope-class	33
SpatVolume-class	34
SpatVoxel-class	35
vect_to_envelope	36
volume	37
voxel_profiles	39
voxel_to_envelope	40
woa_cache_clear	42
woa_cache_dir	43
woa_download	43
woa_load_nc	45
Index	47

as_envelope

Coerce a 2D footprint to a 2.5D min-max envelope

Description

Build a [SpatEnvelope](#) — one continuous [depth_min, depth_max] interval per grid cell — from a 2D horizontal footprint and the depth limits that apply to it. This is the general form of the conversion [vect_to_envelope\(\)](#) performs for polygons: any 2D raster whose non-NA cells mark presence becomes a 3D domain once depth limits are attached to it.

Usage

```
as_envelope(x, depth_min, depth_max)
```

Arguments

x SpatRaster. Either a single-layer footprint whose non-NA cells are present, or a two-layer raster already named `depth_min`, `depth_max`.

depth_min, depth_max Shallowest and deepest depth in metres, positive down. Each is either a single number applying to the whole footprint, or a single-layer SpatRaster on the grid of `x` giving the limit per cell. Omit both when `x` already carries the two depth layers.

Details

`x` is the footprint. Its values are not read, only their non-NA pattern: a rasterized species range, a Global Fishing Watch effort layer, or a plain presence mask all work. A SpatRaster that *already* has exactly the two layers `depth_min` and `depth_max` (the output of [vect_to_envelope\(\)](#), or a [SpatEnvelope](#) itself) is promoted directly instead, in which case `depth_min` and `depth_max` must be omitted.

Depths are positive metres increasing downward, following the package's depth sign convention. Flip GEBCO-style elevation first, e.g. `terra::clamp(-terra::project(bathy, x), lower = 0)`, which gives a positive seafloor depth with land clamped to 0.

Value

A [SpatEnvelope](#) with layers `depth_min` and `depth_max`, on the grid of `x`.

Examples

```
# A 2x2 footprint: three cells present (any non-NA value), one absent.
fp <- terra::rast(nrows = 2, ncols = 2, xmin = 0, xmax = 2,
                  ymin = 0, ymax = 2)
terra::values(fp) <- c(1, 1, 1, NA)

# A species recorded between 0 and 200 m.
e <- as_envelope(fp, depth_min = 0, depth_max = 200)
terra::values(e)

# Per-cell limits are allowed too, as single-layer rasters.
dmax <- terra::setValues(terra::rast(fp), c(100, 200, 300, 400))
terra::values(as_envelope(fp, depth_min = 10, depth_max = dmax))
```

as_voxel	<i>Create a voxel object</i>
----------	------------------------------

Description

Wrap a multi-depth raster as a [SpatVoxel](#): the validated 3D form used throughout the package, with one layer per standard depth and layer names following the {variable}_depth={value} convention.

Usage

```
as_voxel(x, depths = NULL, varname = "value")
```

Arguments

x	SpatRaster with {variable}_depth={value} layer names, a list of single-depth SpatRasters, or an existing SpatVoxel .
depths	Optional numeric vector, one depth per layer, in metres. When supplied, layer names are (re)built from depths and varname, replacing any existing names. Required if x has no conforming layer names.
varname	Character. Variable name used when building layer names from depths. Ignored when depths is NULL.

Details

Most terra operations (crop(), mask(), [[, arithmetic) propagate the class, so a SpatVoxel normally survives them; a few, such as mean(), return a plain SpatRaster instead. Re-wrapping is cheap either way, since as_voxel() is idempotent — given a valid SpatVoxel and no depths it returns it untouched.

Propagating the class does not re-run the validity rules, so an operation that changes the raster layer set can leave an object still labelled SpatVoxel even if invalid. Passing an invalid SpatVoxel to as_voxel() rebuilds from layer names, passing through the function as if a plain multi-layer raster. Pass depths to rebuild the layer names.

Depths are positive metres increasing downward, matching the World Ocean Atlas convention. Negative depths are an error rather than being silently negated, since flipping the sign would change what the data mean.

Value

A [SpatVoxel](#) whose layers are ordered shallow to deep.

Examples

```
r <- terra::rast(nrows = 2, ncols = 2, nlyrs = 3)
terra::values(r) <- runif(terra::ncell(r) * 3)

# build the layer names from a depth vector
```

```
v <- as_voxel(r, depths = c(0, 100, 200), varname = "temp")
names(v)

# already-conforming names are used as they stand, and sorted if needed
names(r) <- c("temp_depth=200", "temp_depth=0", "temp_depth=100")
names(as_voxel(r))
```

calc_volume_overlap *Per-cell 3D volume overlap between two rasterized domains*

Description

Computes the depth interval and volume of each domain and of their intersection, cell by cell. The intersection is [intersect_3d\(\)](#); this function measures it. Accepts any combination of [SpatEnvelope](#) and [SpatVoxel](#); when the two differ, the envelope is discretized onto the voxel's depth levels with [envelope_to_voxel\(\)](#) rather than the voxel being collapsed, because [voxel_to_envelope\(\)](#) fills interior gaps and would overstate the overlap.

Usage

```
calc_volume_overlap(x, y, ...)

## S4 method for signature 'SpatEnvelope,SpatEnvelope'
calc_volume_overlap(x, y, ...)

## S4 method for signature 'SpatVoxel,SpatVoxel'
calc_volume_overlap(x, y, bounds = c("top", "midpoint"))

## S4 method for signature 'SpatEnvelope,SpatVoxel'
calc_volume_overlap(x, y, bounds = c("top", "midpoint"), ...)

## S4 method for signature 'SpatVoxel,SpatEnvelope'
calc_volume_overlap(x, y, bounds = c("top", "midpoint"), ...)

## S4 method for signature 'SpatRaster,SpatRaster'
calc_volume_overlap(x, y, ...)
```

Arguments

x, y	The two domains, each a SpatEnvelope or SpatVoxel .
...	Arguments for the voxel methods, which are an error for a pair of envelopes.
bounds	Voxel methods only. See volume() . Also governs how an envelope is discretized when the two inputs differ.

Details

Two voxels must be sampled at the same depth levels, and all inputs must be on the same grid. To ask only whether two domains overlap, use [intersects_3d\(\)](#); it computes no volumes.

Value

Multi-layer SpatRaster with 9 layers:

depth_min_a, depth_max_a Depth limits of x (m)

depth_min_b, depth_max_b Depth limits of y (m)

depth_min_overlap, depth_max_overlap Depth limits of the intersection (m). NA where the domains do not overlap in depth.

volume_a, volume_b Per-cell volume of each domain (km³)

volume_overlap Per-cell overlap volume (km³). 0 where both are present but their depths do not intersect, NA where either is absent.

For voxel inputs the depth layers are *summary bounds* — the shallowest and deepest occupied level. A voxel with an interior gap is not solid between them, so read the volume layers, not the depth layers, for occupied volume. The result is a plain SpatRaster: it is neither an envelope nor a voxel.

See Also

[intersect_3d\(\)](#) for the shared domain itself, and [volume\(\)](#) for its total volume; [intersects_3d\(\)](#) when only the presence of overlap is needed.

Examples

```
fp <- terra::rast(nrows = 1, ncols = 2, xmin = 0, xmax = 2000,
                  ymin = 0, ymax = 1000,
                  crs = "+proj=laea +lat_0=0 +lon_0=0 +datum=WGS84 +units=m")
terra::values(fp) <- c(1, 1)

a <- as_envelope(fp, depth_min = 0, depth_max = 100)
b <- as_envelope(fp, depth_min = 50, depth_max = 200)

ov <- calc_volume_overlap(a, b)
names(ov)
terra::global(ov[[c("volume_a", "volume_b", "volume_overlap")]], "sum",
              na.rm = TRUE)

# Mixed input: the envelope is discretized onto the voxel's levels.
bv <- envelope_to_voxel(b, depths = c(0, 50, 100, 150, 200))
terra::global(calc_volume_overlap(a, bv)[["volume_overlap"]], "sum",
              na.rm = TRUE)
```

create_study_raster *Create a study area raster grid*

Description

Build an empty raster covering the combined extent of one or more spatial objects. Useful for defining the common grid before rasterizing species ranges and fishery footprints.

Usage

```
create_study_raster(layers, res = 0.01, crs = "EPSG:4326")
```

Arguments

- layers List of sf, sfc, SpatVector, or SpatRaster objects. The output extent will cover all objects.
- res Numeric vector of length 1 or 2. Cell resolution in units of crs (degrees for lon/lat). Default 0.01 (~1 km at equator).
- crs Character. Coordinate reference system. Default "EPSG:4326".

Value

An empty SpatRaster with the computed extent, resolution, and CRS.

Examples

```
# Two features in different places: the grid has to cover both.
a <- terra::vect("POLYGON ((0 0, 2 0, 2 2, 0 2, 0 0))", crs = "EPSG:4326")
b <- terra::vect("POLYGON ((3 3, 5 3, 5 5, 3 5, 3 3))", crs = "EPSG:4326")

grid <- create_study_raster(list(a, b), res = 0.5)
terra::ext(grid)
dim(grid)                   # rows, columns, layers
terra::hasValues(grid) # FALSE: the grid is empty, it only defines geometry

# Inputs are projected to `crs` before their extents are combined, so a list
# mixing coordinate systems is fine.
create_study_raster(list(a, terra::project(b, "EPSG:3857")), res = 0.5)
```

depths	<i>Depths of a voxel's layers</i>
--------	-----------------------------------

Description

Read the depth axis of a [SpatVoxel](#). Depth is the layer index in a voxel, and the depth itself is carried in the layer name following the {variable}_depth={value} convention used throughout the package; this parses it back out.

Usage

```
depths(x)
```

Arguments

- x A [SpatVoxel](#), a [terra::SpatRaster](#), or a character vector of layer names.

Details

`x` is normally a `SpatVoxel`, but any `terra::SpatRaster` whose layer names follow the convention works, as does a bare character vector of layer names — useful for the row names of a `terra::global()` result, which carry the layer names but not the raster.

Depths are positive metres increasing downward.

Value

Numeric vector, one depth per layer of `x` (or per element, when `x` is character). `NA` for any name that does not follow the convention; an error when no name does.

See Also

`as_voxel()`, which builds those layer names.

