cw-tiler Documentation

Release 0.2.0

David Lindenbaum, Nick Weir

Dec 12, 2018

Contents:

Author CosmiQ Works

Version 0.2

CHAPTER 1

Sections

CHAPTER 2

Tiling functions

`cw_tiler.main.`**`calculate_analysis_grid`**(*utm_bounds*, *stride_size_meters=300*, *cell_size_meters=400*, *quad_space=False*, *snapToGrid=False*)

Wrapper for `calculate_anchor_points()` and `calculate_cells()`.

Based on UTM boundaries of an image tile, stride size, and cell size, output a dictionary of boundary lists for analysis chips.

Parameters

- **utm_bounds** (list-like of shape `(W, S, E, N)`) – UTM coordinate limits of the input tile.
- **stride_size_meters** (*int, optional*) – Step size in both X and Y directions between cells in units of meters. Defaults to `300` .
- **cell_size_meters** (*int, optional*) – Extent of each cell in both X and Y directions in units of meters. Defaults to `400` .
- **quad_space** (*bool, optional*) – See `calculate_anchor_points()` . `quad_space` . Defaults to `False` .
- **snapToGrid** (*bool, optional*) –

Returns cells_list_dict – A dict whose keys are either `0` or `[0, 1, 2, 3]` (see `calculate_anchor_points()` . `quad_space`), and whose values are `list` s of boundaries in the shape `[W, S, E, N]` . Boundaries are in UTM coordinates.

Return type dict of list(s) of lists

`cw_tiler.main.`**`calculate_anchor_points`**(*utm_bounds*, *stride_size_meters=400*, *extend=False*, *quad_space=False*)

Get anchor point (lower left corner of bbox) for chips from a tile.

Parameters

- **utm_bounds** (*tuple of 4 floats*) – A `tuple` of shape `(min_x, min_y, max_x, max_y)` that defines the spatial extent of the tile to be split. Coordinates should be in UTM.

- **stride_size_meters** (*int, optional*) – Stride size in both X and Y directions for generating chips. Defaults to `400`.
- **extend** (*bool, optional*) – Defines whether UTM boundaries should be rounded to the nearest integer outward from *utm_bounds* (*extend* == `True`) or inward from *utm_bounds* (*extend* == `False`). Defaults to `False` (inward).
- **quad_space** (*bool, optional*) – If tiles will overlap by no more than half their X and/or Y extent in each direction, *quad_space* can be used to split chip anchors into four non-overlapping subsets. For example, if anchor points are 400m apart and each chip will be 800m by 800m, *quad_space* will generate four sets which do not internally overlap; however, this would fail if tiles are 900m by 900m. Defaults to `False`, in which case the returned `anchor_point_list_dict` will comprise a single list of anchor points.

Returns

- **anchor_point_list_dict** (*dict of list(s) of lists*)
- If `quad_space==True` , *anchor_point_list_dict* is a
- `dict` with four keys `[0, 1, 2, 3]` corresponding to the four
- subsets of chips generated (see *quad_space*). If
- `quad_space==False` , *anchor_point_list_dict* is a
- `dict` with a single key, `0` , that corresponds to a list of all
- *of the generated anchor points. Each anchor point in the list(s) is an*
- `[x, y]` pair of UTM coordinates denoting the SW corner of a chip.

`cw_tiler.main.`**calculate_cells**(*anchor_point_list_dict*, *cell_size_meters*, *utm_bounds=[]*)

Calculate boundaries for image cells (chips) from anchor points.

This function takes the output from *`calculate_anchor_points()`* as well as a desired cell size (*cell_size_meters*) and outputs `(W, S, E, N)` tuples for generating cells.

Parameters

- **anchor_point_list_dict** (*dict*) – Output of *`calculate_anchor_points()`*. See that function for details.
- **cell_size_meters** (*int or float*) – Desired width and height of each cell in meters.
- **utm_bounds** (*list -like of float s, optional*) – A `list`-like of shape `(W, S, E, N)` that defines the limits of an input image tile in UTM coordinates to ensure that no cells extend beyond those limits. If not provided, all cells will be included even if they extend beyond the UTM limits of the source imagery.

