yt_georaster

Release 1.0.0.dev1

Team yt_georaster

Nov 28, 2022

CONTENTS:

`yt_georaster` is a yt extension for analyzing geotagged image files that are loadable with rasterio. The `yt_georaster` extension combines `yt` and `rasterio`, allowing users to query data contained within geometric shapes, like circles, rectangles, and arbitrary polygons saved as Shapefiles. Multiple images with different resolutions and coordinate reference systems can be loaded together. All queried data is transformed to the same resolution and coordinate reference system and returned as NumPy arrays of the same shape. Data from multiple images can also be re-saved to a single, multiband GeoTIFF file.

CHAPTER

ONE

ADDITIONAL RESOURCES

- rasterio documentation
- yt documentation

1.1 Help

If you encounter problems, we want to help and there are lots of places to get help. As an extension of the yt project, we are members of the yt community. There is a dedicated `#yt_georaster` channel on the yt project Slack and questions can also be posted to the yt users mailing list. Bugs and feature requests can also be posted on the ytree issues page.

See you out there!

1.2 Installation

At present, `yt_georaster` can only be installed from source. To do this, one must clone the repository and use `pip` or `conda` to install. This process is detailed below. the `yt_georaster` repository lives at https://github.com/ruithnadsteud/yt_georaster.

1.2.1 Dependencies

It is recommended to use a package manager, such as conda (miniconda, anaconda, etc.) to install packages. Most of the `yt_georaster's` dependencies can be installed by `pip` or `conda` automatically while installing `yt_georaster`. There are a few notable exceptions.

yt

`yt_georaster` requires `yt` version 4.0, which is due to be release imminently, but is not out yet. Thus, `yt` must be installed from source in a manner similar to `yt_georaster`. This process is detailed here and shown in brief below.

```
$ git clone https://github.com/yt-project/yt
$ cd yt
$ pip install -e .
```

gdal

The gdal library is the main dependency of rasterio. Depending on your operating system, it can be difficult to install, but there are many options, including conda and pip. If you had a particularly difficult time with this but eventually succeeded, please consider documenting your setup here.

rasterio

rasterio is marginally easier to install than gdal and has similar options. Add your success story here so others can benefit!

1.2.2 Installing from Source

Ok, you've made it this far. Great job. To install yt_georaster from source, do the following:

```
$ git clone https://github.com/ruithnadsteud/yt_georaster
$ cd yt_georaster
$ pip install -e .
```

You're ready to go.

1.3 Loading Data

The yt_georaster extension provides support for all file types that are loadable by rasterio, including GeoTIFF and JPEG2000. For everything to work correctly, the image files must include geotagging metadata, such as the coordinate reference system (CRS) and transform. Consult the rasterio documentation for more information.

The yt load() function can be used to load all supported data. The relevant yt documentation is here. This typically takes the form:

```
>>> import yt
>>> ds = yt.load(filename)
```

To load data supported by yt_georaster, just add import yt.extensions.georaster to your script.

```
>>> import yt
>>> import yt.extensions.georaster
>>> ds = yt.load(filename)
```

1.3.1 Loading a Single Image

To load a file or files, provide the paths as individual arguments to yt.load.

```
>>> import yt
>>> import yt.extensions.georaster
>>> ds = yt.load("200km_2p5m_N38E34.tif")
yt : [INFO     ] 2021-06-29 12:37:13,171 Parameters: domain_dimensions         = [80000␣
↪80000     1]
yt : [INFO     ] 2021-06-29 12:37:13,174 Parameters: domain_left_edge          =␣
↪[3444000. 3642000.       0.] m
```

(continues on next page)

(continued from previous page)

```
yt : [INFO     ] 2021-06-29 12:37:13,174 Parameters: domain_right_edge         = [3.
↪644e+06 3.842e+06 1.000e+00] m
```

Upon loading, `yt` will print the dimensions (i.e., the number of pixels in x and y) and coordinates of the left and right corners of the image in the loaded image's CRS. Because `yt` mainly deals with 3D data, the dimensions will be reported with a value of 1 in the z direction and values of 0 m and 1 m for the coordinates of the left and right corners. Do not worry; nothing bad will happen because of this.

The values printed after loading are attributes associated with the loaded dataset (i.e., the `ds`). Similarly, one can access the resolution of image.

```
>>> print (ds.domain_dimensions)
[80000 80000     1]
>>> print (ds.domain_right_edge.to('km'))
[3.644e+03 3.842e+03 1.000e-03] km
>>> print (ds.resolution)
unyt_array([2.5, 2.5], 'm')
```

Note that values that should have units are returned as unyt_arrays. See this discussion in yt for more information about quantities with units.

1.3.2 Loading Multiple Images

Multiple images can be loaded into the same dataset by providing each file path to `yt.load`.

```
>>> import yt
>>> import yt.extensions.georaster

>>> filenames = glob.glob("Landsat-8_sample_L2/*.TIF") + \
...   glob.glob("M2_Sentinel-2_test_data/*.jp2")
>>> ds = yt.load(*filenames)
yt : [INFO     ] 2021-06-29 13:19:24,134 Parameters: domain_dimensions         = [7581␣
↪7741    1]
yt : [INFO     ] 2021-06-29 13:19:24,134 Parameters: domain_left_edge          = [␣
↪361485. -116415.       0.] m
yt : [INFO     ] 2021-06-29 13:19:24,135 Parameters: domain_right_edge         = [5.
↪88915e+05 1.15815e+05 1.00000e+00] m
```

Note, the argument to `yt.load` is `*filenames` and not just `filenames`. This expands the list into its individual items.