Examples

```
r <- terra::rast(nrows = 2, ncols = 2, nlyrs = 3)
terra::values(r) <- runif(terra::ncell(r) * 3)
v <- as_voxel(r, depths = c(0, 100, 200), varname = "temp")

names(v)
depths(v)

# A character vector works too, so the depths of a per-layer summary can be
# recovered from its row names.
per_depth <- terra::global(v, "mean", na.rm = TRUE)
depths(rownames(per_depth))

# Names that do not follow the convention come back NA, as long as at least
# one name does.
depths(c("temp_depth=0", "not_a_depth_layer"))
```

envelope_to_voxel

Convert Envelope 2.5D -> Voxel 3D

Description

Expand a `SpatEnvelope` to the `SpatVoxel`, with an input of depth levels. A depth level belongs to a cell when the slab of water it stands for shares some thickness with that cell's `[depth_min, depth_max]` interval; depths outside the interval, and cells that are `NA` in the envelope, are `NA` in every layer.

Usage

```
envelope_to_voxel(
  x,
  depths,
  values = NULL,
```

```

    profile = NULL,
    bounds = c("top", "midpoint"),
    varname = "presence"
)

```

Arguments

x	SpatEnvelope, e.g. from as_envelope() or voxel_to_envelope() .
depths	Array of values that can be coerced into numeric type, represents metres depth below sea level. Sorted shallow to deep, and deduplicated, before use.
values	Optional numeric or SpatRaster of same CRS, resolution, extent as x. Defines values to write for voxel cells that are within depth intervals. NULL (the default) writes 1 at every occupied level, giving simple presence/absence.
profile	Optional function distributing values across the depth dimension, taking (ind, depths, n_depths) and returning the weight to apply. NULL (the default) writes the same value at every depth, as profile_flat() does; profile_equal() divides the value equally across the depths a cell occupies. Any function meeting the contract in voxel_profiles works, including your own.
bounds	How to place the edges of the slab each depth level stands for. "top" (the default) runs the slab from the level down to the next, so the level is its shallow edge; "midpoint" puts the edges halfway between neighbouring levels. Under both, the outer edges are clamped to range(depths) rather than extrapolated.
varname	Name to use for depth layer, taking on form of {varname}_depth={depths[i]}

Details

- values is the magnitude, which can be a single number for the whole grid or a SpatRaster carrying a value per cell. Default value is 1, which makes the output a presence mask. Ex. values can be 2D fisheries effort rasters.
- profile decides how that magnitude is spread down the water column. Default NULL means the full value is written at every occupied level. Providing a profile_* function determines how values is transformed for each depth level. Ex. [profile_equal\(\)](#) divides it evenly over the levels the cell occupies, so sum of all depth levels in the voxel equals values.

A depth level stands for a slab of water rather than a knife-edge, and a cell occupies that level when its envelope overlaps the slab by more than a shared edge. An interval of [10, 20] against levels c(0, 100) therefore occupies the 0 m level, rather than falling between the levels and coming back empty. Touching is not overlapping: an interval that ends exactly where a slab begins, [0, 100] against the slab 100-200, does not occupy that level, so adjacent depth ranges are placed on disjoint levels and do not intersect as voxels. This is the same rule [intersect_3d\(\)](#) applies between two envelopes.

Where a slab's edges fall is a property of the dataset the levels came from, not of the levels themselves, so bounds selects the convention:

- "top" (the default) runs each slab from its level down to the next, so a level names the top of what it stands for. Levels c(0, 100, 200) give slabs 0-100, 100-200.

- "midpoint" puts the edges halfway between neighbouring levels, so a level sits in the middle of what it stands for. This is the World Ocean Atlas convention: 0, 5, 10, ... the 0 m layer covers 0–2.5 m, the 5 m layer 2.5–7.5 m, the 10 m layer 7.5–12.5 m, and so on.

Under either convention the slabs cover the span from the shallowest level to the deepest and no further: outer edges are clamped to `range(depths)` rather than extrapolated, which is why WOA's 0 m layer starts at the surface rather than half a gap above it. The expansion is therefore limited only at the two ends when a cell whose envelope lies wholly above `min(depths)` or wholly below `max(depths)`. Under "top" the deepest level has no next level to run to, so it stands for a zero-thickness slab: an envelope is recorded there only when it reaches below `max(depths)`, and one that ends exactly at `max(depths)` stops at the level above. For the same reason a [voxel_to_envelope\(\)](#) round trip drops each cell's deepest occupied level: the envelope's `depth_max` names that level, which is the top of its slab.

Value

A [SpatVoxel](#) with one layer per depth in `depths`, on the grid of `x`, layers ordered shallow to deep.

See Also

[voxel_to_envelope\(\)](#), the reverse (and lossy) collapse.

Examples

```
fp <- terra::rast(nrows = 1, ncols = 2, xmin = 0, xmax = 2,
                  ymin = 0, ymax = 1)
terra::values(fp) <- c(1, NA)
e <- as_envelope(fp, depth_min = 50, depth_max = 200)

# presence at every standard depth the envelope covers
terra::values(envelope_to_voxel(e, depths = c(0, 50, 100, 200, 300)))

# the same pattern, carrying a constant of your choosing instead of 1
terra::values(
  envelope_to_voxel(e, depths = c(0, 50, 100, 200, 300),
                    values = 5)
)

# spread that value down the column instead: an even share at each of the
# depths the cell occupies, summing back to 1
terra::values(
  envelope_to_voxel(e, depths = c(0, 50, 100, 200, 300),
                    profile = profile_equal,
                    varname = "time")
)

# the same envelope read against WOA-style layer bounds: [60, 70] sits in the
# 50–150 m slab of the 100 m level, where the default puts it at 0–100 m
shallow <- as_envelope(fp, depth_min = 60, depth_max = 70)
terra::values(envelope_to_voxel(shallow, depths = c(0, 100, 200)))
terra::values(envelope_to_voxel(shallow, depths = c(0, 100, 200),
                                bounds = "midpoint"))
```

```
# a per-cell magnitude comes in as a raster: split each cell's effort evenly
# over the depths it occupies, conserving the cell total
effort <- terra::setValues(terra::rast(fp), c(100, NA))
terra::values(
  envelope_to_voxel(
    e, depths = c(0, 50, 100, 200, 300), values = effort,
    profile = profile_equal,
    varname = "effort"
  )
)
```

extract_to_area	<i>Extract a 3D raster to an area and a depth band</i>
-----------------	--

Description

Crop a [SpatVoxel](#) to an area polygon and select the depth layers within a given depth range. The nearest available depth layers to `min_depth` and `max_depth` are used as the inclusive bounds, so the result always has at least one layer; omit either to run to that end of the voxel.

Usage

```
extract_to_area(area, rast_3d, min_depth = NULL, max_depth = NULL)
```

Arguments

<code>area</code>	sf, sfc, or SpatVector . Area polygon to crop the voxel to. Reprojected to the voxel's CRS when the two differ.
<code>rast_3d</code>	SpatVoxel . Multi-depth raster, e.g. from as_voxel() or envelope_to_voxel() .
<code>min_depth</code>	Numeric. Shallowest depth in metres. NULL (the default) starts at the voxel's shallowest layer.
<code>max_depth</code>	Numeric. Deepest depth in metres. NULL (the default) runs to the voxel's deepest layer.

Details

This is the area counterpart of [extract_to_point\(\)](#): the same "restrict a 3D source to a target geometry" operation, with a polygon as the target rather than observation points.

To restrict a voxel to a species' *per-cell* depth window rather than one depth band across the whole area, build a [SpatEnvelope](#) and mask with it instead: `mask(rast_3d, range_rast)`. See [mask-3d](#).

Value

A [SpatVoxel](#) cropped to area and filtered to the depth range.

See Also

`extract_to_point()` for the point counterpart; `depths()` for the layer depths the bounds snap to.

Examples

```
# A 4x4 voxel over five standard depths.
r <- terra::rast(nrows = 4, ncols = 4, xmin = -10, xmax = 10,
                 ymin = -10, ymax = 10, nlyrs = 5, crs = "EPSG:4326")
terra::values(r) <- seq_len(terra::ncell(r) * 5)
v <- as_voxel(r, depths = c(0, 50, 100, 500, 1000), varname = "t_an")

area <- terra::vect("POLYGON ((-5 -5, 5 -5, 5 5, -5 5, -5 -5))",
                   crs = "EPSG:4326")

# 40-600 m snaps to the 50, 100 and 500 m layers.
out <- extract_to_area(area, v, min_depth = 40, max_depth = 600)
names(out)
terra::ext(out) # cropped to the polygon

# Omit the bounds to keep every layer.
names(extract_to_area(area, v))
```

extract_to_point

Extract values from netCDFs to one or more point observations

Description

This is the general point-extraction interface. It can extract environmental values for one or multiple observations. The user chooses the type of extraction with the `method` argument. The more specific functions `extract2d()`, `extract3d_surface()`, `extract3d_bottom()`, `extract3d_nearest()` and `extract3d_all()` are wrappers around this function.

Usage

```
extract_to_point(
  data = NULL,
  nc,
  lon = NULL,
  lat = NULL,
  depth = NULL,
  date = NULL,
  var = NULL,
  method = c("2d", "surface", "bottom", "nearest", "all"),
  lon_col = "lon",
  lat_col = "lat",
  date_col = "date",
  depth_col = "depth",
  id_col = NULL,
```

```

    file_col = "file",
    lon_dim = NULL,
    lat_dim = NULL,
    depth_dim = NULL,
    time_dim = NULL,
    time_match = c("nearest", "exact"),
    max_time_diff = NULL,
    output_prefix = NULL,
    output_col = NULL,
    verbose = FALSE
  )

```

Arguments

data	Optional object containing observation points. Can be a data frame, tibble, sf object with POINT geometries, matrix with column names, or named list. If NULL, coordinates can be supplied directly through lon, lat, and optionally depth and date. For sf objects, longitude and latitude are obtained from the POINT geometries; projected geometries are transformed internally to longitude and latitude when a CRS is available.
nc	netCDF source. Can be a file path, vector of file paths, list of file paths, opened ncdf4 object, list of opened objects, or a data frame with a file column.
lon	Optional numeric longitude value or vector for direct point extraction when data = NULL.
lat	Optional numeric latitude value or vector for direct point extraction when data = NULL.
depth	Optional numeric observation depth value or vector for direct point extraction. Required when data = NULL and method = "nearest" or method = "all".
date	Optional observation date or vector of dates for direct point extraction. Used when data = NULL and the netCDF variable has a time dimension. Character dates should use "YYYY-MM-DD" format. Direct input vectors may have length 1 or a common compatible length.
var	Name of the variable to extract from the netCDF file. If NULL, the function tries to detect the variable automatically from each file. This only works when each netCDF file contains one single variable.
method	Extraction method. Options are "2d", "surface", "bottom", "nearest" and "all". With method = "all", the function returns nearest, surface and bottom outputs together.
lon_col	Name of the longitude column in data.
lat_col	Name of the latitude column in data.
date_col	Name of the date column in data. Required only when the netCDF variable has a time dimension.
depth_col	Name of the observation-depth column in data. Required only when method = "nearest" or method = "all".
id_col	Optional observation identifier column. It is not required for extraction, but it is checked when provided to help users detect mistakes.