Returns cells_list_dict – A dict whose keys are either `0` or `[0, 1, 2, 3]` (see *`calculate_anchor_points()`* . `quad_space`), and whose values are `list` s of boundaries in the shape `[W, S, E, N]` . Boundaries are in UTM coordinates.

Return type dict of list(s) of lists

`cw_tiler.main.`**get_chip**(*source*, *ll_x*, *ll_y*, *gsd*, *utm_crs=''*, *indexes=None*, *tilesize=256*, *nodata=None*, *alpha=None*)

Get an image tile of specific pixel size.

This wrapper function permits passing of *ll_x*, *ll_y*, *gsd*, and *tile_size_pixels* in place of boundary coordinates to extract an image region of defined pixel extent.

Parameters

- **source** (rasterio.Dataset) – Source imagery dataset to tile.
- **ll_x** (*int or float*) – Lower left x position (i.e. Western bound).
- **ll_y** (*int or float*) – Lower left y position (i.e. Southern bound).
- **gsd** (*float*) – Ground sample distance of the source imagery in meter/pixel units.
- **utm_crs** (rasterio.crs.CRS, optional) – UTM coordinate reference system string for the imagery. If not provided, this is calculated using *cw_tiler.utils.get_wgs84_bounds()* and *cw_tiler.utils.calculate_UTM_crs()* .
- **indexes** (*tuple of 3 ints, optional*) – Band indexes for the output. By default, extracts all of the indexes from *source*.
- **tilesize** (*int, optional*) – Output image X and Y pixel extent. Defaults to 256 .
- **nodata** (*int or float, optional*) – Value to use for *nodata* pixels during tiling. By default, uses the existing *nodata* value in *source*.
- **alpha** (*int, optional*) – Alpha band index for tiling. By default, uses the same band as specified by *source*.

Returns

data [numpy.ndarray] int pixel values. Shape is (C, Y, X) if retrieving multiple channels, (Y, X) otherwise.

mask [numpy.ndarray] int mask indicating which pixels contain information and which are *nodata*. Pixels containing data have value 255, *nodata* pixels have value 0.

window [rasterio.windows.Window] rasterio.windows.Window object indicating the raster location of the dataset subregion being returned in *data*.

window_transform [affine.Affine] Affine transformation for the window.

Return type (data, mask, window, window_transform tuple.

cw_tiler.main.**tile_utm**(*source*, *ll_x*, *ll_y*, *ur_x*, *ur_y*, *indexes=None*, *tilesize=256*, *nodata=None*, *alpha=None*, *dst_crs='epsg:4326'*)

Create a UTM tile from a file or a rasterio.Dataset in memory.

This function is a wrapper around *tile_utm_source()* to enable passing of file paths instead of pre-loaded rasterio.Dataset s.

Parameters

- **source** (rasterio.Dataset) – Source imagery dataset to tile.
- **ll_x** (*int or float*) – Lower left x position (i.e. Western bound).
- **ll_y** (*int or float*) – Lower left y position (i.e. Southern bound).
- **ur_x** (*int or float*) – Upper right x position (i.e. Eastern bound).
- **ur_y** (*int or float*) – Upper right y position (i.e. Northern bound).
- **indexes** (*tuple of 3 ints, optional*) – Band indexes for the output. By default, extracts all of the indexes from *source* .
- **tilesize** (int, optional) – Output image X and Y pixel extent. Defaults to 256.
- **nodata** (*int or float, optional*) – Value to use for nodata pixels during tiling. By default, uses the existing nodata value in *src*.
- **alpha** (int, optional) – Alpha band index for tiling. By default, uses the same band as specified by *src*.

- **dst_crs** (*str, optional*) – Coordinate reference system for output. Defaults to `"epsg:4326"`.

