yt-xarray, Facilitating Software Reuse Between Space and Earth Sciences

Chris Havlin & Matt Turk

May 03, 2024

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

In this presentation, we describe recent efforts to better link two open source Python packages, yt and xarray, in order to improve the ability to use yt with NASA, cloud-hosted datasets. yt provides support for the analysis and visualization of multi-resolution volumetric data. It can read, understand, and process data from dozens of different platforms, including well-used and established astrophysical simulation data formats as well as observational and simulation data from a number of geophysical domains. Xarray is a popular metadata-preserving array library with excellent support for analysis of remotely stored data, particular through its support for cloud-native formats like Zarr. Recent work on both a new yt_xarray package and core yt has simplified the ability to analyze and visualize geospatial and geophysical datasets loaded with xarray using yt. In this presentation, we describe some of the improvements, including 3D volume rendering with embedded interpolation and use of yt analysis methods with remotely hosted data using xarray as a data backend.

0.1 Introduction

`yt` is a Open Source Python package for analysis and visualization of volumetric data. It was originally written for analysis of multi-resolution astrophysical simulation outputs and has expanded its use cases into additional domains such as geodynamics, geophysics, weather simulation and engineering. `yt` can ingest data from a wide range of data structures including AMR grid patches, Octree structures, smoothed-particle hydrodynamic output and unstructured meshes. Additionally, a large portion of the methods in `yt` are parallelized with MPI and commonly used in HPC systems for analysis of simulation data.

Recent efforts to improve the use of `yt` in geoscience domains have focused on improving documentation (Havlin et al., 2020, 2021) and improving interoperability with other Python packages which has resulted in the `yt_xarray` package. `xararay` is a popular metadata-preserving array library with support for a large number of file formats including traditional formats like netCDF and HDF but also newer cloud optimized storage solutions like Zarr arrays.

The work presented here describes recent improvements to `yt_xarray`, most notably, the introduction of a coordinate transformation framework to simplify the steps required to utilize any of the methods in `yt` that rely on ray tracing (such as Volume Rendering). We also describe current efforts to leverage Zarr within the `yt` framework, both indirectly through the `xarray` backend exposed by `yt_xarray` and more directly within `yt` to access chunked particle data and multi-resolution grid structures.

0.2 Introducing `yt_xarray`

`yt_xarray` is a package built to facilitate data transfer between `xarray` and `yt`. Its primary implementation is through an `xarray` accessor object, accessible off of any `xarray` dataset object. Given an `xarray` dataset, `ds`, the primary method `ds.yt.load_grid` will load a field or subset of fields from the `xarray` dataset within a wrapping `yt` dataset:

```
import yt_xarray

# load a standard xarray dataset
ds = yt_xarray.open_dataset("path/to/your/dataset.nc") # or xarray.open_dataset
```

(continues on next page)

(continued from previous page)

```
# initialize a yt dataset wrapper
yt_ds = ds.yt.load_grid(fields=(['list','of','fields']))
```

The `yt` dataset, `yt_ds`, is a full-fledged `yt` dataset that can be used with `yt` but containing references to the original `xarray` dataset. This means that data required by `yt` is read in from the original `xarray` dataset on demand, without copying data by default.

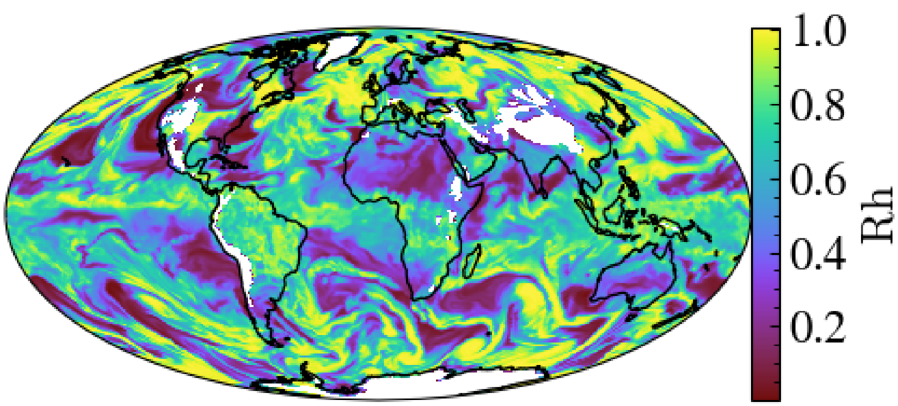
0.2.1 Recent Improvements

A number of recent updates to both `yt_xarray` and `yt` have simplified using `yt` with `xarray` datasets.