file_col	Column containing file paths when nc is a data frame.
lon_dim	Optional name of the longitude dimension in the netCDF file.
lat_dim	Optional name of the latitude dimension in the netCDF file.
depth_dim	Optional name of the depth dimension in the netCDF file.
time_dim	Optional name of the time dimension in the netCDF file.
time_match	Character. How observation dates are matched to netCDF time values. "nearest" (default) selects the closest available date; "exact" requires an exact date match and returns NA when no match exists.
max_time_diff	Optional non-negative numeric value giving the maximum allowed difference, in days, between an observation date and the nearest netCDF date when time_match = "nearest". If NULL (default), no maximum temporal distance is imposed. Ignored when time_match = "exact".
output_prefix	Optional name used for the extracted output column. If omitted, var is used.
output_col	Optional alias for output_prefix. Use this when you want to define the exact name of the output column directly. If both output_col and output_prefix are provided, output_col is used. With method = "all", this name is used as the suffix in nearest_*, surface_* and seabottom_* columns.
verbose	Logical. If TRUE, progress messages are printed every 100 rows.

Value

If data is provided, the input observations with extracted values appended. Data frames, tibbles and sf objects retain their structure; matrices and named lists are returned as data frames. If data = NULL, direct extraction returns a numeric value for one point and one output, a numeric vector for multiple points and one output, a named numeric vector for one point and multiple outputs (for example, method = "all"), or a data frame for multiple points and multiple outputs.

Examples

```
nc_file <- system.file(
  "extdata", "example_3d.nc",
  package = "ocean3d"
)

# Extract the value closest to a single observation
extract_to_point(
  nc = nc_file,
  lon = 0,
  lat = 40,
  depth = 60,
  date = "2020-01-03",
  var = "temp",
  method = "nearest"
)

# Extract values for multiple observations
extract_to_point(
  nc = nc_file,
```

```

lon = c(0, 1),
lat = c(40, 41),
depth = c(60, 100),
date = "2020-01-03",
var = "temp",
method = "nearest"
)

```

extract2d

Extract values from a 2D netCDF variable

Description

Use this function when the variable has longitude and latitude dimensions, and optionally a time dimension, but no depth dimension.

Usage

```
extract2d(data, nc, var = NULL, ...)
```

Arguments

data	Object containing observation points. Can be a data frame, tibble, sf object with POINT geometries, matrix with column names, or named list.
nc	netCDF source. Can be a file path, vector of file paths, list of file paths, opened ncdf4 object, list of opened objects, or a data frame with a file column.
var	Name of the variable to extract from the netCDF file. If NULL, the function tries to detect the variable automatically from each file. This only works when each netCDF file contains one single variable.
...	Additional arguments passed to extract_to_point() .

Value

The input observations with extracted values appended. Data frames, tibbles and sf objects retain their structure; matrices and named lists are returned as data frames.

Examples

```

nc_file <- system.file(
  "extdata", "example_2d.nc",
  package = "ocean3d"
)

observations <- data.frame(
  lon = c(0, 1),
  lat = c(40, 41),
  date = as.Date(c("2020-01-01", "2020-01-03"))
)

```

```

)

extract2d(
  data = observations,
  nc = nc_file,
  var = "temp"
)

```

extract3d_all

Extract nearest, surface and bottom values from a 3D netCDF variable

Description

For each observation, this function extracts the full vertical profile at the nearest longitude, latitude and time cell. It then returns three summary columns: nearest valid depth, surface and bottom.

Usage

```
extract3d_all(data, nc, var = NULL, ...)
```

Arguments

data	Object containing observation points. Can be a data frame, tibble, sf object with POINT geometries, matrix with column names, or named list.
nc	netCDF source. Can be a file path, vector of file paths, list of file paths, opened ncdf4 object, list of opened objects, or a data frame with a file column.
var	Name of the variable to extract from the netCDF file. If NULL, the function tries to detect the variable automatically from each file. This only works when each netCDF file contains one single variable.
...	Additional arguments passed to extract_to_point() .

Value

The input observations with three extracted columns: nearest_*, surface_* and seabottom_*. Data frames, tibbles and sf objects retain their structure; matrices and named lists are returned as data frames.

Examples

```

nc_file <- system.file(
  "extdata", "example_3d.nc",
  package = "ocean3d"
)

observations <- data.frame(
  lon = c(0, 1),
  lat = c(40, 41),

```

```

    depth = c(60, 100),
    date = as.Date(c("2020-01-01", "2020-01-03"))
  )

  extract3d_all(
    data = observations,
    nc = nc_file,
    var = "temp"
  )

```

extract3d_bottom	<i>Extract the bottom available layer from a 3D netCDF variable</i>
------------------	---

Description

For each observation, this function extracts the full vertical profile at the nearest longitude, latitude and time cell, then returns the deepest non-missing value. This is useful when the bathymetry of the netCDF grid means that deeper layers are missing over shallow areas.

Usage

```
extract3d_bottom(data, nc, var = NULL, ...)
```

Arguments

data	Object containing observation points. Can be a data frame, tibble, sf object with POINT geometries, matrix with column names, or named list.
nc	netCDF source. Can be a file path, vector of file paths, list of file paths, opened ncdf4 object, list of opened objects, or a data frame with a file column.
var	Name of the variable to extract from the netCDF file. If NULL, the function tries to detect the variable automatically from each file. This only works when each netCDF file contains one single variable.
...	Additional arguments passed to extract_to_point() .

Value

The input observations with extracted values appended. Data frames, tibbles and sf objects retain their structure; matrices and named lists are returned as data frames.

Examples

```

nc_file <- system.file(
  "extdata", "example_3d.nc",
  package = "ocean3d"
)

observations <- data.frame(

```

```

lon = c(0, 1),
lat = c(40, 41),
date = as.Date(c("2020-01-01", "2020-01-03"))
)

extract3d_bottom(
  data = observations,
  nc = nc_file,
  var = "temp"
)

```

extract3d_nearest	<i>Extract the nearest valid depth layer from a 3D netCDF variable</i>
-------------------	--

Description

For each observation, this function reads the full vertical profile and finds the valid, non-missing netCDF depth layer closest to the observation depth. This avoids returning NA when the geometrically closest depth layer is invalid at that grid cell, for example below the seabed.

Usage

```
extract3d_nearest(data, nc, var = NULL, ...)
```

Arguments

data	Object containing observation points. Can be a data frame, tibble, sf object with POINT geometries, matrix with column names, or named list.
nc	netCDF source. Can be a file path, vector of file paths, list of file paths, opened ncdf4 object, list of opened objects, or a data frame with a file column.
var	Name of the variable to extract from the netCDF file. If NULL, the function tries to detect the variable automatically from each file. This only works when each netCDF file contains one single variable.
...	Additional arguments passed to extract_to_point() .

Value

The input observations with extracted values appended. Data frames, tibbles and sf objects retain their structure; matrices and named lists are returned as data frames.

Examples

```

nc_file <- system.file(
  "extdata", "example_3d.nc",
  package = "ocean3d"
)

```

```

observations <- data.frame(
  lon = c(0, 1),
  lat = c(40, 41),
  depth = c(60, 100),
  date = as.Date(c("2020-01-01", "2020-01-03"))
)

extract3d_nearest(
  data = observations,
  nc = nc_file,
  var = "temp"
)

```

extract3d_surface	<i>Extract the surface layer from a 3D netCDF variable</i>
-------------------	--

Description

The surface layer is defined as the first available depth layer in the netCDF file. This is usually the shallowest layer.

Usage

```
extract3d_surface(data, nc, var = NULL, ...)
```

Arguments

data	Object containing observation points. Can be a data frame, tibble, sf object with POINT geometries, matrix with column names, or named list.
nc	netCDF source. Can be a file path, vector of file paths, list of file paths, opened ncdf4 object, list of opened objects, or a data frame with a file column.
var	Name of the variable to extract from the netCDF file. If NULL, the function tries to detect the variable automatically from each file. This only works when each netCDF file contains one single variable.
...	Additional arguments passed to extract_to_point() .

Value

The input observations with extracted values appended. Data frames, tibbles and sf objects retain their structure; matrices and named lists are returned as data frames.

Examples

```
nc_file <- system.file(
  "extdata", "example_3d.nc",
  package = "ocean3d"
)

observations <- data.frame(
  lon = c(0, 1),
  lat = c(40, 41),
  date = as.Date(c("2020-01-01", "2020-01-03"))
)

extract3d_surface(
  data = observations,
  nc = nc_file,
  var = "temp"
)
```

fetch_species_assessments

Fetch species assessment data from IUCN Red List API

Description

Query IUCN Red List API for species assessment data including taxonomy, Red List category, depth limits, and assessment metadata. Retrieves the most recent global assessment for each species.

Usage

```
fetch_species_assessments(
  api_key,
  sis_ids = NULL,
  species_names = NULL,
  group_code = NULL
)
```

Arguments

api_key	Character. IUCN Red List API token. Set once per session with <code>rredlist::rl_use_iucn()</code> , or pass directly here.
sis_ids	Numeric vector. SIS taxon IDs (i.e., <code>id_no</code> from IUCN shapefiles). Default <code>NULL</code> .
species_names	Character vector. Scientific names in "Genus species" format. Default <code>NULL</code> .
group_code	Character. A single comprehensive group code (e.g., "sharks_and_rays"). Default <code>NULL</code> .

Details

Exactly one of `sis_ids`, `species_names`, or `group_code` must be provided.

Requires the `rredlist` package (`install.packages("rredlist")`).

Value

Data frame with columns: `assessment_id`, `assessment_date`, `sis_id`, `scientific_name`, `kingdom_name`, `phylum_name`, `class_name`, `order_name`, `family_name`, `genus_name`, `species_name`, `subpopulation_name`, `red_list_category`, `systems_code`, `upper_depth_limit`, `lower_depth_limit`, `citation`, `url`.