Returns

data [`numpy.ndarray`] int pixel values. Shape is `(C, Y, X)` if retrieving multiple channels, `(Y, X)` otherwise.

mask [`numpy.ndarray`] int mask indicating which pixels contain information and which are *nodata*. Pixels containing data have value `255`, *nodata* pixels have value `0`.

window [`rasterio.windows.Window`] `rasterio.windows.Window` object indicating the raster location of the dataset subregion being returned in *data*.

window_transform [`affine.Affine`] Affine transformation for the window.

Return type `(data, mask, window, window_transform` tuple.

`cw_tiler.main.`**`tile_utm_source`**(*src, ll_x, ll_y, ur_x, ur_y, indexes=None, tilesize=256, nodata=None, alpha=None, dst_crs='epsg:4326'*)

Create a UTM tile from a `rasterio.Dataset` in memory.

Parameters

- **src** (`rasterio.Dataset`) – Source imagery dataset to tile.
- **ll_x** (*int or float*) – Lower left x position (i.e. Western bound).
- **ll_y** (*int or f*) –
- **loat** – Lower left y position (i.e. Southern bound).
- **ur_x** (*int or float*) – Upper right x position (i.e. Eastern bound).
- **ur_y** (*int or float*) – Upper right y position (i.e. Northern bound).
- **indexes** (*tuple of 3 ints, optional*) – Band indexes for the output. By default, extracts all of the indexes from *src*.
- **tilesize** (*int, optional*) – Output image X and Y pixel extent. Defaults to `256`.
- **nodata** (*int or float, optional*) – Value to use for *nodata* pixels during tiling. By default, uses the existing *nodata* value in *src*.
- **alpha** (*int, optional*) – Alpha band index for tiling. By default, uses the same band as specified by *src*.
- **dst_crs** (*str, optional*) – Coordinate reference system for output. Defaults to `"epsg:4326"`.

Returns

data [`numpy.ndarray`] int pixel values. Shape is `(C, Y, X)` if retrieving multiple channels, `(Y, X)` otherwise.

mask [`numpy.ndarray`] int mask indicating which pixels contain information and which are *nodata*. Pixels containing data have value `255`, *nodata* pixels have value `0`.

window [`rasterio.windows.Window`] `rasterio.windows.Window` object indicating the raster location of the dataset subregion being returned in *data*.

window_transform [`affine.Affine`] Affine transformation for the window.

Return type `(data, mask, window, window_transform)` tuple.

CHAPTER 3

Utility functions

3.1 Raster utilities

cw_tiler.utils: utility functions for raster files.

`cw_tiler.utils.`**`calculate_UTM_crs`**(*coords*)

Calculate UTM Projection String.

Parameters **coords** (*list*) – `[longitude, latitude]` or `[min_longitude, min_latitude, max_longitude, max_latitude]`.

Returns **out** – returns proj4 projection string

Return type str

`cw_tiler.utils.`**`get_utm_bounds`**(*source*, *utm_EPSG*)

Transform bounds from source crs to a UTM crs.

Parameters

- **source** (str or `rasterio.io.DatasetReader`) – Source dataset. Can either be a string path to a dataset GeoTIFF or a `rasterio.io.DatasetReader` object.
- **utm_EPSG** (*str*) – `rasterio.crs.CRS` string indicating the UTM crs to transform into.

Returns **utm_bounds** – Bounding box limits in *utm_EPSG* crs coordinates with shape `(W, S, E, N)`.

Return type tuple

`cw_tiler.utils.`**`get_utm_vrt`**(*source*, *crs='EPSG:3857'*, *resampling=<Resampling.bilinear: 1>*, *src_nodata=None*, *dst_nodata=None*)

Get a `rasterio.vrt.WarpedVRT` projection of a dataset.

Parameters

- **source** (`rasterio.io.DatasetReader`) – The dataset to virtually warp using `rasterio.vrt.WarpedVRT`.