The Base Image

When loading multiple images, the information printed upon load is associated with the first argument given. This image is referred to in this document as the **base image**. All queried data will be returned in the resolution and CRS of the base image. **Only data within the bounds of the base image can be queried.** Printing either `ds` or `ds.parameter_filename` will tell you what the base image is.

```
>>> print (ds)
LC08_L2SP_171060_20210227_20210304_02_T1_QA_PIXEL
>>> print (ds.parameter_filename)
Landsat-8_sample_L2/LC08_L2SP_171060_20210227_20210304_02_T1_QA_PIXEL.TIF
```

1.3.3 Specifying your Coordinate Reference System

At load you can also specify what coordinate reference system (CRS) you want to handle your dataset in.

```
>>> import yt
>>> import yt.extensions.georaster

>>> filenames = glob.glob("Landsat-8_sample_L2/*.TIF") + \
...   glob.glob("M2_Sentinel-2_test_data/*.jp2")
>>> ds = yt.load(*filenames, crs="epsg:32736")
```

This should work for all projected systems. Instead of using the CRS of your base image the dataset is assigned the CRS you provide and yt will convert everything into this coordinate reference system as you query that data.

1.4 Data Fields

Data access in `yt` is built on the concept of fields. The user creates a *geometric data container* (e.g., a circle, rectangle, or polygon) and queries a field (i.e., data from a specific file or band) by name. Data is returned as arrays of all pixels inside the container.

1.4.1 Fields on Disk

To see a list of all fields on disk, have a look at the `field_list` attribute associated with any loaded dataset.

```
>>> import yt
>>> import yt.extensions.georaster

>>> ds = yt.load("circle.tif")
yt : [INFO     ] 2021-06-29 14:47:28,378 Parameters: domain_dimensions         = [1549␣
↪1549    1]
yt : [INFO     ] 2021-06-29 14:47:28,378 Parameters: domain_left_edge          = [451965.
↪ -23535.      0.] m
yt : [INFO     ] 2021-06-29 14:47:28,379 Parameters: domain_right_edge         = [4.
↪98435e+05 2.29350e+04 1.00000e+00] m

>>> print (ds.field_list)
[('circle', 'band_1'), ('circle', 'band_2'), ('circle', 'band_3'), ('circle', 'band_4')]
```

Fields in `yt` are named with tuples, where the first item is typically referred to as the "field type" and the second as the "field name". For a generic rasterio-loadable image file, the field type will be the name of the file (minus the extension) and the fields will be named "band_" and the band number. To see the list of available field types, check out the `fluid_types` attribute.

```
>>> print (ds.fluid_types)
('index', 'circle')
```

The "index" fluid type is for accessing fields associated with the position and size of pixels, such as "x", "y", and "area".

1.4.2 Satellite-Specific Fields and Aliases

`yt_georaster` can recognize naming conventions of data products from a few different satellites. If a loaded file matches the naming conventions of a supported satellite, the fields will be named with the field type corresponding to the unique image identifier (usually the first part of the filename before the name of the band) and the field names as a combination of the satellite type, band number, and resolution. For example, a Sentinel-2 image from band 02 with a resolution of 10 meters will be named "S2_B02_10m".

For every band field, an alias field will be created pointing to the highest resolution image available. In the example below, we load two Sentinel-2 images of band 2 at resolutions of 10 and 20 meters.

```
>>> filenames = glob.glob("Sentinel-2_sample_L2A/T30UVG_20200601T113331_*.jp2")
>>> ds = yt.load(*filenames)

>>> print (ds.fluid_types)
('T30UVG_20200601T113331', 'index')

>>> print (ds.field_list)
[('T30UVG_20200601T113331', 'S2_B01_60m'),
 ('T30UVG_20200601T113331', 'S2_B02_10m'),
 ('T30UVG_20200601T113331', 'S2_B02_20m'),
 ...
]

>>> print (ds.fields.T30UVG_20200601T113331.S2_B02_10m)
On-Disk Field (T30UVG_20200601T113331, S2_B02_10m): (units: )
>>> print (ds.fields.T30UVG_20200601T113331.S2_B02_20m)
On-Disk Field (T30UVG_20200601T113331, S2_B02_20m): (units: )

>>> print (ds.fields.T30UVG_20200601T113331.S2_B02)
Alias Field for "('T30UVG_20200601T113331', 'S2_B02_10m')" (T30UVG_20200601T113331, S2_
↪B02): (units: )
```

These aliases will be made for all bands, even if only a single resolution is available.