Examples

```
## Not run:
# By SIS taxon IDs (e.g., id_no column from IUCN shapefile)
assessments <- fetch_species_assessments(api_key, sis_ids = c(39332, 39385))

# By scientific names
assessments <- fetch_species_assessments(
  api_key,
  species_names = c("Sphyrna lewini", "Carcharhinus amblyrhynchos")
)

# By comprehensive group (all sharks and rays)
assessments <- fetch_species_assessments(
  api_key,
  group_code = "sharks_and_rays"
)

## End(Not run)
```

fill_missing_depths	<i>Fill missing depth values</i>
---------------------	----------------------------------

Description

Fix swapped upper/lower depth values and fill NAs using genus-level means. IUCN Red List assessments (see [fetch_species_assessments\(\)](#)) sometimes record depth limits the wrong way around or omit them entirely. Designed for use inside `dplyr::mutate()` — returns a two-column tibble (`upper_depth` and `lower_depth`) that can be unpacked with `mutate()`.

Usage

```
fill_missing_depths(upper, lower, genus, method = "genus_mean")
```

Arguments

upper	Numeric vector. Upper (shallower) depth limit values, possibly with NAs or swapped values.
lower	Numeric vector. Lower (deeper) depth limit values, possibly with NAs or swapped values.
genus	Character vector. Genus names, used to compute genus-level mean depths for filling NAs.
method	Character. Method for filling missing values. Currently only "genus_mean" is supported. Default "genus_mean".

Value

A tibble with columns `upper_depth` and `lower_depth`, suitable for use with `dplyr::mutate()`.

Examples

```
# Depth limits as they can arrive from an IUCN assessment: the first row is
# swapped (upper deeper than lower) and the third is missing entirely.
assessments <- data.frame(
  genus_name = c("Carcharhinus", "Carcharhinus", "Carcharhinus", "Sphyrna"),
  upper_depth_limit = c(280, 0, NA, 0),
  lower_depth_limit = c(0, 100, NA, 512)
)

fill_missing_depths(
  assessments$upper_depth_limit,
  assessments$lower_depth_limit,
  assessments$genus_name
)

# A genus with no non-missing values anywhere has nothing to fill from:
fill_missing_depths(c(NA, NA), c(NA, NA), c("Raja", "Raja"))

# Typical use, unpacking both columns inside dplyr::mutate():
## Not run:
assessments <- assessments %>%
  mutate(fill_missing_depths(upper_depth_limit, lower_depth_limit,
                             genus_name))

## End(Not run)
```

`gfw_effort_to_raster` *Rasterise a GFW effort tibble onto a target grid*

Description

Turn the long-format apparent-fishing-hours tibble returned by `gfwr::gfw_ais_fishing_hours()` (formerly `gfwr::get_raster()`) into a multi-layer `SpatRaster` on the package's canonical study grid. Each level of `layer_by` becomes its own layer, named `effort_<level>` (e.g. `effort_drifting_longlines`).

Usage

```
gfw_effort_to_raster(
  effort,
  grid = NULL,
  layer_by = "geartype",
  value = "Apparent Fishing Hours",
  fun = "sum"
)
```

Arguments

<code>effort</code>	Data frame. Output of <code>gfwr::gfw_ais_fishing_hours()</code> (a long-format tibble with at minimum Lat, Lon, a value, and a grouping column).
<code>grid</code>	<code>SpatRaster</code> . Target grid (extent, resolution, CRS) — typically the same grid used for species ranges and WOA extraction. If <code>grid = NULL</code> , assume target grid from input effort data frame.
<code>layer_by</code>	Character. Column in <code>effort</code> whose levels become layers. <code>NULL</code> produces a single-layer total-effort raster. Default <code>"geartype"</code> .
<code>value</code>	Character. Column in <code>effort</code> to aggregate. Default <code>"Apparent Fishing Hours"</code> .
<code>fun</code>	Character or function. Aggregation applied to records that fall into the same cell $\times$ layer level. Default <code>"sum"</code> .

Details

The input is expected to carry a cell centroid (Lat, Lon), a value column (default `"Apparent Fishing Hours"`), and one categorical column matching the API's `group_by` — for example `geartype` or `flag` (note lower-case; this is what `gfwr` actually returns). Records that fall into the same target cell $\times$ layer level are aggregated with `fun`.

Value

A `SpatRaster` with one layer per `layer_by` level (or one layer if `layer_by = NULL`). Layer names follow `effort_<level>`.

Examples

```
## Not run:
jpn_eez_id <- gfwr::gfw_region_id(region = "JPN", region_source = "EEZ")$id
effort <- gfwr::gfw_ais_fishing_hours(
  spatial_resolution = "LOW",          # 0.01 deg
  temporal_resolution = "YEARLY",      # one bucket per year in [start, end]
  start_date          = "2022-01-01",
  end_date            = "2022-12-31",
  region              = jpn_eez_id,
  region_source       = "EEZ",
  group_by            = "GEARTYPE"
)

# Returns data.frame
```

```

head(effort)

# Create multi-layer SpatRaster from data.frame, one for each geartype
effort_by_gear <- gfw_effort_to_raster(
  effort    = effort,
  layer_by  = "geartype",
  value     = "Apparent Fishing Hours",
  fun       = "sum"
)

# Plot drifting longlines effort
terra::plot(effort_by_gear$effort_drifting_longlines)

## End(Not run)

```

intersect_3d

The 3D space two objects share

Description

`intersect_3d()` returns the part of space that is inside both `x` and `y`. It answers the question "what do these two share?". To ask only "do they share anything?", use [intersects_3d\(\)](#). To keep one object's values where the other is present, use [mask\(\)](#).

Usage

```

intersect_3d(x, y, ...)

## S4 method for signature 'SpatEnvelope,SpatEnvelope'
intersect_3d(x, y, ...)

## S4 method for signature 'SpatVoxel,SpatVoxel'
intersect_3d(x, y)

## S4 method for signature 'SpatEnvelope,SpatVoxel'
intersect_3d(x, y, bounds = c("top", "midpoint"), ...)

## S4 method for signature 'SpatVoxel,SpatEnvelope'
intersect_3d(x, y, bounds = c("top", "midpoint"), ...)

## S4 method for signature 'SpatEnvelope,ANY'
intersect_3d(x, y, ...)

## S4 method for signature 'SpatVoxel,ANY'
intersect_3d(x, y)

## S4 method for signature 'ANY,SpatEnvelope'
intersect_3d(x, y, ...)

```

```
## S4 method for signature 'ANY,SpatVoxel'
intersect_3d(x, y, ...)

## S4 method for signature 'ANY,ANY'
intersect_3d(x, y, ...)
```

Arguments

x, y	The two objects. At least one must be a SpatEnvelope or a SpatVoxel . The other can be either of those, a single-layer SpatRaster footprint, or polygons (SpatVector , sf , sfc). Rasters must share one grid (CRS, extent, resolution).
...	Arguments for the voxel methods. They are an error when both inputs are envelopes.
bounds	Only when one input is an envelope and the other a voxel. How the envelope is placed on the voxel's depth levels: "top" (default) or "midpoint". See envelope_to_voxel() .

Details

At least one of x and y must be a 3D object: a [SpatEnvelope](#) or a [SpatVoxel](#). The other can be a 3D object too, or a 2D object. A 2D object restricts the result horizontally and leaves its depths alone.

Two envelopes Cell by cell, the result runs from the deeper depth_min to the shallower depth_max. Intervals that only touch share no water, so that cell is empty (NA).

Two voxels Both must be sampled at the same depth levels. A level is in the result where both voxels occupy it.

An envelope and a voxel The envelope is first placed on the voxel's depth levels with [envelope_to_voxel\(\)](#). The result is a voxel.

A 3D object and a [SpatRaster](#) The raster is a footprint. Its non-NA cells are inside; its values are not read. It must have one layer and be on the same grid.

A 3D object and polygons A [SpatVector](#), [sf](#) or [sfc](#) object. The polygons are rasterised onto the 3D object's grid. A cell is inside when its centre is. Polygons in another CRS are projected first.

The order of x and y does not matter.

The result is a domain, not a field. An envelope result carries the shared depth interval. A voxel result carries presence: 1 where a level is shared, NA elsewhere, in layers named presence_depth=<value>. The cell values of a voxel input are not carried over. To keep them, use [mask\(\)](#).

Value

A [SpatEnvelope](#) when both inputs are envelopes, or when one is an envelope and the other is 2D. Otherwise a [SpatVoxel](#) of presence. Either is on the grid of the 3D input.

terra's intersect()

terra::intersect() on two rasters is a 2D test that ignores depth. On a [SpatEnvelope](#) or a [SpatVoxel](#), in either position, it is an error that points to [intersects_3d\(\)](#) and intersect_3d(). Plain rasters, vectors and extents keep terra's behaviour.

See Also

[intersects_3d\(\)](#) for the yes/no form; [mask\(\)](#) to keep a voxel's values inside a domain; [volume\(\)](#) for the volume of the result; [calc_volume_overlap\(\)](#), which is built on this function.

Examples

```
fp <- terra::rast(nrows = 1, ncols = 3, xmin = 0, xmax = 3000,
                  ymin = 0, ymax = 1000,
                  crs = "+proj=laea +lat_0=0 +lon_0=0 +datum=WGS84 +units=m")
terra::values(fp) <- c(1, 1, 1)

# Two species: one at 0-100 m, one at 50-200 m everywhere.
a <- as_envelope(fp, depth_min = 0, depth_max = 100)
b <- as_envelope(fp, depth_min = 50, depth_max = 200)

# They share 50-100 m in every cell.
shared <- intersect_3d(a, b)
terra::values(shared)
volume(shared)

# With a voxel, the result is a presence voxel on the voxel's levels.
bv <- envelope_to_voxel(b, depths = c(0, 50, 100, 150, 200))
names(intersect_3d(a, bv))

# With polygons, the envelope is kept where the polygon is.
poly <- terra::vect("POLYGON ((0 0, 2000 0, 2000 1000, 0 1000, 0 0))",
                    crs = terra::crs(fp))
terra::values(intersect_3d(a, poly))
```

intersects_3d

Do two objects share any 3D space?

Description

intersects_3d() tests, cell by cell, whether x and y overlap in 3D. It is the yes/no form of [intersect_3d\(\)](#) and takes the same inputs. It computes only the presence pattern, with no cell areas and no volumes, so it is the cheap choice for richness and tally maps.

