yt_xarray: Facilitating Software Reuse Between Space and Earth Sciences

Contents

Authors

- Chris Havlin: University of Illinois Urbana-Champaign, School of Information Sciences,
- Matt Turk: University of Illinois Urbana-Champaign, School of Information Sciences,

Abstract

Copy/Paste

Introduction

Overview of the project.

Skip to main content

Overview of the primary software pieces:

`xarray`

xarray: words: geo, NASA, cloud-native formats (zarr)

`yt`

more words

`yt_xarray`

more words

Recent improvements to `yt`, `yt_xarray`

yt_xarray: yt api access:

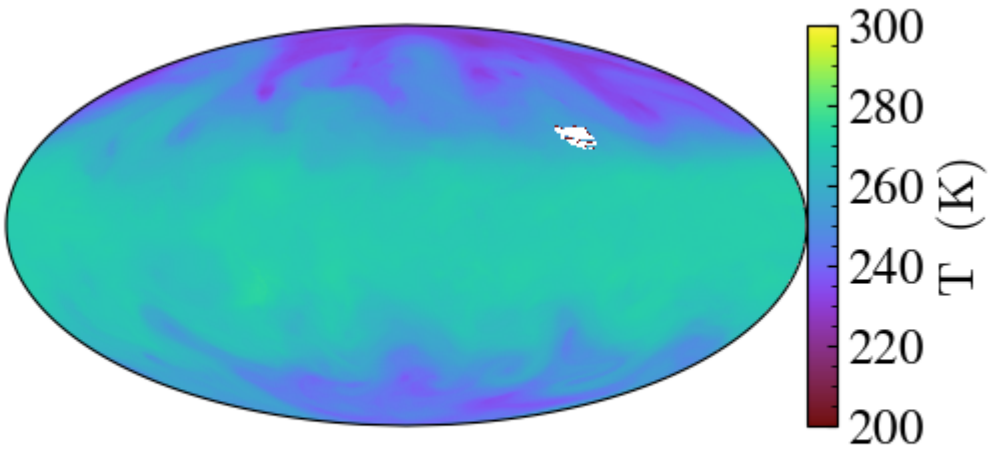
initial setup: silencing unrelated warnings and logs for brevity here.

```
import yt_xarray
import yt
import warnings
warnings.filterwarnings("ignore") # silence the cartopy bounds warning

yt.set_log_level(50)
yt_xarray.utilities.logging.ytxr_log.setLevel(50)
```

load the data. data is local file, data is from the MERRA-2 reanalysis dataset (hosted at GES DISC, NASA EarthData)

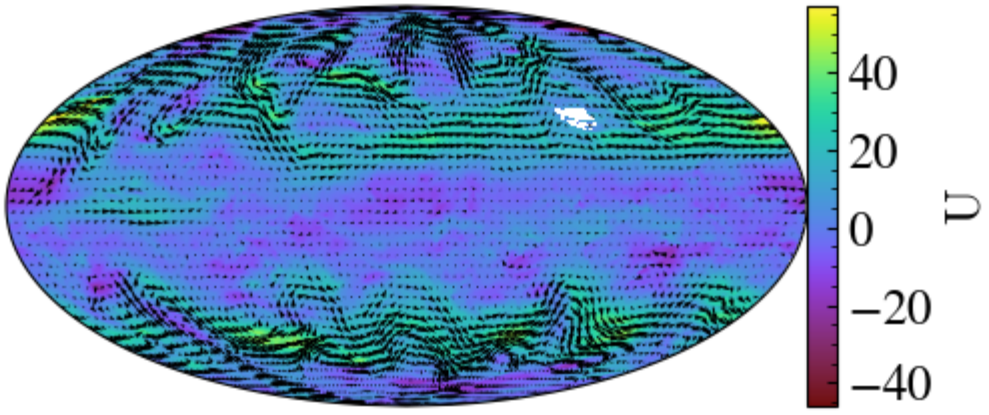
```
dsx = yt_xarray.open_dataset("sample_nc/MERRA2_100.inst3_3d_asm_Np.19800120.nc4")

dsx0 = dsx.isel({'time':0})
slc = dsx0.yt.SlicePlot('altitude', 'T', window_size=(4,2))
slc.set_log('T', False)
```

Skip to main content



yt: geoquiver:

```
ds = dsx.yt.load_grid(fields=['U', 'V'], sel_dict={'time':0}, use_callable=False)

slc = yt.SlicePlot(ds, 'altitude', 'U', window_size=(4,2))
slc.set_log('U', False)
slc.annotate_quiver('U', 'V')
slc.show()
```



yt: cartesian cutting plane:

Skip to main content

Embedded Transformations within yt_xarray

data commonly in geographic coordinates.