Landsat-8

Landsat-8 fields are prefaced with "L8_". Click around enough on this Landsat-8 mission page and you'll find the band definitions in a pdf file. These have been aliased as the following:

Band	Aliases
B1	visible_1
B2	visible_2
B3	visible_3
B4	red
B5	nir
B6	swir_1
B7	swir_2
B8	pan
B9	cirrus
B10	tirs_1
B11	tirs_2

```
>>> filenames = glob.glob("Landsat-8_sample_L2/LC08_L2SP_171060_20210227_20210304_02_T1*.
↪TIF")
>>> ds = yt.load(*filenames)
yt : [INFO     ] 2021-06-29 16:57:21,839 Parameters: domain_dimensions         = [7581␣
↪7741    1]
yt : [INFO     ] 2021-06-29 16:57:21,839 Parameters: domain_left_edge          = [␣
↪361485. -116415.       0.] m
yt : [INFO     ] 2021-06-29 16:57:21,840 Parameters: domain_right_edge         = [5.
↪88915e+05 1.15815e+05 1.00000e+00] m

>>> print (ds.fields.LC08_L2SP_171060_20210227_20210304_02_T1.red)
Alias Field for "('LC08_L2SP_171060_20210227_20210304_02_T1', 'L8_B4')" (LC08_L2SP_
↪171060_20210227_20210304_02_T1, red): (units: )
```

Sentinel-2

Sentinel-2 fields are prefaced with “S2_”. Bands are defined, for example, here and here. These have been aliased as the following:

Band	Aliases
B01	ultra_blue
B02	blue
B03	green
B04	red
B05	vnir_1, red_edge_1
B06	vnir_2, red_edge_2
B07	vnir_3
B08	vnir_4
B8A	vnir_5, nir
B09	swir_1
B10	swir_2
B11	swir_3
B12	swir_4

```
>>> filenames = glob.glob("Sentinel-2_sample_L2A/T30UVG_20200601T113331_*.jp2")
>>> ds = yt.load(*filenames)

>>> print (ds.fields.T30UVG_20200601T113331.red)
Alias Field for "('T30UVG_20200601T113331', 'S2_B04')" (T30UVG_20200601T113331, red):␣
↪(units: )
```

1.4.3 Derived Fields

In addition to the fields on disk, yt_georaster defines a series of "derived fields", which are arithmetic combinations of other existing fields. The full list of available derived fields can be seen by inspecting the derived_field_list attribute associated with the loaded dataset. This list is quite long and includes things not specifically relevant to yt_georaster (as they are defined within yt itself). Those specific to yt_georaster are listed below. Each of the fields below will exist for every field type that defines all the required fields.

To see how each derived field is defined, use the get_source function.

```
>>> print (ds.fields.T30UVG_20200601T113331.NDWI.get_source())
            def _ndwi(field, data):
                ftype = field.name[0]
                green = data[ftype, "green"]
                nir = data[ftype, "nir"]
                return (green - nir) / (green + nir)
```

For more information defining new derived fields, see creating-derived-fields. In the table below, <field type> refers to any loaded satellite data for which the required bands are available.

Field	Description
("index", "area")	pixel area (dx*dy)
(<field type>, "NDWI")	Normalised difference water index
(<field type>, "MCI")	Maximum chlorophyll index
(<field type>, "CDOM")	Colored Dissolved Organic Matter
(<field type>, "EVI")	Enhanced Vegetation Index
(<field type>, "NDVI")	Normalised Difference Vegetation Index
(<field type>, "LS_temperature")	Landsat Surface Temperature

1.5 Data Containers

Data access in yt is facilitated through geometric data containers. A user defines a shape in terms of its position and size. The object created will return data for all pixels contained within the shape. Since yt was designed primarily for 3D data, the native data containers (see Data-objects) are things like spheres, cylinders, and rectangular prisms. yt_georaster offers three 2D containers: *Circles*, *Rectangles*, and *Polygons*. The circle and rectangle containers are thing wrappers around the 3D cylinder and rectangular prism containers. Data containers will return NumPy arrays of *Data Fields* for all pixels inside the container. They can also be used with *Plotting Data*.

There are a number of other interesting things that can be done with them. See, for example, the links below:

- derived-quantities
- filtering-data
- cut-regions

1.5.1 Data Resampling and Transforming

`yt_georaster` supports loading images with different resolutions and coordinate reference systems (see *Loading Multiple Images*). When data is queried with a data container, it is resampled to the resolution of *The Base Image* and transformed into the CRS of *The Base Image*.

Data cannot be queried outside the bounds of the base image. However, data **can** be queried outside of secondary (i.e., not the base) images. All pixels outside a secondary image will be returned as zeros.

1.5.2 Circles

A `circle()` is defined by a center and radius.

```
>>> import yt
>>> import yt.extensions.georaster

>>> filenames = glob.glob("Landsat-8_sample_L2/*.TIF") + \
...   glob.glob("M2_Sentinel-2_test_data/*.jp2")
>>> ds = yt.load(*filenames)
yt : [INFO     ] 2021-06-29 13:19:24,134 Parameters: domain_dimensions         = [7581␣
↪7741    1]
yt : [INFO     ] 2021-06-29 13:19:24,134 Parameters: domain_left_edge          = [␣
↪361485. -116415.       0.] m
yt : [INFO     ] 2021-06-29 13:19:24,135 Parameters: domain_right_edge         = [5.
↪88915e+05 1.15815e+05 1.00000e+00] m

>>> # an array with units
>>> center = ds.arr([500, 0], "km")
>>> # a single value with units
>>> radius = ds.quan(10, "km")

>>> cir = ds.circle(center, radius)
```