Usage

```
intersects_3d(x, y, ...)
```

```
## S4 method for signature 'SpatEnvelope,SpatEnvelope'
```

```

intersects_3d(x, y, ...)

## S4 method for signature 'SpatVoxel,SpatVoxel'
intersects_3d(x, y)

## S4 method for signature 'SpatEnvelope,SpatVoxel'
intersects_3d(x, y, bounds = c("top", "midpoint"), ...)

## S4 method for signature 'SpatVoxel,SpatEnvelope'
intersects_3d(x, y, bounds = c("top", "midpoint"), ...)

## S4 method for signature 'SpatEnvelope,ANY'
intersects_3d(x, y, ...)

## S4 method for signature 'SpatVoxel,ANY'
intersects_3d(x, y)

## S4 method for signature 'ANY,SpatEnvelope'
intersects_3d(x, y, ...)

## S4 method for signature 'ANY,SpatVoxel'
intersects_3d(x, y, ...)

## S4 method for signature 'ANY,ANY'
intersects_3d(x, y, ...)

```

Arguments

<code>x, y</code>	The two objects. At least one must be a SpatEnvelope or a SpatVoxel . The other can be either of those, a single-layer SpatRaster footprint, or polygons (SpatVector , sf , sf). Rasters must share one grid (CRS, extent, resolution).
<code>...</code>	Arguments for the voxel methods. They are an error when both inputs are envelopes.
<code>bounds</code>	Only when one input is an envelope and the other a voxel. How the envelope is placed on the voxel's depth levels: "top" (default) or "midpoint". See envelope_to_voxel() .

Details

Each cell gets one of three answers:

- TRUE: both objects are present and their depths overlap.
- FALSE: both are present but their depths do not overlap, or only one of them is present.
- NA: neither is present. There is nothing to compare.

This is how `terra` answers the 2D question for two rasters, so the result sums and plots like any other boolean layer. Depth intervals that only touch do not overlap. When `y` is a 2D footprint or polygons there is no depth axis to compare, so the test is only whether both are present.

The order of `x` and `y` does not matter.

Value

A single-layer boolean `SpatRaster` named `intersects`, on the grid of the 3D input: `TRUE`, `FALSE` or `NA` per cell as above.

See Also

[intersect_3d\(\)](#) for the shared space itself; [mask\(\)](#) to keep a voxel's values inside a domain.

Examples

```
fp <- terra::rast(nrows = 1, ncols = 4, xmin = 0, xmax = 4000,
                 ymin = 0, ymax = 1000,
                 crs = "+proj=laea +lat_0=0 +lon_0=0 +datum=WGS84 +units=m")

# Four cells. A is present in cells 1-3 at 0-100 m. B is present in cells
# 1, 2 and 4, at 50-200 m in cell 1 and 300-400 m elsewhere.
a <- as_envelope(terra::setValues(fp, c(1, 1, 1, NA)),
                depth_min = 0, depth_max = 100)
b <- as_envelope(terra::setValues(fp, c(1, 1, NA, 1)),
                depth_min = terra::setValues(fp, c(50, 300, NA, 300)),
                depth_max = terra::setValues(fp, c(200, 400, NA, 400)))

# Cell 1: overlap. Cell 2: both present, depths disjoint. Cell 3: A only.
# Cell 4: B only.
terra::values(intersects_3d(a, b))

# Summing a stack of these gives a richness map. na.rm = TRUE counts each
# TRUE as 1 and each FALSE as 0.
richness <- sum(c(intersects_3d(a, b), intersects_3d(a, a)), na.rm = TRUE)
terra::values(richness)
```

`load_gebco_bathymetry` *Load GEBCO bathymetry raster*

Description

Load a GEBCO bathymetry raster from a NetCDF file using [terra::rast\(\)](#). Validates that the file is NetCDF format, contains an elevation variable, and has global extent (-180 to 180, -90 to 90). Values are returned as-is (GEBCO uses negative values for below sea level).

Usage

```
load_gebco_bathymetry(file_path)
```

Arguments

`file_path` Character. Path to GEBCO bathymetry NetCDF file (e.g., "gebco_2025_sub_ice_topo/GEBCO_2025-su

Value

SpatRaster with elevation values in metres.

Examples

```
# A real GEBCO grid is a multi-gigabyte download, so this example builds a
# tiny stand-in with the same structure (global extent, `elevation`
# variable, negative values below sea level) and loads it.
nc_path <- file.path(tempdir(), "gebco_example.nc")

template <- terra::rast(
  nrows = 36, ncols = 72,
  xmin = -180, xmax = 180, ymin = -90, ymax = 90,
  crs = "EPSG:4326"
)
terra::values(template) <- seq(-6000, 0, length.out = terra::ncell(template))
terra::varnames(template) <- "elevation"
terra::writeCDF(template, nc_path, varname = "elevation", overwrite = TRUE)

bathy <- load_gebco_bathymetry(nc_path)
bathy

# Depths are negative below sea level; flip the sign for use with
# vect_to_envelope(), which expects positive depths.
depth <- -bathy
terra::global(depth, "max", na.rm = TRUE)

# In practice, point at a downloaded GEBCO NetCDF instead:
# bathy <- load_gebco_bathymetry(
#   "gebco_2025_sub_ice_topo/GEBCO_2025_sub_ice.nc"
# )

unlink(nc_path)
```

mask-3d

Keep a raster's values inside a 3D domain

Description

These methods make `terra::mask()` depth-aware. `mask(x, mask)` keeps the values of `x` where `mask` is present and sets the rest to NA. When `mask` is a [SpatEnvelope](#) or a [SpatVoxel](#), "present" is decided cell by cell **and** depth by depth. A value of `x` stays only where the mask domain reaches that cell at that depth.

Usage

```
## S4 method for signature 'SpatVoxel,SpatEnvelope'
mask(x, mask, bounds = c("top", "midpoint"), ...)
```

```

## S4 method for signature 'SpatVoxel,SpatVoxel'
mask(x, mask, ...)

## S4 method for signature 'SpatEnvelope,SpatEnvelope'
mask(x, mask, ...)

## S4 method for signature 'SpatEnvelope,SpatVoxel'
mask(x, mask, bounds = c("top", "midpoint"), ...)

## S4 method for signature 'SpatRaster,SpatEnvelope'
mask(x, mask, ...)

## S4 method for signature 'SpatRaster,SpatVoxel'
mask(x, mask, ...)

```

Arguments

x	The raster whose values are kept: a SpatVoxel , a SpatEnvelope , or a plain SpatRaster .
mask	The 3D domain to keep x inside: a SpatEnvelope or a SpatVoxel on the same grid as x.
bounds	Only when one side is an envelope and the other a voxel. How the envelope is placed on the voxel's depth levels: "top" (default) or "midpoint". See envelope_to_voxel() .
...	Passed on to terra::mask() . For example, <code>inverse = TRUE</code> keeps the values <i>outside</i> the domain instead.

Details

What each pairing does:

voxel masked by an envelope The envelope is first placed on the voxel's own depth levels with [envelope_to_voxel\(\)](#). Each depth layer of the voxel is then masked by the matching level. This replaces the hand-written `terra::mask(v, envelope_to_voxel(e, depths(v)))` and cannot be pointed at the wrong depths.

voxel masked by a voxel Both must be sampled at the same depth levels. Each depth layer of x is masked by the matching layer of the mask, wherever the mask is occupied.

envelope masked by an envelope or a voxel A cell of x is kept where [intersects_3d\(\)](#) finds the two domains overlap. Its depth interval is kept whole. To narrow the interval instead, use [intersect_3d\(\)](#).

plain SpatRaster masked by an envelope or a voxel A cell is kept where the mask domain is present at any depth. Without this method terra would read the mask's first layer only.

A voxel or envelope masked by a plain [SpatRaster](#) or by polygons is terra's own [mask\(\)](#). Every layer is masked by the same 2D pattern and the class is kept. No method is added for those cases.

Value

x, with the same class and layers, with every value outside the domain set to NA.

See Also

[intersect_3d\(\)](#) for the shared domain itself; [intersects_3d\(\)](#) for the yes/no test; [extract_to_area\(\)](#) to restrict a voxel to a polygon and one depth band instead of a per-cell depth window.

Examples

```
fp <- terra::rast(nrows = 1, ncols = 3, xmin = 0, xmax = 3000,
                  ymin = 0, ymax = 1000,
                  crs = "+proj=laea +lat_0=0 +lon_0=0 +datum=WGS84 +units=m")
terra::values(fp) <- c(1, 1, 1)

# A temperature voxel on three levels, and a species found at 0-100 m in
# cells 1-2 only.
temp <- as_voxel(terra::rast(list(terra::setValues(fp, c(20, 21, 22)),
                                terra::setValues(fp, c(15, 16, 17)),
                                terra::setValues(fp, c(10, 11, 12)))),
                depths = c(0, 100, 200), varname = "temp")
range_env <- as_envelope(terra::setValues(fp, c(1, 1, NA)),
                        depth_min = 0, depth_max = 100)

# The temperatures inside the species' range: cell 3 and the 200 m level
# are gone.
terra::values(mask(temp, range_env))

# A 2D raster masked by the range keeps cells where the range is present
# at any depth.
terra::values(mask(fp, range_env))
```

 occupied

Reduce a variable voxel to a presence voxel

Description

A [SpatVoxel](#) holds a variable, such as temperature, oxygen or fishing effort. It does not say which cells count as occupied. `occupied()` applies a predicate and returns presence: 1 where fun is TRUE, NA elsewhere. The grid and the depth levels stay the same.

Usage

```
occupied(x, fun = function(v) !is.na(v))
```

Arguments

x	A SpatVoxel .
fun	Function taking a vector of cell values and returning a logical vector of the same length. NA counts as FALSE. Defaults to <code>function(v) !is.na(v)</code> , which treats any recorded value as occupied.

Details

The 3D query verbs read presence directly and take no predicate. Threshold each voxel first, so both sides of a query use their own cutoff:

```
intersect_3d(occupied(temp, function(v) v > 15),
             occupied(effort, function(v) v > 0))
```

fun receives one depth layer's cell values as a numeric vector. It must return a logical vector of the same length. A predicate that summarises the layer, such as `function(v) v > mean(v, na.rm = TRUE)`, therefore compares each cell against its own depth. [voxel_to_envelope\(\)](#) uses the same contract.