First, a number of functions from the `yt` API have been exposed from within the `yt_xarray .yt` accessor object, including: `SlicePlot`, `ProjectionPlot`, `PhasePlot` and `ProfilePlot`, allowing the user to skip the intermediate step of creating a `yt` dataset object.

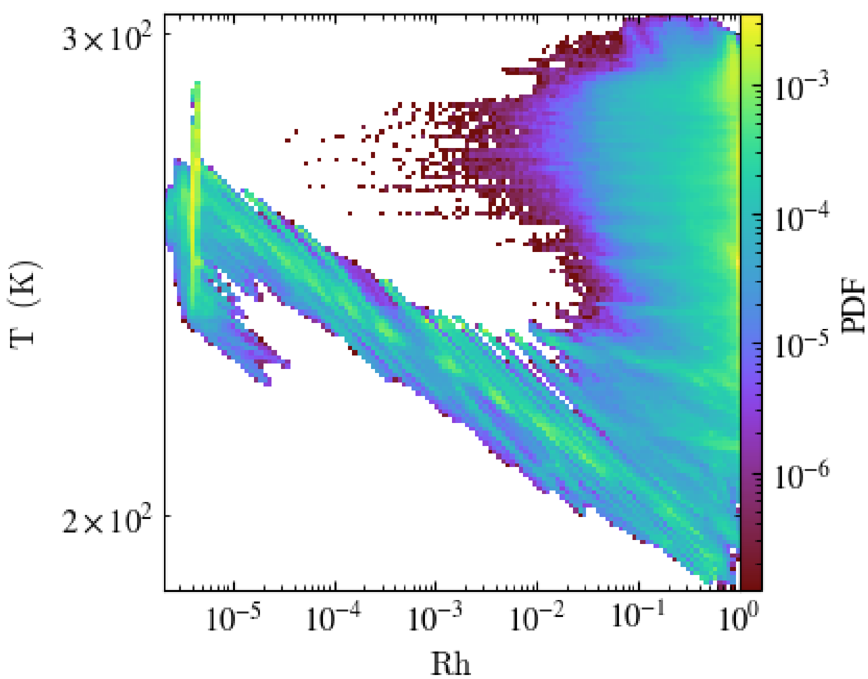
The following example creates a slice plot of a MERRA-2 reanalysis file (Global Modeling and Assimilation Office, 2015):

```
import yt_xarray
import cartopy.feature as cfeature
import numpy as np

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', 'RH', center=(800, 0.,0.))
slc.set_log('RH', False)
slc.render()
slc.plots['RH'].axes.add_feature(cfeature.COASTLINE)
slc.show()
```



The following code creates a `yt.PhasePlot`, a 2D binned statistic plot by binning the temperature (T) and relative humidity (RH) variables across the whole dataset. The 3rd variable is the binned field (in this case an array of ones) and values are summed within bins (by setting `weight_field=None`) and normalized by their totals (`fractional=True`), resulting in a global 2D probability distribution of T vs RH.

```
# add a field fo
ones_da = xr.DataArray(np.ones(dsx0.RH.shape), dims=dsx0.RH.dims)
dsx0['ones_field'] = ones_da

pp = dsx0.yt.PhasePlot('RH', 'T', 'ones_field', weight_field=None, fractional=True,␣
↪figure_size=(3,3))
pp.set_colorbar_label('ones_field','PDF')
pp.set_font_size(12)
pp.show()
pp.save('slice_images/merra2_phase_plot.png')
```



Within `yt`, there have also been improvements to plot annotations for geopgraphic data (released in `yt` v4.4.0) and in-progress arbitrary cutting planes which will allow cross-section construction (`yt` PR#4847)

0.3 Embedded Transformations within yt_xarray

In addition to providing methods for creating `yt` datasets that directly reference `xarray` datasets, ongoing development within `yt_xarray` (yt_xarray PR #75) will provide a number of methods for building cartesian `yt` datasets with embedded transformations and interpolation of `xarray` datasets defined in non-cartesian coordinates. This approach provides a convenient way of utilizing `yt` methods that rely on cartesian geometries without having to pre-interpolate data, making reproducible workflows to, for example, generate 3D volume renderings much simpler to make.