Field data is access by querying the data container like a dictionary.

```
>>> print (cir["LC08_L2SP_171060_20210227_20210304_02_T1", "L8_B2"])
unyt_array([8956., 8974., 8980., ..., 7541., 7550., 7493.], '(dimensionless)')

>>> print (cir["index", "area"].sum().to("m**2"))
314156700.00000006 m**2
```

Data is returned a unyt array, a subclass of the NumPy array supporting symbolic units. The raw NumPy array can be accessed by appending `.d`.

```
>>> cir["LC08_L2SP_171060_20210227_20210304_02_T1", "L8_B2"].d
array([8956., 8974., 8980., ..., 7541., 7550., 7493.])
```

1.5.3 Rectangles

A `rectangle()` is defined by the coordinates of the left and right corners. Note, the values of the right corner must be greater than the left corner. A `rectangle_from_center()` can also be defined by a center, width, and height.

1.5.4 Polygons

`yt_georaster` supports arbitrary polygons loaded from Shapefiles. **Currently, the shape must be in the CRS of the base image.** A `YTPolygon` object is created by specifying the path to the shapefile.

```
>>> poly = ds.polygon("example_polygon_mabira_forest/mabira_forest.shp")
>>> print (poly["LC08_L2SP_171060_20210227_20210304_02_T1", "red"])
unyt_array([ 8324.,  8340.,  8372., ..., 10422., 10536., 10333.], '(dimensionless)')

>>> print (poly["index", "area"].sum())
331.2063 km**2
```

Note: The current implementation of the polygon container considers any cell overlapping the polygon boundary to be "contained" within the polygon. Polygon data containers are implemented with the `shapely` package using `within`. This can be modified to include only pixels whose centers are inside the polygon by using the `intersects` class method instead.

1.5.5 Data from the Base Image

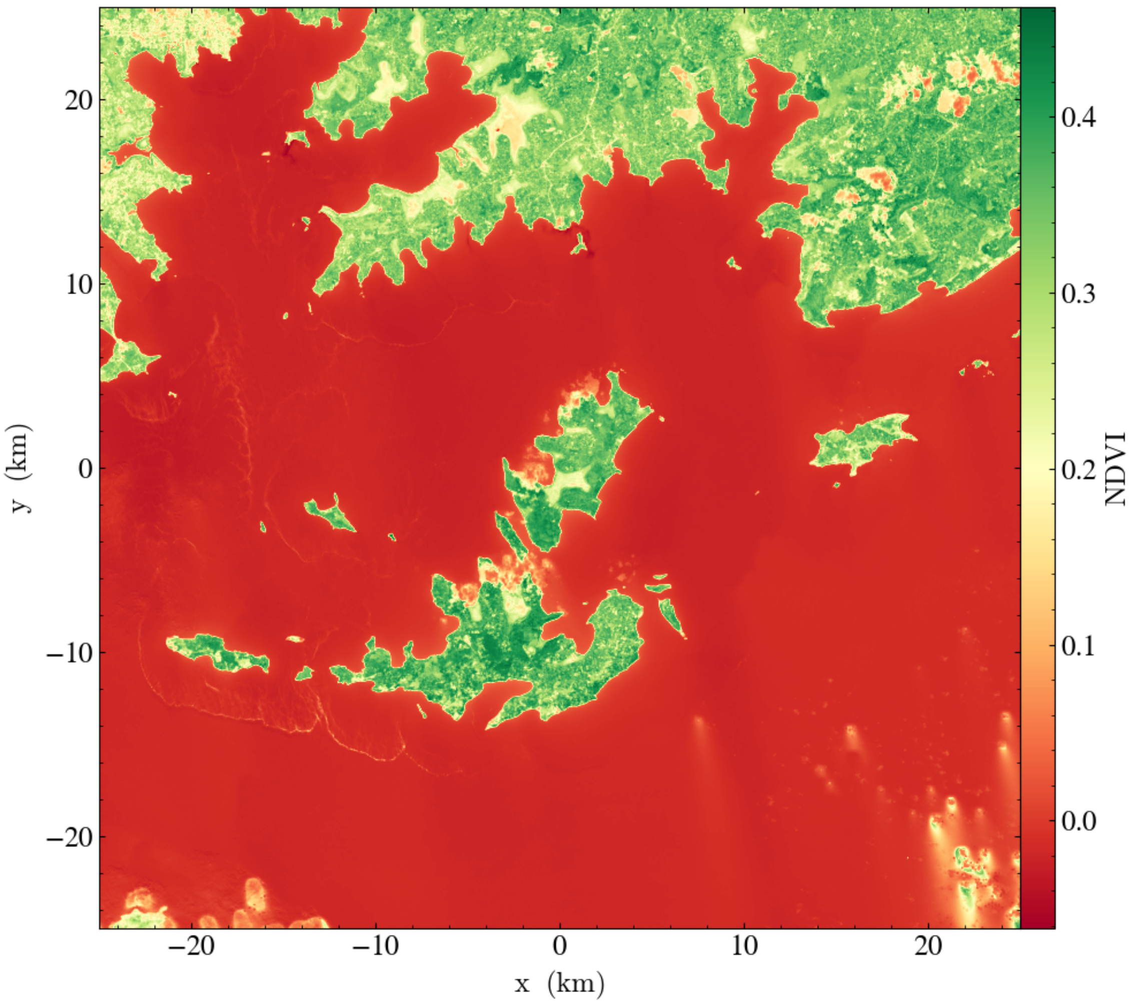
In addition to geometric shapes, data can be queried for 2D grid representing *The Base Image*. This will return data as 2D arrays (technically, 3D arrays with the last third dimension of size 1) corresponding to the dimensions of the base image. This is done by accessing the `data` attribute.

```
>>> import glob
>>> import yt
>>> import yt.extensions.georaster

>>> fns = glob.glob("M2_Sentinel-2_test_data/*.jp2")
>>> ds = yt.load(*fns)
yt : [INFO     ] 2021-06-30 10:34:46,490 Parameters: domain_dimensions         = [1830␣
↪1830    1]
yt : [INFO     ] 2021-06-30 10:34:46,490 Parameters: domain_left_edge          = [␣
↪399960. 9890200.       0.] m
yt : [INFO     ] 2021-06-30 10:34:46,491 Parameters: domain_right_edge         = [5.
↪0976e+05 1.0000e+07 1.0000e+00] m

>>> print (ds.data["T36MVE_20210315T075701", "NDWI"].shape)
yt : [INFO     ] 2021-06-30 10:34:58,748 Resampling ('T36MVE_20210315T075701', 'S2_B03_
↪10m'): 10.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 10:35:00,706 Resampling ('T36MVE_20210315T075701', 'S2_B8A_
↪20m'): 20.0 to 60.0 m.
(1830, 1830, 1)
```