Running `occupied()` on a presence voxel returns it unchanged.

Value

A [SpatVoxel](#) on the depth levels of x. Values are 1 (occupied) or NA. Layers are named `presence_depth={value}`.

See Also

[voxel_to_envelope\(\)](#) collapses the result to a [SpatEnvelope](#). [volume\(\)](#), [intersect_3d\(\)](#), [intersects_3d\(\)](#) and [mask\(\)](#) take a presence voxel.

Examples

```
r <- terra::rast(nrows = 1, ncols = 3, nlyrs = 2)
terra::values(r) <- cbind(c(12, 18, 4), c(9, 16, 2))
v <- as_voxel(r, depths = c(0, 100), varname = "temp")

# Occupied where the water is warmer than 10 degrees.
warm <- occupied(v, function(x) x > 10)
names(warm)
terra::values(warm)

# The verbs need no predicate.
volume(warm)

# A presence voxel is returned unchanged.
identical(terra::values(occupied(warm)), terra::values(warm))

# Each side of a query uses its own cutoff.
effort <- as_voxel(terra::setValues(r, cbind(c(0, 3, 0), c(0, 2, 0))),
                  depths = c(0, 100), varname = "effort")
terra::values(intersect_3d(warm, occupied(effort, function(x) x > 0)))
```

SpatEnvelope-class	<i>2.5D min-max envelope: exactly depth_min, depth_max</i>
--------------------	--

Description

A SpatEnvelope is a [terra::SpatRaster](#) with exactly two layers, depth_min and depth_max, in which **depth is the cell value**. The variable is the object itself: the envelope delimits the vertical extent of a spatial phenomenon (typically a species range) rather than sampling a variable within it.

Details

The vertical interval is per-cell and continuous, and is solid by construction — an envelope cannot represent a gap in the vertical distribution. Depths are positive metres increasing downward, and depth_max is at least depth_min in every cell.

See Also

[SpatVoxel](#), [SpatVolume](#)

Examples

```
# A two-cell footprint, one cell present and one absent. Build an envelope
# with as_envelope() (from a raster footprint) or vect_to_envelope() (from
# polygons).
fp <- terra::rast(nrows = 1, ncols = 2, xmin = 0, xmax = 2, ymin = 0,
                  ymax = 1)
terra::values(fp) <- c(1, NA)
e <- as_envelope(fp, depth_min = 50, depth_max = 200)

names(e)          # always exactly these two layers
terra::values(e)  # ...and depth is the cell value, in metres

# The interval is per-cell, so limits can vary across the grid.
terra::values(
  as_envelope(fp, depth_min = 50,
              depth_max = terra::setValues(terra::rast(fp), c(120, 300)))
)

# Any other set of layers is rejected: a bare footprint is not an envelope.
try(methods::new("SpatEnvelope", fp))
```

SpatVolume-class

Union of the package's 3D domain representations

Description

SpatVolume is a class union over [SpatVoxel](#) and [SpatEnvelope](#). It exists as a dispatch target for operations that are meaningful on either representation because both determine a **3D domain over a 2D grid**: volume, volumetric overlap, vertical extent, printing.

Details

It deliberately asserts no shared structure. The two members store depth in dual roles (layer index vs. cell value), so any operation that touches the depth axis directly should dispatch on the concrete class instead.

See Also

[SpatVoxel](#) and [SpatEnvelope](#), the two members; [envelope_to_voxel\(\)](#) and [voxel_to_envelope\(\)](#) convert between them.

Examples

```
# The same 3D domain in both representations.
fp <- terra::rast(nrows = 1, ncols = 2, xmin = 0, xmax = 2, ymin = 0,
                  ymax = 1)
terra::values(fp) <- c(1, 1)
e <- as_envelope(fp, depth_min = 0, depth_max = 200)
v <- envelope_to_voxel(e, depths = c(0, 100, 200))

# Neither inherits from the other; both belong to the union.
methods::is(e, "SpatVolume")
methods::is(v, "SpatVolume")
methods::is(v, "SpatEnvelope")

# `SpatVolume` is a dispatch target, so a generic written against it accepts
# either representation.
setGeneric("n_depth_layers", function(x) standardGeneric("n_depth_layers"))
setMethod("n_depth_layers", "SpatVolume", function(x) terra::nlyr(x))
n_depth_layers(e)
n_depth_layers(v)

# But the depth axis itself is stored differently in the two, so anything
# reading depths must dispatch on the concrete class rather than the union.
names(e)
names(v)
```

SpatVoxel-class

3D voxel model: one layer per standard depth level

Description

A SpatVoxel is a [terra::SpatRaster](#) in which **depth is the layer index** and the cell values are the variable (temperature, oxygen, presence, ...). Layer names follow the {variable}_depth={value} convention and must be distinct and ordered shallow to deep. The depth axis is grid-wide: every cell is sampled at the same set of standard depths.

Details

A voxel may have interior gaps — a cell can be NA at one depth and non-NA at the depths above and below it.

See Also

[SpatEnvelope](#), [SpatVolume](#)

Examples

```
# Two cells sampled at three standard depths. Build one with as_voxel(),
# which names the layers for you and sorts them shallow to deep.
r <- terra::rast(nrows = 1, ncols = 2, xmin = 0, xmax = 2, ymin = 0,
                 ymax = 1, nlyrs = 3)
terra::values(r) <- cbind(c(12, 11), c(9, NA), c(6, 5))
v <- as_voxel(r, depths = c(0, 100, 200), varname = "temp")

names(v)          # depth lives in the layer name
terra::values(v)  # ...and the variable in the cell values

# Note cell 2: NA at 100 m, but with values above and below it. A voxel can
# hold that interior gap; a SpatEnvelope cannot.

# It is an ordinary SpatRaster underneath, so terra operations work on it
# unchanged. Re-wrapping with as_voxel() is cheap either way, since the
# constructor is idempotent.
methods::is(v, "SpatRaster")
terra::nlyr(terra::crop(v, terra::ext(0, 1, 0, 1)))
identical(names(as_voxel(v)), names(v))

# Validity is enforced on construction: every layer name must carry its
# depth, and the layers must run shallow to deep.
bad <- r
names(bad) <- c("a", "b", "c")
try(methods::new("SpatVoxel", bad))
```

vect_to_envelope *Convert SpatVector or sf to SpatEnvelope*

Description

Rasterize polygons onto a template raster grid and assign per-cell depth limits, outputting an SpatEnvelope object. The depths are clamped by depth_min (shallowest) and depth_max (deepest), and/or by rasters (ex. bathymetry) that are the same coordinate and resolution as the template. Where a constraint raster is NA at a cell its limit there is unknown, so the cell is dropped rather than falling back to the remaining constraints.

Usage

```
vect_to_envelope(polygon, template, depth_min, depth_max)
```

Arguments

polygon	sf or SpatVector, for example species ranges or fishery footprints.
template	SpatRaster that defines the horizontal grid of the SpatEnvelope output.
depth_min	List. Can contain both numeric and SpatRasters that match CRS, resolution, extent of template and contain numeric values. For each cell, the maximum across the depth_min list parameters is used as output SpatEnvelope depth_min layer cell value.
depth_max	List. Can contain both numeric and SpatRasters that match CRS, resolution, extent of template and contain numeric values. For each cell, the minimum across the depth_max list parameters is used as output SpatEnvelope depth_max layer cell value. Note that when depth_min is exactly depth_max, these cell values are dropped and replaced with NA.

Value

SpatEnvelope, with depth_min and depth_max layers.

See Also

[as_envelope\(\)](#), which attaches depth limits to a footprint that is already a raster.

Examples

```
# A 4x4 study grid, and a range polygon covering its middle. `polygon` and
# `template` must already share a CRS: this function does not reproject.
template <- terra::rast(nrows = 4, ncols = 4, xmin = 0, xmax = 4,
  ymin = 0, ymax = 4, crs = "EPSG:4326")
poly <- terra::vect("POLYGON ((1 1, 3 1, 3 3, 1 3, 1 1))", crs = "EPSG:4326")

# A species recorded between 50 m and 200 m throughout its range. Cells
# outside the polygon are NA.
```

```
e <- vect_to_envelope(poly, template, depth_min = 50, depth_max = 200)
terra::values(e)

# Bathymetry is not a special argument: pass the seafloor as a second
# `depth_max` and, per cell, the shallower of the two wins. This seafloor
# deepens from north to south, so the two rows the polygon covers meet the
# bed very differently.
seafloor <- terra::setValues(terra::rast(template),
                             rep(c(20, 45, 250, 400), each = 4))
e2 <- vect_to_envelope(poly, template,
                       depth_min = 50, depth_max = list(300, seafloor))
terra::values(e2)

# Two things happened. Where the bed is at 250 m, the nominal 300 m limit is
# truncated to it. Where the bed is at 45 m it sits above `depth_min`, so
# there is no water column left to occupy and the cell drops out entirely -
# leaving 2 of the polygon's 4 cells.
sum(!is.na(terra::values(e2[["depth_min"]]])))
```

volume

Total 3D volume of a rasterized domain

Description

Sums the occupied volume of a [SpatEnvelope](#) or a [SpatVoxel](#) over the whole grid. Both representations describe a 3D domain over a 2D grid, but they store depth in dual roles, so each computes the vertical extent differently:

Usage

```
volume(x, ...)

## S4 method for signature 'SpatEnvelope'
volume(x, ...)

## S4 method for signature 'SpatVoxel'
volume(x, bounds = c("top", "midpoint"))

## S4 method for signature 'SpatRaster'
volume(x, ...)
```

Arguments

x	A SpatEnvelope or SpatVoxel . A bare SpatRaster is rejected; build one with as_envelope() , vect_to_envelope() or as_voxel() first.
...	Arguments for the SpatVoxel method, which are an error for an envelope.

bounds SpatVoxel only. What each depth level stands for vertically: "top" (default) treats the level as the top of its slab, so the slab runs down to the next level; "midpoint" puts the level in the middle, with edges halfway to each neighbour (the World Ocean Atlas convention). Under "top" the deepest level has no next level and contributes no volume. Outer edges are clamped to `range(depths)` under both. Same argument, same meaning, as in `envelope_to_voxel()`.

Details

SpatEnvelope Depth is the cell value, and the interval is solid by construction. Volume is `sum(cell_area * (depth_max - depth_min))` over present cells.

SpatVoxel Depth is the layer index. Each occupied voxel contributes its cell area times the thickness of the slab its depth level stands for, so interior gaps are excluded rather than filled.