- **crs** (rasterio.crs.CRS, optional) – Coordinate reference system for the VRT. Defaults to 'EPSG:3857' (Web Mercator).
- **resampling** (rasterio.enums.Resampling method, optional) – Resampling method to use. Defaults to `rasterio.enums.Resampling.bilinear()`. Alternatives include `rasterio.enums.Resampling.average()`, `rasterio.enums.Resampling.cubic()`, and others. See docs for `rasterio.enums.Resampling` for more information.
- **src_nodata** (*int or float, optional*) – Source nodata value which will be ignored for interpolation. Defaults to `None` (all data used in interpolation).
- **dst_nodata** (*int or float, optional*) – Destination nodata value which will be ignored for interpolation. Defaults to `None`, in which case the value of *src_nodata* will be used if provided, or `0` otherwise.

Returns

Return type A `rasterio.vrt.WarpedVRT` instance with the transformation.

`cw_tiler.utils.`**`get_utm_vrt_profile`**(*source, crs='EPSG:3857', resampling=<Resampling.bilinear: 1>, src_nodata=None, dst_nodata=None*)

Get a `rasterio.profiles.Profile` for projection of a VRT.

Parameters

- **source** (`rasterio.io.DatasetReader`) – The dataset to virtually warp using `rasterio.vrt.WarpedVRT`.
- **crs** (`rasterio.crs.CRS`, optional) – Coordinate reference system for the VRT. Defaults to `"EPSG:3857"` (Web Mercator).
- **resampling** (`rasterio.enums.Resampling` method, optional) – Resampling method to use. Defaults to `rasterio.enums.Resampling.bilinear`. Alternatives include `rasterio.enums.Resampling.average`, `rasterio.enums.Resampling.cubic`, and others. See docs for `rasterio.enums.Resampling` for more information.
- **src_nodata** (*int or float, optional*) – Source nodata value which will be ignored for interpolation. Defaults to `None` (all data used in interpolation).
- **dst_nodata** (*int or float, optional*) – Destination nodata value which will be ignored for interpolation. Defaults to `None`, in which case the value of *src_nodata* will be used if provided, or `0` otherwise.

Returns

- A `rasterio.profiles.Profile` instance with the transformation
- *applied.*

`cw_tiler.utils.`**`get_wgs84_bounds`**(*source*)

Transform dataset bounds from source crs to wgs84.

Parameters **source** (str or `rasterio.io.DatasetReader`) – Source dataset to get bounds transformation for. Can either be a string path to a dataset file or an opened `rasterio.io.DatasetReader`.

Returns **wgs_bounds** – Bounds tuple for *source* in wgs84 crs with shape `(W, S, E, N)`.

Return type tuple

`cw_tiler.utils.`**`tile_exists_utm`**(*boundsSrc*, *boundsTile*)

Check if suggested tile is within bounds.

Parameters

- **boundsSrc** (*list-like*) – Bounding box limits for the source data in the shape `(W, S, E, N)`.
- **boundsTile** (*list-like*) – Bounding box limits for the target tile in the shape `(W, S, E, N)`.

Returns Do the *boundsSrc* and *boundsTile* bounding boxes overlap?

Return type bool

`cw_tiler.utils.`**`tile_read_utm`**(*source, bounds, tilesize, indexes=[1], nodata=None, alpha=None, dst_crs='EPSG:3857', verbose=False, boundless=False*)

Read data and mask.

Parameters

- **source** (str or `rasterio.io.DatasetReader`) – input file path or `rasterio.io.DatasetReader` object.
- **bounds** (`(W, S, E, N)` tuple) – bounds in *dst_crs* .
- **tilesize** (*int*) – Length of one edge of the output tile in pixels.
- **indexes** (*list of ints or int, optional*) – Channel index(es) to output. Returns a 3D `np.ndarray` of shape (C, Y, X) if *indexes* is a list, or a 2D array if *indexes* is an int channel index. Defaults to `1`.
- **nodata** (*int or float, optional*) – nodata value to use in `rasterio.vrt.WarpedVRT`. Defaults to `None` (use all data in warping).
- **alpha** (*int, optional*) – Force alphaband if not present in the dataset metadata. Defaults to `None` (don't force).
- **dst_crs** (*str, optional*) – Destination coordinate reference system. Defaults to `"EPSG:3857"` (Web Mercator)
- **verbose** (*bool, optional*) – Verbose text output. Defaults to `False`.
- **boundless** (*bool, optional*) – This argument is deprecated and should never be used.