1.6 Plotting Data

yt offers a lot of functionality for plotting. See how-to-make-plots for a full discussion. However, the spatial plots are geared mainly to 3D data, so yt_georaster provides a simple interface to making spatial plots of 2D data. The plot() function will plot a single field, accepting optional arguments for a center, width, and height. If you are working in a Jupyter notebook, use p.show() to display the plot inline.

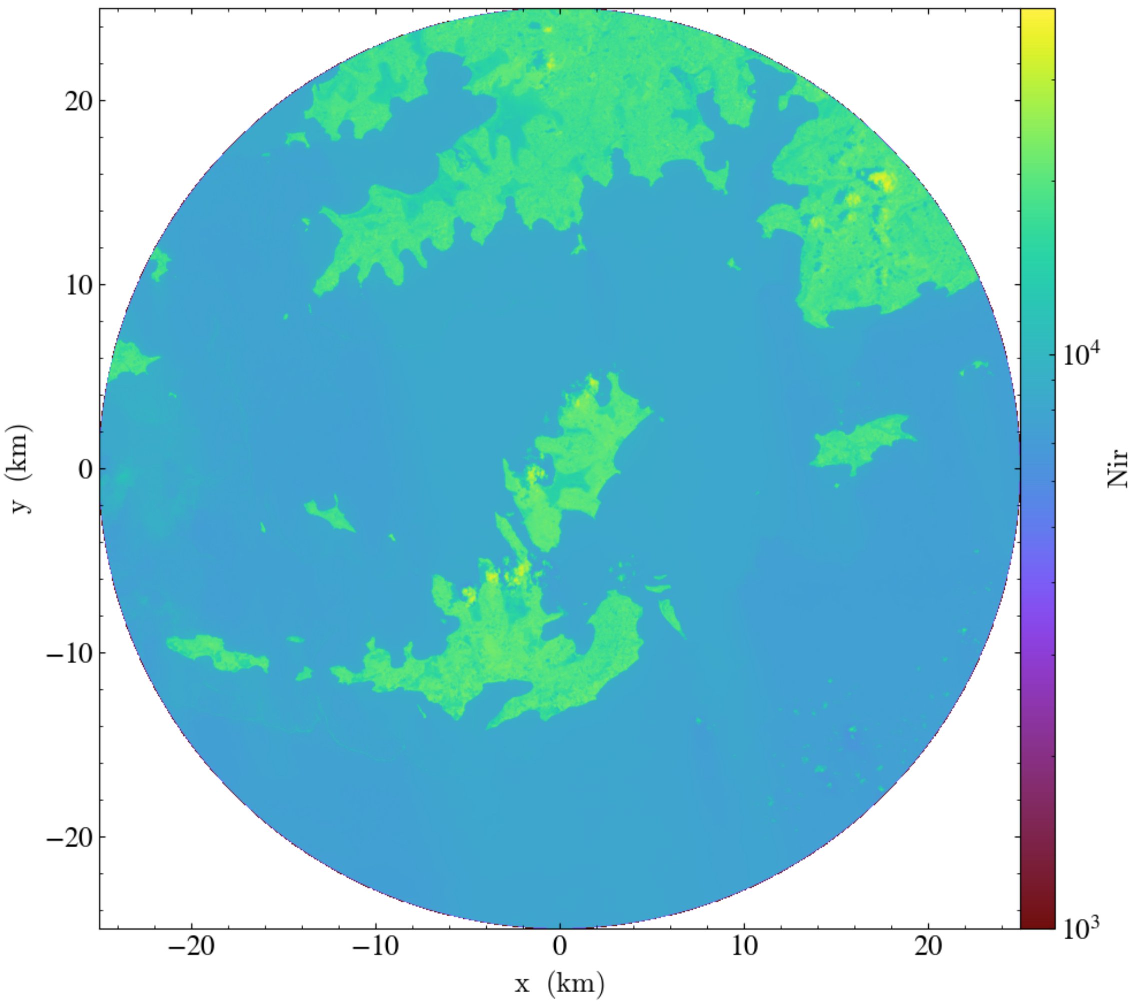
```
>>> import yt
>>> import yt.extensions.georaster

>>> filenames = glob.glob("Landsat-8_sample_L2/*.TIF") + \
...   glob.glob("M2_Sentinel-2_test_data/*.jp2")
>>> ds = yt.load(*filenames)

>>> field = ("LC08_L2SP_171060_20210227_20210304_02_T1", "NDVI")
>>> p = ds.plot(field, center=ds.domain_center,
...             width=(50, "km"), height=(50, "km"))
>>> p.set_cmap(field, "RdYlGn")
>>> p.save("plot_1.png")
```