The general workflow is

1. open the xarray dataset
2. define the transformation from the dataset's native coordinatees to cartesian coordinates
3. define the method of interpolation

4. initialize the yt dataset

In pseudo-code, the above steps look like:

```
import yt_xarray
from yt_xarray.transformations import GeocentricCartesian, build_interpolated_
↪cartesian_ds

ds = yt_xarray.open_dataset(...) # or xr.open_dataset
gc = GeocentricCartesian(...)
ds_yt = build_interpolated_cartesian_ds(ds, gc, ...)  # a cartesian yt dataset
```

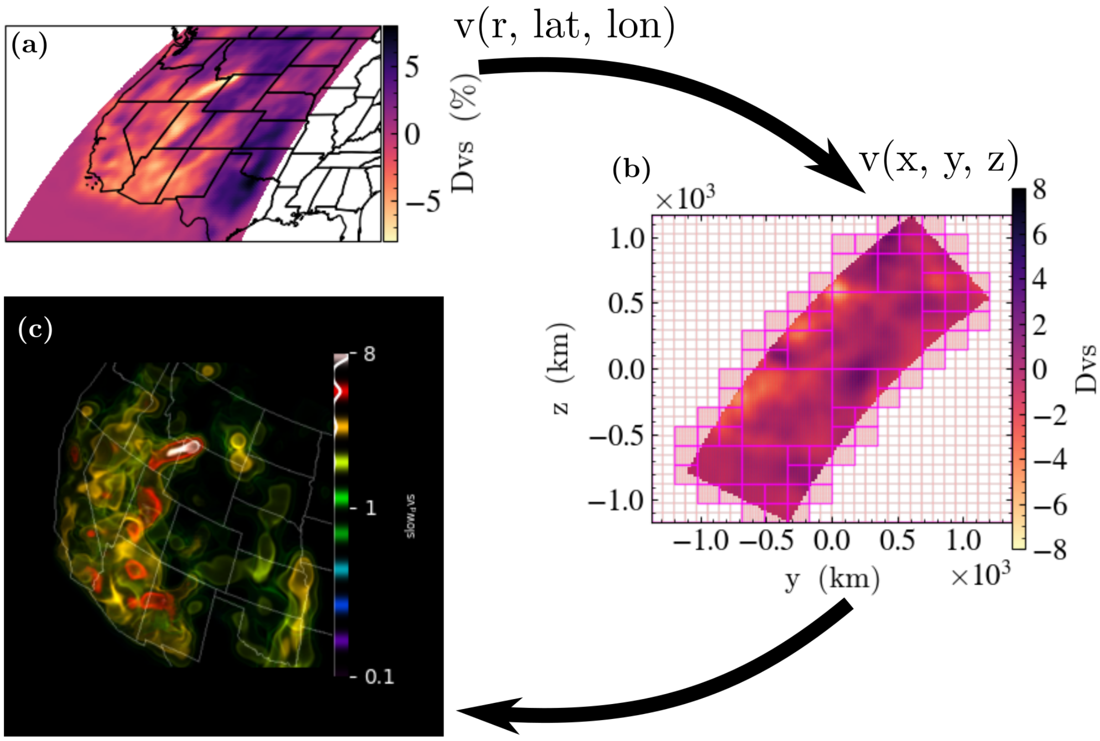
Initially, `build_interpolated_cartesian_ds` only sets up the cartesian grid (or grids) that will be used to wrap the non-cartesian geometry and actual interpolation of the data onto the cartesian grid (or grids) is delayed until `yt` needs the data. During initialization of the `yt` dataset (`build_interpolated_cartesian_ds`, step 4 above), the user can specify parameters that control how the cartesian grid is built. For example, the following pseudo-code:

```
build_interpolated_cartesian_ds(
    ds,
    gc,
    grid_resolution=(16,16,16),
    refine_grid=True,
    refine_by=8,
    refine_min_grid_size=2,
   ...)
```

`grid_resolution` specifies the coarse grid resolution of 16 cells in each dimension. When `refine_grid` is `True`, `yt_xarray` will apply a recursive subdivision of the domain into a number of smaller grids: starting from the coarse grid, the domain is divided in half along each dimension and grid cells are filled with a binary image mask, where cell values are set to 1 if they fall within the bounds of the non-cartesian geometry. Division proceeds recursively, with empty grids being discarded, until the remaining grids satisfy an adjustable fill criteria (or the max number of iterations has been reached). In addition to the recursive bisection, an implementation of the grid refinement algorithim of Berger and Rigoutsos (1991) is available to use by setting `refinement_method='signature_filter'`.