Returns

- **data** (`np.ndarray`) – int pixel values. Shape is `(C, Y, X)` if retrieving multiple channels, `(Y, X)` otherwise.
- **mask** (`np.ndarray`) – int mask indicating which pixels contain information and which are *nodata*. Pixels containing data have value `255`, *nodata* pixels have value `0`.
- **window** (`rasterio.windows.Window`) – `rasterio.windows.Window` object indicating the raster location of the dataset subregion being returned in *data*.
- **window_transform** (`affine.Affine`) – Affine transformation for *window* .

`cw_tiler.utils.`**`utm_getZone`**(*longitude*)

Calculate UTM Zone from Longitude.

Parameters **longitude** (*float*) – longitude coordinate (Degrees.decimal degrees)

Returns **out** – UTM Zone number.

Return type int

`cw_tiler.utils.`**`utm_isNorthern`**(*latitude*)

Determine if a latitude coordinate is in the northern hemisphere.

Parameters **latitude** (*float*) – latitude coordinate (Deg.decimal degrees)

Returns **out** – `True` if *latitude* is in the northern hemisphere, `False` otherwise.

Return type bool

3.2 Vector utilities

`cw_tiler.vector_utils.`**`clip_gdf`**(*gdf*, *poly_to_cut*, *min_partial_perc=0.0*, *geom_type='Polygon'*, *use_sindex=True*)

Clip GDF to a provided polygon.

Note: Clips objects within *gdf* to the region defined by *poly_to_cut*. Also adds several columns to the output:

origarea The original area of the polygons (only used if *geom_type* == `"Polygon"`).

origlen The original length of the objects (only used if *geom_type* == `"LineString"`).

partialDec The fraction of the object that remains after clipping (fraction of area for Polygons, fraction of length for LineStrings.) Can filter based on this by using *min_partial_perc*.

truncated Boolean indicator of whether or not an object was clipped.

Parameters

- **gdf** (`geopandas.GeoDataFrame`) – A `geopandas.GeoDataFrame` of polygons to clip.
- **poly_to_cut** (`shapely.geometry.Polygon`) – The polygon to clip objects in *gdf* to.
- **min_partial_perc** (*float, optional*) – The minimum fraction of an object in *gdf* that must be preserved. Defaults to 0.0 (include any object if any part remains following clipping).
- **geom_type** (*str, optional*) – Type of objects in *gdf*. Can be one of `["Polygon", "LineString"]` . Defaults to `"Polygon"` .
- **use_sindex** (*bool, optional*) – Use the *gdf* sindex be used for searching. Improves efficiency but requires libspatialindex .

Returns **cutGeoDF** – *gdf* with all contained objects clipped to *poly_to_cut* . See notes above for details on additional clipping columns added.

Return type `geopandas.GeoDataFrame`

`cw_tiler.vector_utils.`**`rasterize_gdf`**(*gdf*, *src_shape*, *burn_value=1*, *src_transform=Affine(1.0, 0.0, 0.0, 0.0, 1.0, 0.0)*)

Convert a GeoDataFrame to a binary image (array) mask.

Uses `rasterio.features.rasterize()` to generate a raster mask from object geometries in *gdf* .

Parameters

- **gdf** (`geopandas.GeoDataFrame`) – A `geopandas.GeoDataFrame` of objects to convert into a mask.