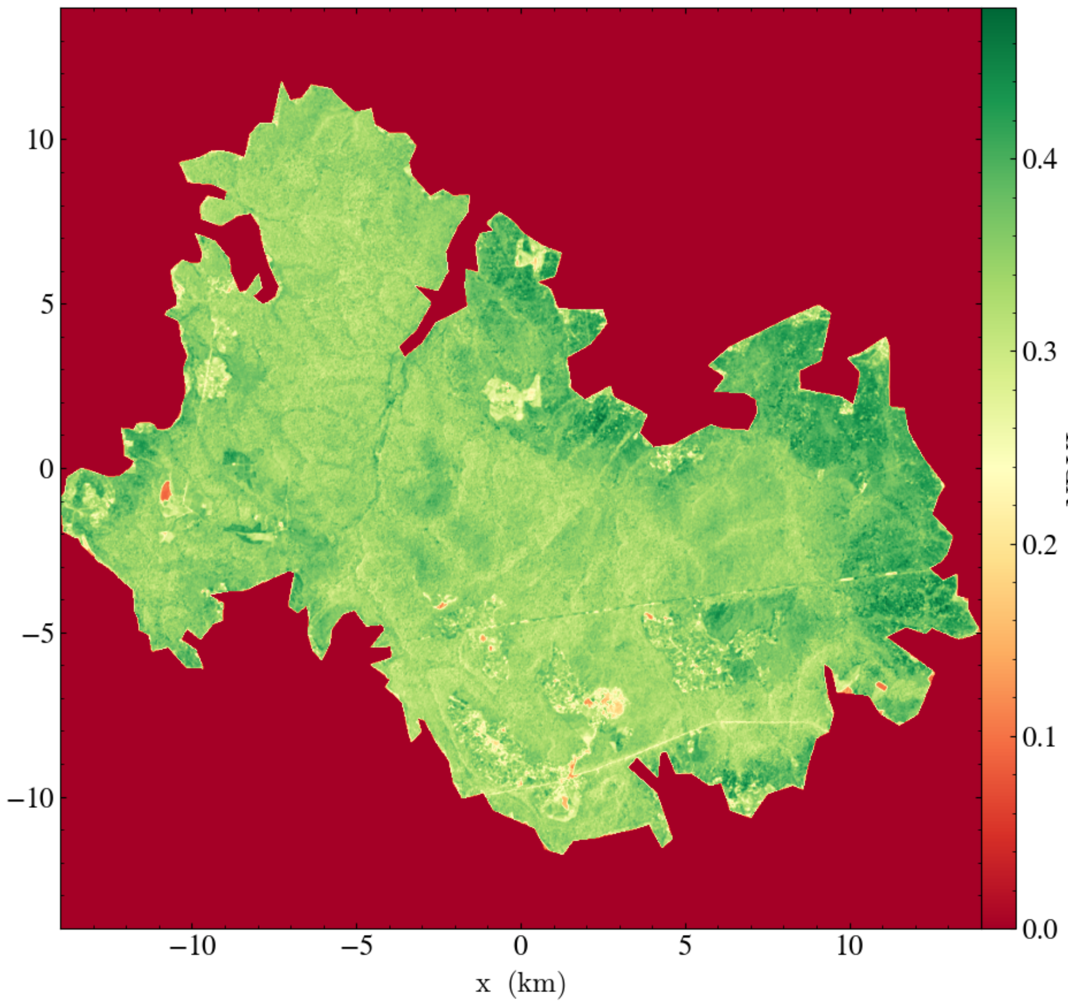
1.6.1 Plotting Data Containers

The `plot()` function also accepts a `data_source` keyword to plot data only within the container. The center, width, and heigh will be set automatically base on the container.

```
>>> cir = ds.circle(ds.domain_center, (25, "km"))
>>> field = ('LC08_L2SP_171060_20210227_20210304_02_T1', "nir")
>>> p = ds.plot(field, data_source=cir)
>>> p.set_zlim(field, 1000, 40000)
>>> p.set_axes_unit('km')
>>> p.save("plot_2.png")
```



```
>>> poly = ds.polygon("example_polygon_mabira_forest/mabira_forest.shp")
>>> field = ("LC08_L2SP_171060_20210227_20210304_02_T1", "NDVI")
>>> p = ds.plot(field, data_source=poly)
>>> p.set_cmap(field, "RdYlGn"))
>>> p.set_axes_unit("km")
>>> p.save("plot_3.png")
```