A voxel volume is quantized to the levels the voxel was sampled at, and a voxel built by `envelope_to_voxel()` counts a level as occupied on *any* overlap with the envelope. So a range of 0-100 m discretized onto 0, 50, 100, 150, 200 m occupies three levels spanning 0-150 m and reports more water than the envelope it came from. Coarse levels overstate; the error shrinks as the levels tighten around the range. Where an envelope's limits fall exactly on levels under `bounds = "top"`, the two agree.

Value

Numeric of length 1. Total volume in km³.

See Also

`calc_volume_overlap()` for the volume two domains share, and `intersect_3d()` for the shared domain itself; **SpatVolume** for why the two representations dispatch separately.

Examples

```
fp <- terra::rast(nrows = 2, ncols = 2, xmin = 0, xmax = 2000,
                  ymin = 0, ymax = 2000,
                  crs = "+proj=laea +lat_0=0 +lon_0=0 +datum=WGS84 +units=m")
terra::values(fp) <- c(1, 1, 1, NA)

# 2.5D: three 1 km2 cells, each 0-200 m => 3 * 1 * 0.2 = 0.6 km3.
e <- as_envelope(fp, depth_min = 0, depth_max = 200)
volume(e)

# 3D: the same domain on three levels. Under "top" the slabs are 100 m,
# 100 m and 0 m, so the discretized volume matches exactly here.
v <- envelope_to_voxel(e, depths = c(0, 100, 200))
volume(v)

# An interior gap is volume a voxel can drop and an envelope cannot.
gappy <- v
gappy[[2]] <- terra::setValues(terra::rast(v[[2]]), NA)
gappy <- as_voxel(gappy)
volume(gappy)
volume(voxel_to_envelope(gappy))
```

voxel_profiles	<i>Vertical profiles for voxel building</i>
----------------	---

Description

A vertical profile decides how the value a cell carries is spread down the depth levels that cell occupies. `envelope_to_voxel()` takes one through its `profile` argument, as a function — either one of these, or one you write.

Usage

```
profile_flat(ind, depths, n_depths)
```

```
profile_equal(ind, depths, n_depths)
```

Arguments

<code>ind</code>	Logical SpatRaster of occupancy, one layer per depth.
<code>depths</code>	Numeric vector of depths, shallow to deep, one per layer of <code>ind</code> .
<code>n_depths</code>	Single-layer SpatRaster counting each cell's occupied levels.

Value

A weight to multiply `ind` by: a single number, or a SpatRaster on the grid of `ind` with either one layer or one per depth.

Functions

- `profile_flat()`: The whole value at every occupied level. This is what `profile = NULL` means in `envelope_to_voxel()`; it exists as a function so the no-profile case is a profile like any other rather than a special case in the caller.
- `profile_equal()`: An even share at each occupied level, so the levels sum back to the cell's value. The share depends on how many levels the cell occupies, which varies across the grid wherever the envelope does: a cell spanning three levels gets a third at each, one spanning ten gets a tenth.

Writing a profile

A profile is called once per `envelope_to_voxel()` call, with three arguments, positionally:

`ind` Logical SpatRaster, one layer per depth, TRUE where a cell's envelope overlaps the slab that `depth` stands for — from the depth down to the next one, the deepest standing for itself alone. A level can therefore be occupied without its own value lying in the envelope. Cells that are NA in the envelope are NA here.

`depths` Numeric vector of the depths those layers stand for, shallow to deep. Always `terra::nlyr(ind)` long.

`n_depths` Single-layer `SpatRaster` counting the levels each cell occupies, i.e. `sum(ind)`. Passed in rather than left to be recomputed, since the caller already needs it.

It returns the weight that multiplies `ind`, in one of three forms: a single number applying to the whole grid, a single-layer `SpatRaster` giving one weight per cell, or a `terra::nlyr(ind)`-layer `SpatRaster` giving one per cell per level. Rasters must be on the grid of `ind`.

Because a profile is handed rasters and returns rasters, write it in terra algebra rather than in terms of one cell at a time. Unoccupied cells are masked out by the caller afterwards, so a profile need not zero them — but it should avoid dividing by a zero level count, since `Inf` is harder to read in a partial result than `NA`.

A profile that conserves does so *per cell*: the weights it returns sum to 1 within each cell, whatever that cell's interval.

Examples

```
fp <- terra::rast(nrows = 1, ncols = 2, xmin = 0, xmax = 2, ymin = 0,
                 ymax = 1)
terra::values(fp) <- c(1, 1)
dmax <- terra::setValues(terra::rast(fp), c(100, 300))
e <- as_envelope(fp, depth_min = 0, depth_max = dmax)
depths <- c(0, 100, 200, 300)

# the whole value at every occupied level
terra::values(envelope_to_voxel(e, depths, values = 12,
                              profile = profile_flat))

# an even share instead, so each cell's levels sum back to its value
terra::values(envelope_to_voxel(e, depths, values = 12,
                              profile = profile_equal))

# a profile of your own: weight the shallow levels twice as heavily as the
# deep ones, still conserving each cell's total
surface_weighted <- function(ind, depths, n_depths) {
  w <- terra::rast(lapply(depths,
                          function(d) ifelse(d <= 100, 2, 1) * ind[[1]]))

  w <- w * ind
  w / sum(w)
}
terra::values(envelope_to_voxel(e, depths, values = 12,
                              profile = surface_weighted))
```

voxel_to_envelope

Collapse Voxel 3D -> Envelope 2.5D

Description

Reduce a [SpatVoxel](#) to the [SpatEnvelope](#) that bounds it. A predicate fun is applied to the cell values at each depth; for every cell, the shallowest depth at which the predicate is `TRUE` becomes `depth_min` and the deepest becomes `depth_max`. Cells where the predicate is never `TRUE` are `NA` in both layers.

Usage

```
voxel_to_envelope(v, fun = function(x) !is.na(x))
```

Arguments

<code>v</code>	SpatVoxel (or a multi-depth SpatRaster with <code>{variable}_depth={value}</code> layer names).
<code>fun</code>	Function taking a vector of cell values and returning a logical vector of the same length. NA results are treated as FALSE. Defaults to <code>\(x) !is.na(x)</code> .

Details

The default predicate, `\(x) !is.na(x)`, gives the plain vertical extent of the data: the shallowest and deepest depths at which the cell has any value. Pass a different predicate to bound a subset of the values instead, e.g. `\(x) x > 15` for the depths over which a cell exceeds 15 degrees.

This conversion is lossy. An envelope stores a single continuous interval per cell, so any interior gap in the voxel is filled in: a cell that satisfies `fun` at 0 m and 200 m but not at 100 m still yields the envelope `[0, 200]`. Use [SpatVoxel](#) directly where interior gaps matter.

Value

A [SpatEnvelope](#) with layers `depth_min` and `depth_max`, on the same grid as `v`.

See Also

[envelope_to_voxel\(\)](#), the reverse expansion.

Examples

```
# Two cells sampled at four standard depths.
r <- terra::rast(nrows = 1, ncols = 2, xmin = 0, xmax = 2,
                 ymin = 0, ymax = 1, nlyrs = 4)
terra::values(r) <- cbind(c(12, NA), c(11, NA), c(NA, 8), c(6, NA))
v <- as_voxel(r, depths = c(0, 50, 100, 200), varname = "temp")

# Default predicate: the vertical extent of the data. Cell 1 has values at
# 0, 50 and 200 m, so it comes back as [0, 200] – the gap at 100 m is
# filled, because an envelope stores one continuous interval per cell.
terra::values(voxel_to_envelope(v))

# Another predicate bounds a subset of the values instead: here the depths
# over which a cell is warmer than 10 degrees. Cell 2 never qualifies, so it
# is NA in both layers.
terra::values(voxel_to_envelope(v, fun = function(x) x > 10))
```

woa_cache_clear	<i>Clear the WOA cache</i>
-----------------	----------------------------

Description

Remove all cached WOA NetCDF files.

Usage

```
woa_cache_clear(confirm = TRUE)
```

Arguments

confirm	Logical. Require interactive confirmation. Default TRUE.
---------	--

Value

Invisibly, TRUE on success.

Examples

```
# Redirect the cache into a temporary directory so this example does not
# delete real downloads. In normal use you would skip this step.
old <- Sys.getenv("R_USER_CACHE_DIR", unset = NA)
Sys.setenv(R_USER_CACHE_DIR = tempdir())

cache <- woa_cache_dir()
file.create(file.path(cache, "woa23_decav_t00_01.nc"))
list.files(cache)

# confirm = FALSE deletes without prompting (e.g. from a script)
woa_cache_clear(confirm = FALSE)
dir.exists(cache)

# The cache is recreated empty by the next call that needs it
woa_cache_dir()

if (is.na(old)) Sys.unsetenv("R_USER_CACHE_DIR") else
  Sys.setenv(R_USER_CACHE_DIR = old)

## Not run:
# Called with no arguments it asks for confirmation when interactive
woa_cache_clear()

## End(Not run)
```

woa_cache_dir	<i>WOA cache directory</i>
---------------	----------------------------

Description

Returns the path to the package's persistent cache directory for downloaded WOA NetCDF files. Uses `tools::R_user_dir()` so the location survives across sessions and follows platform conventions.

Usage

```
woa_cache_dir()
```

Value

Character. Path to cache directory (created if missing).

Examples

```
# Redirect the cache into a temporary directory so this example does not
# write to user filespace. In normal use you would skip this step and let
# the cache live in its platform-conventional location.
old <- Sys.getenv("R_USER_CACHE_DIR", unset = NA)
Sys.setenv(R_USER_CACHE_DIR = tempdir())

# The directory is created if it does not already exist
cache <- woa_cache_dir()
dir.exists(cache)

# This is where woa_download() writes when `output_dir` is not supplied
list.files(cache, pattern = "\\\\.nc$")

if (is.na(old)) Sys.unsetenv("R_USER_CACHE_DIR") else
  Sys.setenv(R_USER_CACHE_DIR = old)
```

woa_download	<i>Download a WOA NetCDF file (with caching)</i>
--------------	--

Description

Download World Ocean Atlas 2023 NetCDF files from the NCEI THREDDS server. Files are cached in `woa_cache_dir()` (or a user-supplied `output_dir`) and skipped on subsequent calls unless `force = TRUE`.