In the following, two examples with real datasets are described, with images from sequential stages of the above transformation workflow along with a final 3D volume rendering. The first example uses a seismic tomography model of the Earth's upper mantle beneath the Western U.S. (Schmandt and Humphreys, 2010), while the second uses a locally downloaded 3D MERRA-2 reanalysis data file (Global Modeling and Assimilation Office, 2015).

0.3.1 Volume Rendering Workflow: Seismic Tomography



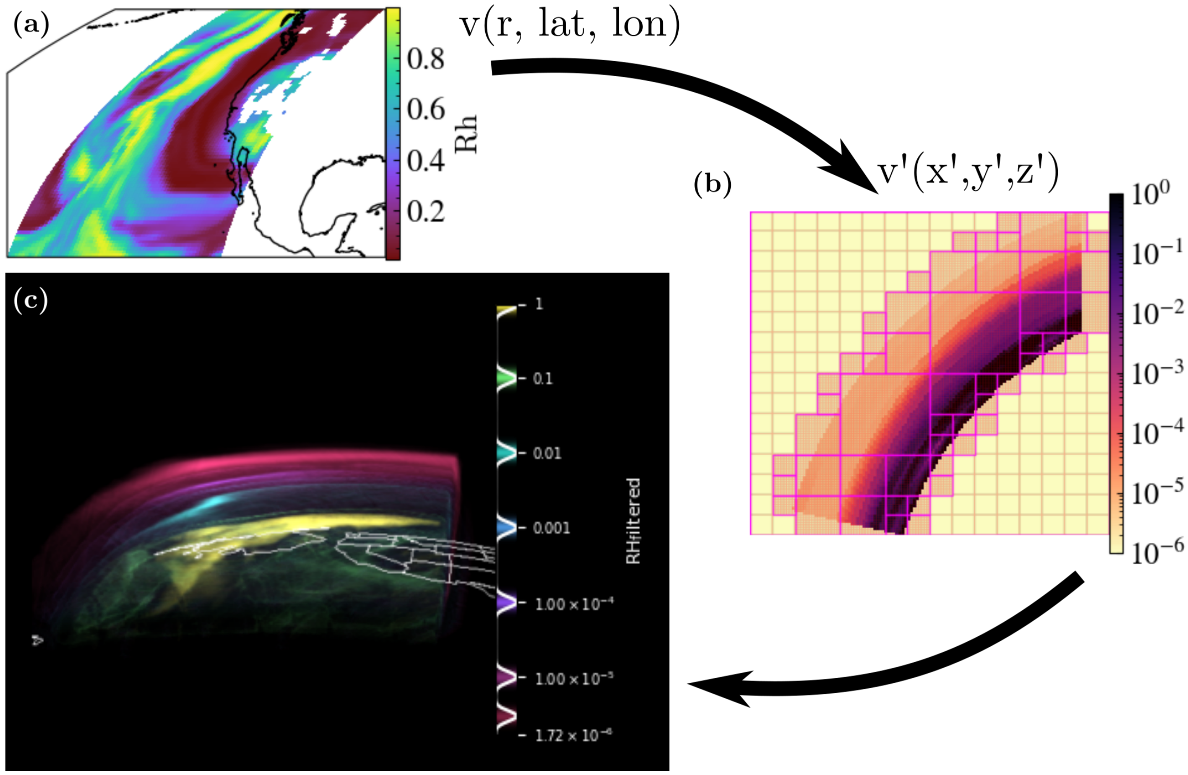
(a) The data is initially loaded with `xarray` in native coordinates which in this case is internal geodetic coordinates (depth beneath the Earth's surface, latitude and longitude). From here, all the usual `xarray` methods apply: the image here is a slice at 100 km depth through shear wave speed anomalies (Dvs), generated in this case with `yt_xarray`'s convenience wrapper of `yt.SlicePlot` (see previous section).

(b) After defining the transformation from geodetic coordinates to cartesian coordinates, a wrapping cartesian grid is refined to generate a cartesian `yt` dataset. This cartesian dataset can be used with any `yt` method. A `yt.SlicePlot` with grid and cell annotations illustrates the refined grid structure: the squares outline in bold represent the edges of grids while fainter lines indicate grid cells. The starting grid here was 32x32x32 with a refinement factor of 8, resulting in a dense sampling where there is data in the underyling geometry. In constructing this plot, `yt` will access data only in the grids intersected by the desired cutting plane and grids are processed individually. This means that only a subset of the full dataset is interpolated on-demand, as needed by yt.