1.7 Saving GeoTIFFs

The `save_as_geotiff()` function will save loaded images to a single, multiband GeoTIFF file. By default, all loaded on-disk fields will be saved for the region spanning *The Base Image*. An optional list of fields (including aliases and derived fields) can be given to save a subset of all fields. Additionally, a data container can be provided with the `data_source` keyword to save only pixels inside the container.

```
>>> import yt
>>> import yt.extensions.georaster

>>> filenames = glob.glob("M2_Sentinel-2_test_data/*.jp2") + \
...   glob.glob("Landsat-8_sample_L2/*.TIF") + \
>>> ds = yt.load(*filenames)
yt : [INFO     ] 2021-06-30 14:32:43,143 Parameters: domain_dimensions         = [1830␣
↪1830    1]
yt : [INFO     ] 2021-06-30 14:32:43,143 Parameters: domain_left_edge          = [␣
↪399960. 9890200.       0.] m
```

(continues on next page)

(continued from previous page)

```
yt : [INFO     ] 2021-06-30 14:32:43,143 Parameters: domain_right_edge         = [5.
↪0976e+05 1.0000e+07 1.0000e+00] m

>>> circle = ds.circle(ds.domain_center, (20, "km"))
>>> fields = [("T36MVE_20210315T075701", "S2_B06"),
...           ("T36MVE_20210315T075701", "NDWI"),
...           ("LC08_L2SP_171060_20210227_20210304_02_T1", "L8_B1"),
...           ("LC08_L2SP_171060_20210227_20210304_02_T1", "LS_temperature")]

>>> ds_fn, fm_fn = save_as_geotiff(
...     ds, "my_data.tif",
...     fields=fields, data_source=circle)
yt : [INFO     ] 2021-06-30 14:32:43,556 Saving 4 fields to my_data.tif.
yt : [INFO     ] 2021-06-30 14:32:43,556 Bounding box: [ 434820. 9925060.] m m - [␣
↪474900. 9965140.] m m with shape (668, 668).
yt : [INFO     ] 2021-06-30 14:32:43,575 Saving ('LC08_L2SP_171060_20210227_20210304_02_
↪T1', 'L8_B1') to band 1/4.
yt : [INFO     ] 2021-06-30 14:32:43,689 Resampling ('LC08_L2SP_171060_20210227_20210304_
↪02_T1', 'L8_B1_30m'): 30.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 14:32:43,836 Saving ('T36MVE_20210315T075701', 'S2_B06') to␣
↪band 2/4.
yt : [INFO     ] 2021-06-30 14:32:44,174 Resampling ('T36MVE_20210315T075701', 'S2_B06_
↪20m'): 20.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 14:32:44,535 Saving ('T36MVE_20210315T075701', 'NDWI') to␣
↪band 3/4.
yt : [INFO     ] 2021-06-30 14:32:45,599 Resampling ('T36MVE_20210315T075701', 'S2_B03_
↪10m'): 10.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 14:32:45,950 Resampling ('T36MVE_20210315T075701', 'S2_B8A_
↪20m'): 20.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 14:32:46,987 Resampling ('T36MVE_20210315T075701', 'S2_B03_
↪10m'): 10.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 14:32:47,337 Resampling ('T36MVE_20210315T075701', 'S2_B8A_
↪20m'): 20.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 14:32:47,386 Saving ('LC08_L2SP_171060_20210227_20210304_02_
↪T1', 'LS_temperature') to band 4/4.
yt : [INFO     ] 2021-06-30 14:32:47,504 Resampling ('LC08_L2SP_171060_20210227_20210304_
↪02_T1', 'L8_B10_30m'): 30.0 to 60.0 m.
yt : [INFO     ] 2021-06-30 14:32:47,684 Field map saved to my_data_fields.yaml.
yt : [INFO     ] 2021-06-30 14:32:47,684 Save complete. Reload data with:
ds = yt.load("my_data.tif", field_map="my_data_fields.yaml")
```