- **src_shape** (*list-like of 2 ints*) – Shape of the output array in `(Y, X)` pixel units.
- **burn_value** (*int in range(0, 255), optional*) – Integer value for pixels corresponding to objects from *gdf* . Defaults to 1.
- **src_transform** (`affine.Affine`, optional) – Affine transformation for the output raster. If not provided, defaults to arbitrary pixel units.

Returns **img** – A NumPy array of integers with 0s where no pixels from objects in *gdf* exist, and *burn_value* where they do. Shape is defined by *src_shape*.

Return type `np.ndarray`, dtype `uint8`

`cw_tiler.vector_utils.`**`read_vector_file`**(*geoFileName*)

Read Fiona-Supported Files into GeoPandas GeoDataFrame.

> **Warning:** This will raise an exception for empty GeoJSON files, which GDAL and Fiona cannot read. `try/except` the `Fiona.errors.DriverError` or `Fiona._err.CPLE_OpenFailedError` if you must use this.

`cw_tiler.vector_utils.`**`search_gdf_bounds`**(*gdf*, *tile_bounds*)

Use *tile_bounds* to subset *gdf* and return the intersect.

Parameters

- **gdf** (`geopandas.GeoDataFrame`) – A `geopandas.GeoDataFrame` of polygons to subset.
- **tile_bounds** (*tuple*) – A tuple of shape `(W, S, E, N)` that denotes the boundaries of a tile.

Returns **smallGdf** – The subset of *gdf* that overlaps with *tile_bounds* .

Return type `geopandas.GeoDataFrame`

`cw_tiler.vector_utils.`**`search_gdf_polygon`**(*gdf*, *tile_polygon*)

Find polygons in a GeoDataFrame that overlap with *tile_polygon* .

Parameters

- **gdf** (`geopandas.GeoDataFrame`) – A `geopandas.GeoDataFrame` of polygons to search.
- **tile_polygon** (`shapely.geometry.Polygon`) – A `shapely.geometry.Polygon` denoting a tile's bounds.

Returns **precise_matches** – The subset of *gdf* that overlaps with *tile_polygon* . If there are no overlaps, this will return an empty `geopandas.GeoDataFrame`.

Return type `geopandas.GeoDataFrame`

`cw_tiler.vector_utils.`**`transformToUTM`**(*gdf*, *utm_crs*, *estimate=True*, *calculate_sindex=True*)

Transform GeoDataFrame to UTM coordinate reference system.

Parameters

- **gdf** (`geopandas.GeoDataFrame`) – `geopandas.GeoDataFrame` to transform.
- **utm_crs** (*str*) – `rasterio.crs.CRS` string for destination UTM CRS.
- **estimate** (*bool, optional*) – Deprecated since version 0.2.0: This argument is no longer used.

- **calculate_sindex** (*bool, optional*) – Deprecated since version 0.2.0: This argument is no longer used.

Returns **gdf** – The input geopandas.GeoDataFrame converted to *utm_crs* coordinate reference system.

Return type geopandas.GeoDataFrame

cw_tiler.vector_utils.**vector_tile_utm**(*gdf*, *tile_bounds*, *min_partial_perc=0.1*, *geom_type='Polygon'*, *use_sindex=True*)

Wrapper for *clip_gdf()* that converts *tile_bounds* to a polygon.

Parameters

- **gdf** (geopandas.GeoDataFrame) – A geopandas.GeoDataFrame of polygons to clip.
- **tile_bounds** (*list-like of floats*) – list of shape (W, S, E, N) denoting the boundaries of an imagery tile. Converted to a polygon for *clip_gdf()*.
- **min_partial_perc** (*float*) – The minimum fraction of an object in *gdf* that must be preserved. Defaults to 0.0 (include any object if any part remains following clipping).
- **use_sindex** (*bool, optional*) – Use the *gdf* sindex be used for searching. Improves efficiency but requires libspatialindex .

Returns **small_gdf** – *gdf* with all contained objects clipped to *tile_bounds*. See notes above for details on additional clipping columns added.

Return type geopandas.GeoDataFrame

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index