Usage

```

woa_download(
  variable,
  period = "annual",
  resolution = "0.25",
  decade = NULL,
  output_dir = NULL,
  force = FALSE,
  quiet = FALSE
)

```

Arguments

variable	Character. One of "temperature", "salinity", "dissolved_oxygen", "oxygen_saturation", "AOU", "nitrate", "phosphate", "silicate", "density".
period	Character or numeric. "annual" (default), "monthly" (all 12 months), "seasonal" (4 seasons), or a numeric vector where 0 = annual, 1:12 = monthly, 13:16 = seasonal.
resolution	Character or numeric. "0.25" (default), "1", or "5" degrees.
decade	Character. Decade code (e.g. "decav", "all"). Defaults to the canonical decade for the variable.
output_dir	Character. Destination directory. Defaults to <code>woa_cache_dir()</code> .
force	Logical. If TRUE, re-download even if the file exists. Default FALSE.
quiet	Logical. Suppress download progress. Default FALSE.

Details

On first use of the persistent cache in an interactive session, the function prompts for consent to write to user filespace (per CRAN Repository Policy). When the `curl` package is available, the estimated size of each remote file is reported before downloading. A full set of WOA files can be many gigabytes — supply `output_dir` to direct the cache elsewhere (HPC scratch, external drive, etc.).

Value

Character vector of paths to downloaded .nc files.

Examples

```

## Not run:
# Annual temperature climatology at 1-degree resolution, cached in a
# temporary directory so nothing is written to user filespace
tmp <- file.path(tempdir(), "woa")
f <- woa_download("temperature", period = "annual", resolution = "1",
                 output_dir = tmp)
woa_load_nc(f)

# All 12 monthly files, downloaded in parallel, into the default cache

```

```

monthly <- woa_download("temperature", period = "monthly",
                        resolution = "1")
length(monthly)

# Specific periods: 0 = annual, 1:12 = monthly, 13:16 = seasonal
winter <- woa_download("dissolved_oxygen", period = 13, resolution = "1")

# Other variables use their own canonical decade codes. Note that
# 0.25-degree files are large (~1 GB each for salinity).
sal <- woa_download("salinity", period = "annual", resolution = "0.25")

# Re-download a file that is already cached
f <- woa_download("temperature", period = "annual", resolution = "1",
                  output_dir = tmp, force = TRUE, quiet = TRUE)

## End(Not run)

```

woa_load_nc

Load a WOA NetCDF file

Description

Load a WOA .nc file and select layers for a given statistical field. Returns a SpatRaster with layer names already following the {variable}_depth={value} convention used throughout this package (native to WOA NetCDFs).

Usage

```
woa_load_nc(file_path, field = "an")
```

Arguments

file_path	Character. Path to a WOA .nc file.
field	Character. Statistical field to select. One of: "an" (objectively analyzed climatology), "mn" (statistical mean), "dd" (number of observations), "sd" (standard deviation), "se" (standard error), "oa" (mean minus climatology), "gp" (number of mean values within radius of influence), "sdo" (objectively analyzed standard deviation), "sea" (standard error of the analysis). See the WOA 2023 Product Documentation: https://repository.library.noaa.gov/view/noaa/70581

Value

SpatRaster with standardized depth layer names.

Examples

```
# Build a small WOA-style NetCDF to demonstrate field selection.
# Real files come from woa_download().
nc <- file.path(tempdir(), "woa_demo.nc")
demo <- terra::rast(nrows = 2, ncols = 2, nlyrs = 4,
                    xmin = 0, xmax = 2, ymin = 0, ymax = 2,
                    crs = "EPSG:4326")
names(demo) <- c("t_an_depth=0", "t_an_depth=100",
                "t_sd_depth=0", "t_sd_depth=100")
terra::values(demo) <- matrix(1:16, nrow = 4)
terra::writeCDF(demo, nc, overwrite = TRUE, split = TRUE)

# Objectively analyzed climatology (the default field)
temp <- woa_load_nc(nc)
names(temp)

# Standard deviation layers from the same file
names(woa_load_nc(nc, field = "sd"))

unlink(nc)

## Not run:
# Typical use on a downloaded file
f <- woa_download("temperature", period = "annual", resolution = "1")
temp <- woa_load_nc(f, field = "an")
terra::plot(temp[["t_an_depth=0"]])

## End(Not run)
```

Index

as_envelope, [2](#)
as_envelope(), [9](#), [36](#), [37](#)
as_voxel, [4](#)
as_voxel(), [8](#), [11](#), [37](#)

calc_volume_overlap, [5](#)
calc_volume_overlap(), [26](#), [38](#)
calc_volume_overlap, SpatEnvelope, SpatEnvelope-method
 (calc_volume_overlap), [5](#)
calc_volume_overlap, SpatEnvelope, SpatVoxel-method
 (calc_volume_overlap), [5](#)
calc_volume_overlap, SpatRaster, SpatRaster-method
 (calc_volume_overlap), [5](#)
calc_volume_overlap, SpatVoxel, SpatEnvelope-method
 (calc_volume_overlap), [5](#)
calc_volume_overlap, SpatVoxel, SpatVoxel-method
 (calc_volume_overlap), [5](#)
create_study_raster, [6](#)

depths, [7](#)
depths(), [12](#)
dplyr::mutate(), [21](#), [22](#)

envelope_to_voxel, [8](#)
envelope_to_voxel(), [5](#), [11](#), [25](#), [27](#), [30](#), [34](#),
 [38](#), [39](#), [41](#)
extract2d, [15](#)
extract3d_all, [16](#)
extract3d_bottom, [17](#)
extract3d_nearest, [18](#)
extract3d_surface, [19](#)
extract_to_area, [11](#)
extract_to_area(), [31](#)
extract_to_point, [12](#)
extract_to_point(), [11](#), [12](#), [15–19](#)

fetch_species_assessments, [20](#)
fetch_species_assessments(), [21](#)
fill_missing_depths, [21](#)

gfw_effort_to_raster, [22](#)

intersect, SpatEnvelope, SpatEnvelope-method
 (intersect_3d), [24](#)
intersect, SpatEnvelope, SpatRaster-method
 (intersect_3d), [24](#)
intersect, SpatEnvelope, SpatVoxel-method
 (intersect_3d), [24](#)
intersect, SpatRaster, SpatEnvelope-method
 (intersect_3d), [24](#)
intersect, SpatRaster, SpatVoxel-method
 (intersect_3d), [24](#)
intersect, SpatVoxel, SpatEnvelope-method
 (intersect_3d), [24](#)
intersect, SpatVoxel, SpatRaster-method
 (intersect_3d), [24](#)
intersect, SpatVoxel, SpatVoxel-method
 (intersect_3d), [24](#)
intersect-guard(intersect_3d), [24](#)
intersect_3d, [24](#)
intersect_3d(), [5](#), [6](#), [9](#), [26](#), [28](#), [30–32](#), [38](#)
intersect_3d, ANY, ANY-method
 (intersect_3d), [24](#)
intersect_3d, ANY, SpatEnvelope-method
 (intersect_3d), [24](#)
intersect_3d, ANY, SpatVoxel-method
 (intersect_3d), [24](#)
intersect_3d, SpatEnvelope, ANY-method
 (intersect_3d), [24](#)
intersect_3d, SpatEnvelope, SpatEnvelope-method
 (intersect_3d), [24](#)
intersect_3d, SpatEnvelope, SpatVoxel-method
 (intersect_3d), [24](#)
intersect_3d, SpatVoxel, ANY-method
 (intersect_3d), [24](#)
intersect_3d, SpatVoxel, SpatEnvelope-method
 (intersect_3d), [24](#)
intersect_3d, SpatVoxel, SpatVoxel-method
 (intersect_3d), [24](#)
intersects_3d, [26](#)
intersects_3d(), [5](#), [6](#), [24](#), [26](#), [30–32](#)

- intersects_3d, ANY, ANY-method
 (intersects_3d), 26
- intersects_3d, ANY, SpatEnvelope-method
 (intersects_3d), 26
- intersects_3d, ANY, SpatVoxel-method
 (intersects_3d), 26
- intersects_3d, SpatEnvelope, ANY-method
 (intersects_3d), 26
- intersects_3d, SpatEnvelope, SpatEnvelope-method
 (intersects_3d), 26
- intersects_3d, SpatEnvelope, SpatVoxel-method
 (intersects_3d), 26
- intersects_3d, SpatVoxel, ANY-method
 (intersects_3d), 26
- intersects_3d, SpatVoxel, SpatEnvelope-method
 (intersects_3d), 26
- intersects_3d, SpatVoxel, SpatVoxel-method
 (intersects_3d), 26

- load_gebco_bathymetry, 28

- mask(), 24–26, 28, 32
- mask, SpatEnvelope, SpatEnvelope-method
 (mask-3d), 29
- mask, SpatEnvelope, SpatVoxel-method
 (mask-3d), 29
- mask, SpatRaster, SpatEnvelope-method
 (mask-3d), 29
- mask, SpatRaster, SpatVoxel-method
 (mask-3d), 29
- mask, SpatVoxel, SpatEnvelope-method
 (mask-3d), 29
- mask, SpatVoxel, SpatVoxel-method
 (mask-3d), 29
- mask-3d, 11, 29

- occupied, 31

- profile_equal (voxel_profiles), 39
- profile_equal(), 9
- profile_flat (voxel_profiles), 39
- profile_flat(), 9

- SpatEnvelope, 2, 3, 5, 8, 11, 25–27, 29, 30,
 32, 34, 35, 37, 38, 40, 41
- SpatEnvelope-class, 33
- SpatVolume, 33, 35, 38
- SpatVolume-class, 34
- SpatVoxel, 4, 5, 7, 8, 10, 11, 25–27, 29–34,
 37, 38, 40, 41

- SpatVoxel-class, 35

- terra::global(), 8
- terra::mask(), 29, 30
- terra::rast(), 28
- terra::SpatRaster, 7, 8, 33, 35
- tools::R_user_dir(), 43

- vect_to_envelope, 36
- vect_to_envelope(), 2, 3, 37
- volume, 37
- volume(), 5, 6, 26, 32
- volume, SpatEnvelope-method (volume), 37
- volume, SpatRaster-method (volume), 37
- volume, SpatVoxel-method (volume), 37
- voxel_profiles, 9, 39
- voxel_to_envelope, 40
- voxel_to_envelope(), 5, 9, 10, 32, 34

- woa_cache_clear, 42
- woa_cache_dir, 43
- woa_cache_dir(), 43, 44
- woa_download, 43
- woa_load_nc, 45