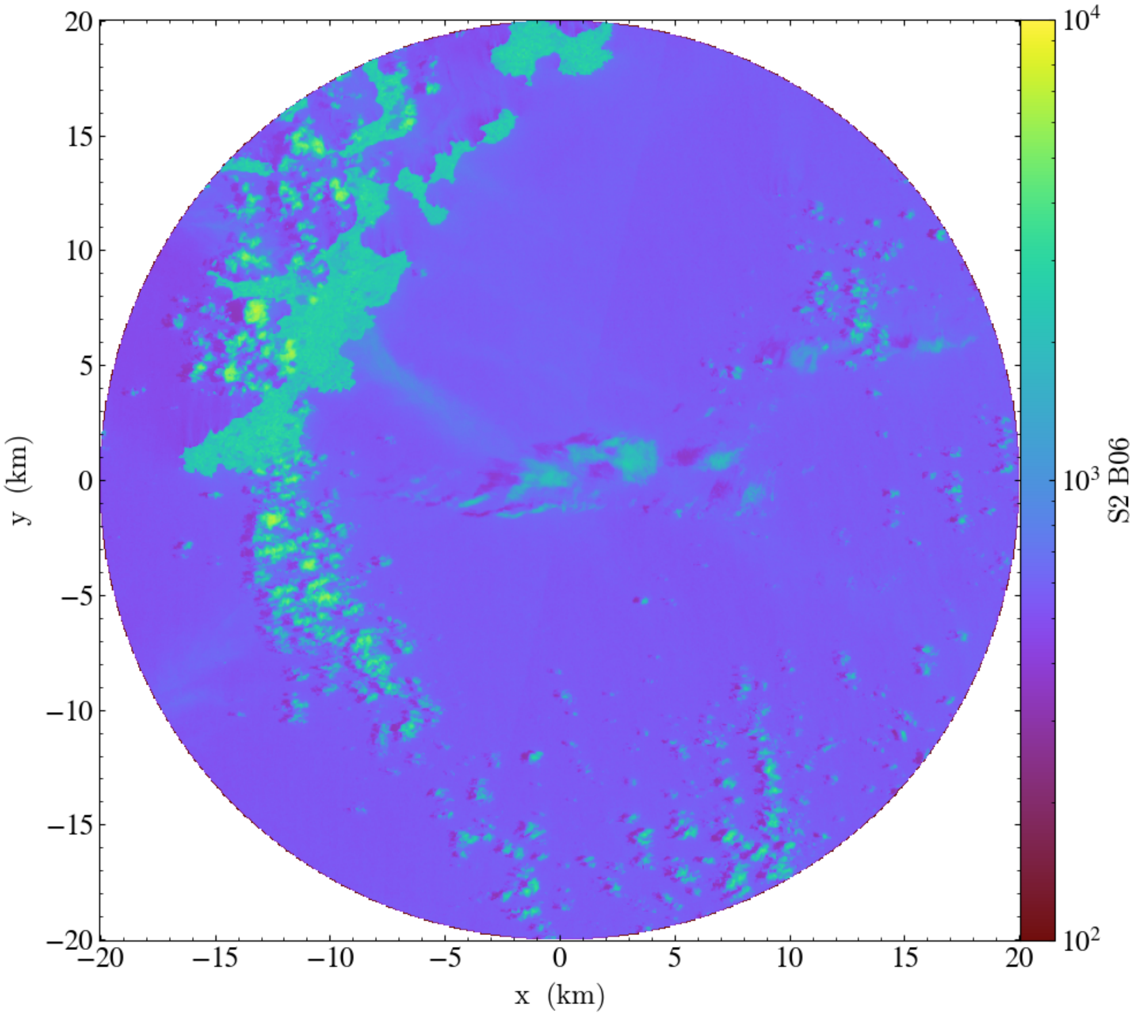
The `save_as_geotiff()` function will return paths to the saved GeoTIFF file as well as a supplementary yaml file containing a mapping from the GeoTIFF bands to the original field names. This can be provided to `yt.load` using the `field_map` keyword so the fields in the new file can be accessed with the original names.

```
>>> ds_new = yt.load(ds_fn, field_map=fm_fn)
yt : [INFO     ] 2021-06-30 14:44:51,851 Parameters: domain_dimensions         = [668␣
↪668   1]
yt : [INFO     ] 2021-06-30 14:44:51,851 Parameters: domain_left_edge          = [␣
↪434820. 9925060.       0.] m
yt : [INFO     ] 2021-06-30 14:44:51,852 Parameters: domain_right_edge         = [4.
↪74900e+05 9.96514e+06 1.00000e+00] m
```

(continues on next page)

(continued from previous page)

```
>>> print (ds_new.field_list)
[('LC08_L2SP_171060_20210227_20210304_02_T1', 'L8_B1'),
 ('LC08_L2SP_171060_20210227_20210304_02_T1', 'LS_temperature'),
 ('T36MVE_20210315T075701', 'NDWI'),
 ('T36MVE_20210315T075701', 'S2_B06')]

>>> field = ("T36MVE_20210315T075701", "S2_B06")
>>> p = ds_new.plot(field)
>>> p.set_zlim(field, 100, 10000)
>>> p.set_axes_unit("km")
>>> p.save("plot_4.png")
```



1.8 API Reference

1.8.1 Functions

1.8.2 Classes

Is This Page Empty or Broken?

If you are reading this on readthedocs.org, this page may be empty or the links to the various functions and classes may not be clickable. Getting the API docs to show up requires installing `yt_georaster`, which is difficult to accomplish on readthedocs. Sometimes it works. Anyway, you can build the docs locally by installing `yt_georaster` with the developer requirements and using sphinx to build the docs.

```
$ cd yt_georaster
$ pip install -e .[dev]
$ cd doc
$ make html
```

These docs can then be found in the `yt_georaster` source directory in `doc/build/html`.

CHAPTER

TWO

SEARCH

- search