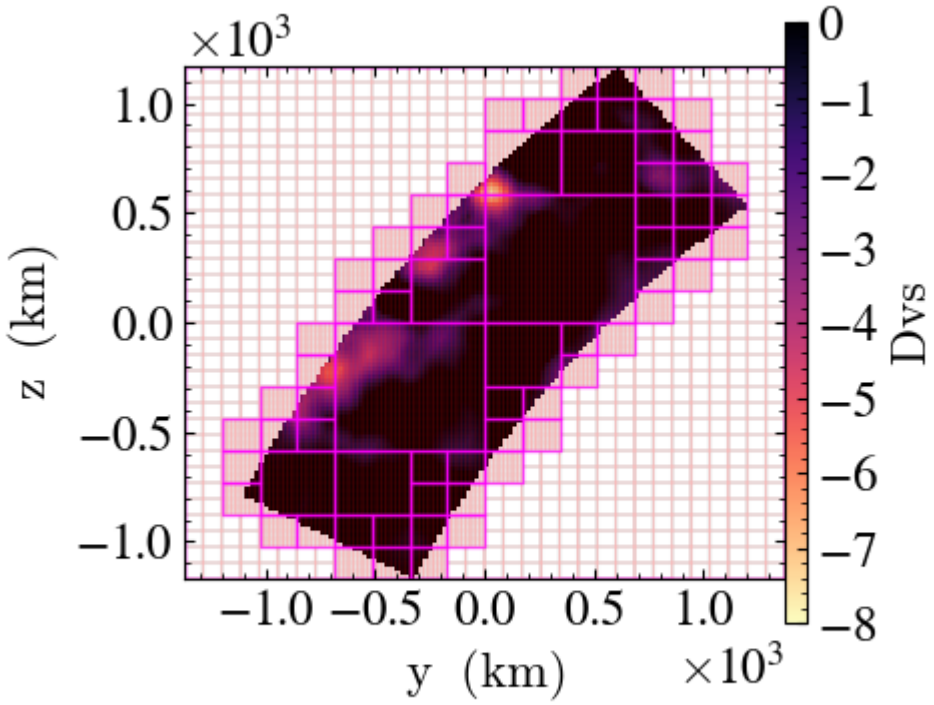
An example with 3D geographic data

```
import xarray as xr
import yt_xarray
import yt
from yt_xarray import transformations as tf
from yt_xarray.utilities.logging import ytxr_log
import numpy as np
import cartopy
from yt.visualization.volume_rendering.render_source import LineSource
from dask import delayed, compute
import shapely
```

```
yt.set_log_level(50)
ytxr_log.setLevel(50)

ds = yt_xarray.open_dataset("IRIS/wUS-SH-2010_percent.nc")
grid_resolution = (32, 32, 32)
gc = tf.GeocentricCartesian(radial_type='depth', r_o=6371., use_neg_lons=True)
ds_yt = tf.build_interpolated_cartesian_ds(
    ds,
    gc,
    fields = 'dvs' ,
    grid_resolution = grid_resolution,
    refine_grid=True,
    refine_max_iters=2000,
    refine_min_grid_size=4,
    refine_by=4,
    interp_method='interpolate',
)
```

```
slc = yt.SlicePlot(ds_yt, 'x', ('stream', 'dvs'), window_size=(3,3))
slc.set_log(("stream", "dvs"), False)
slc.set_cmap(("stream", "dvs"), "magma_r")
slc.set_zlim(("stream", "dvs"), -8, 0)
slc.annotate_cell_edges(color=(1,0,0), alpha=0.3)
slc.annotate_grids(edgecolors=(1,0,1,1))
slc.show()
```

Skip to main content



add a field to return abs(dvs<0)

```
def _slow_vels(field, data):
    dvs = data['dvs'].copy()
    dvs[np.isnan(dvs)] = 0.0
    dvs[dvs>0] = 0.0
    return np.abs(dvs)

ds_yt.add_field(
    name=("stream", "slow_dvs"),
    function=_slow_vels,
    sampling_type="local",
)
```

volume render: use the yt_xarray transformer to easily create a yt `LineSource` in the correct cartesian coordinates. Using dask here to process NaturalEarth state boundaries accessed via cartopy

Skip to main content

```
def process_state(state):
    linesegs = []
    if isinstance(state, shapely.geometry.polygon.Polygon):
        geoms_iter = [state,]
    elif isinstance(state, shapely.geometry.multipolygon.MultiPolygon):
        geoms_iter = state.geoms
    else:
        msg = f"Unexpected geometry type: {type(state)}"
        raise TypeError(msg)

    for geom in geoms_iter:
         linesegs = transform_geom_bounds(linesegs, geom.boundary.xy)
    return linesegs
```

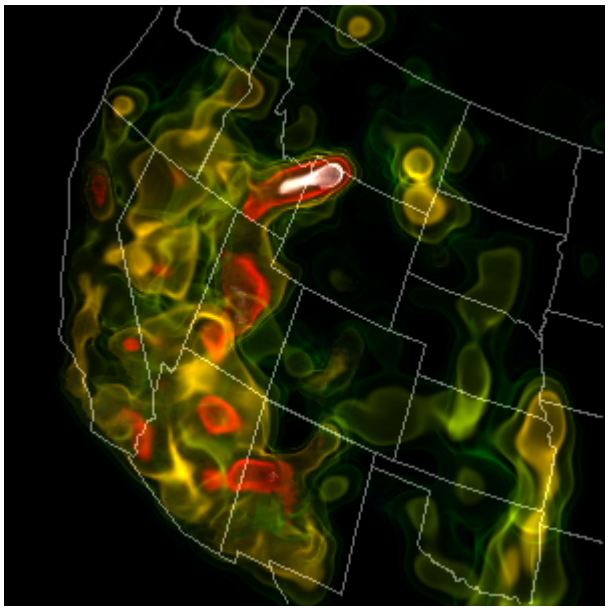
```
def transform_geom_bounds(linesegs, xy):
    lons = np.array(xy[0])
    lats = np.array(xy[1])

    x, y, z = gc.to_transformed(latitude=lats,
                                longitude=lons,
                                depth=0.0)

    for iseg in range(len(x)-1):
        lineseg = [[x[iseg], y[iseg], z[iseg]], [x[iseg+1], y[iseg+1], z[iseg+1]]]
        linesegs.append(lineseg)
    return linesegs
```

```
state_segs = []
for s in cartopy.feature.STATES.geometries():
    state_segs.append(delayed(process_state)(s))

state_segs = np.concatenate(compute(*state_segs))
colors = np.ones((state_segs.shape[0], 4))
colors[:,3] = 0.1
lsrc = LineSource(state_segs,colors=colors)
```

now ready for the volume rendering

Skip to main content

```
reg = ds_yt.region(ds_yt.domain_center,
                   ds_yt.domain_left_edge,
                   ds_yt.domain_right_edge)
sc = yt.create_scene(reg, field=('stream', 'slow_dvs'))
cam = sc.add_camera(ds_yt)

# transfer function
source = sc[0]
source.tfh.set_bounds((0.1, 8))
source.tfh.set_log(True)

# add state outlines
sc.add_source(lsrc)

# adjust camera
cam.zoom(2)
cam.yaw(100*np.pi/180)
cam.roll(220*np.pi/180)
cam.rotate(30*np.pi/180)
cam.set_resolution((300,300))

sc.show(sigma_clip=5.)
```



Utilizing Cloud Native data formats with yt_xarray

zarr via xr

Skip to main content

zarr with yt

zarr with yt paragraph or two

Summary

In summary

Technical Appendix

notebook requirements, notes on use of development branches, etc.

development branches:

yt: need dev (until yt4.4, geoquiver) yt_xrarray: need PR branch

building this book

https://jupyterbook.org/en/stable/advanced/pdf.html

```
$ pyenv virtualenv 3.10.11 yt_NASA_SMD
$ pyenv activate yt_NASA_SMD
```

from top level

```
$ pip install -r requirements.txt
$ jupyter-book build yt_xr_2024/ --builder pdfhtml
$ cp yt_xr_2024/_build/pdf/book.pdf ./yt_xr_2024.pdf
```

all the data