(c) Once the cartesian `yt` dataset is available, methods in `yt` that rely on cartesian ray tracing are available to use. The image here is a 3D volume rendering of only the slow velocity anomalies (where Dvs is less than 0). The transfer function used here consists of a number of gaussian samples of the data spaced linearlly between 0.1 and 8 percent and results in a clear signal of the Yellowstone hot spot track beneath northeast Idaho and northwest Wymoing, where high temperatures and partially molten rock decrease the shear wave speed

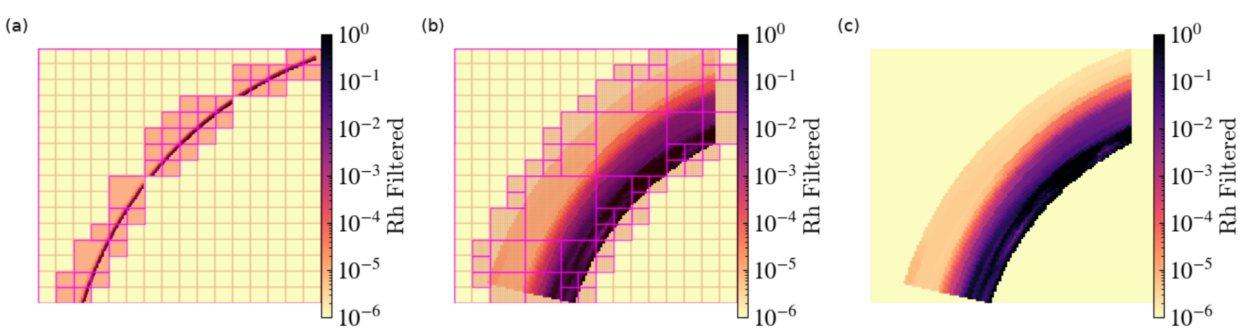
0.3.2 Volume Rendering Workflow: Atmospheric Geophysics, MERRA-2

The second example demonstrates an example with a 3D atmospheric field using data from MERRA-2. Due to the small radial length scale of the atmosphere compared to the features being rendered, an additional radial scale factor is applied in transforming to virtual cartesian coordinates.



(a) Again, data is initially loaded with `xarray` in native coordinates and a plot of relative humidty (RH) at 800 hPa is made using the `SlicePlot` convenience method.

(b) Defining the transformation in this case is more complicated than the previous case. The data here is defined with dimensions of (pressure level, latitude, longitude) and so an additional transformation from pressure level to altitude is required. Additionally, the radial scale of the atmosphere is small compared to the longitudinal and latitudinal distances of interest here. If we were to transform to geocentric cartesian coordinates, the atmosphere would be very difficult to resolve, as illustrated in panel (a) of the following image:

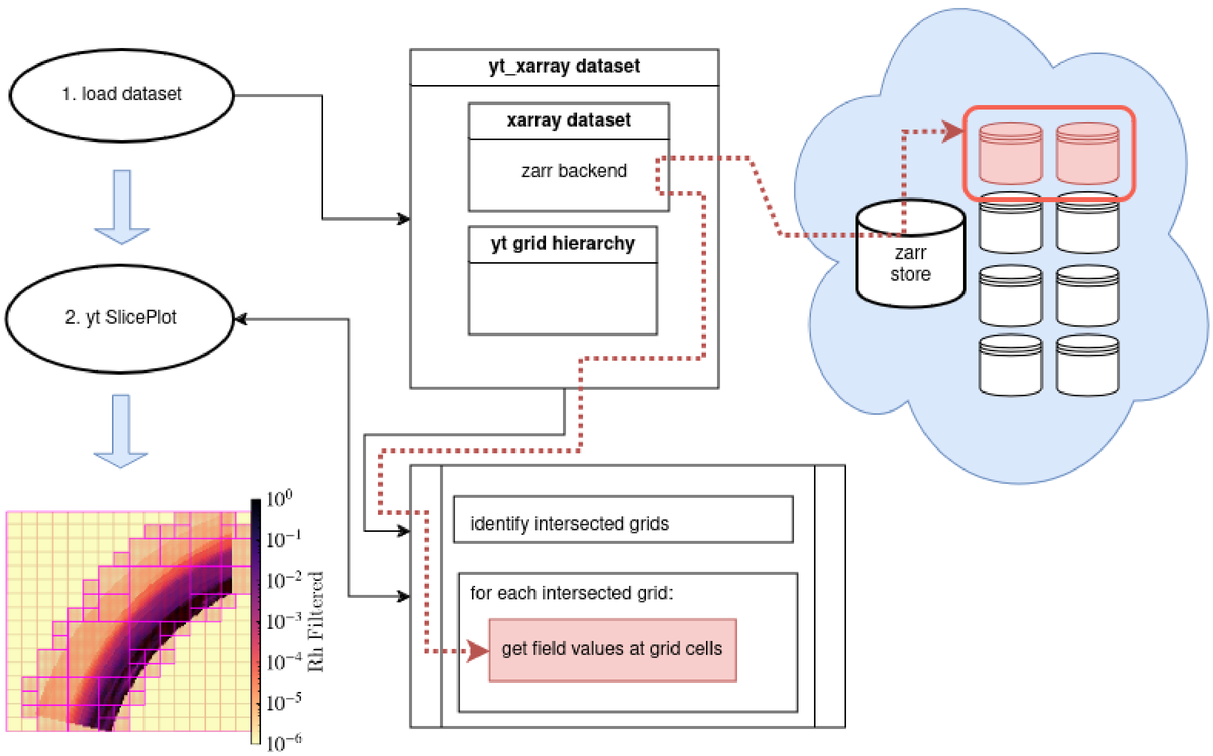


For the purposes of volume rendering, we can overcome this issue by applying a radial scale factor: panel (b) shows a scale factor of 20 and (c) shows the same without the grid annotations. So to create our cartesian `yt` dataset that contains the `xarray` dataset in native geometry, we first must implement a pressure-to-height transformation and then utilize a scaled geodetic to cartesian transformation. Both of these transformations could be contained within a single transformation object, but because this particular MERRA-2 file contains a height variable, the actual implementation can be simplified slightly. In addition to the transformation considerations, the RH field ocntains missing values identify by NaN values. Because projections in `yt` do not yet handle `NaN` values well, the interpolation here fills in missing values with a small non-negative number.

(c) Returning to the volume rendering workflow image, once a cartesian yt dataset is initialized, we can again utilize the volume rendering methods. In this case, a logarithmic transfer function highlights a number of relative humidity values (within the range of actual data and avoiding the small nonzero filler used for missing values).

In summary, the embedded transformation framework within `yt_xarray` also convenient access to methods in `yt` that require a dataset in cartesian coordinates. A cartesian grid (or grids if using refinement) is built to wrap the underlying non-cartesian geometry and data is read and interpolated onto the cartesian grid on-demand by chunks using the linked xarray dataset.

0.4 Utilizing Cloud Native data formats with yt_xarray



One of the benefits of linking `yt` to `xarray` dynamically is the access to the well-developed methods within `xarray` to work with a range of file types, in particular cloud-native formats like `Zarr` (CITATION). Additionally, because `yt_xarray` is careful to delay data reads until required while maintaining links to the underlying `xarray` dataset, the chunked reading possible with `Zarr` (or dask arrays).

The flow chart above illustrates the steps and objects involved in the `yt-xarray-Zarr` workflow. The steps visible to the user are at left: (1) load a dataset, (2) construct a slice plot and then return an image. Behind the scenes, initially loading a `yt_xarray` dataset links the underlying `xarray` dataset within a standard `yt` dataset and initializes the `yt` grid hierarchy. When a `yt` selection method is applied, such as creating a slice plot (or extracting data from a geometric subselection), `yt` first identifies grids within the hierarchy that intersect the selection object. For each grid that is intersected (and only for those intersected), `yt` will fetch data at those grid cells. At this point, `yt` will request data from the underlying `xarray` dataset.

At present, we are focusing on a number of complimentary avenues of development and research to improve analysis of cloud-native data with `yt` and `yt_xarray`. First, we are composite a set of tutorial notebooks demonstrating analysis workflows with `yt_xarray` that utilize subsets of cloud-hosted NASA Earth Observation Data in order to increase awareness and uptake of the current functionality. Additionally, we are investigating approaches to reading Zarr files from `yt` for both smoothed-particle hydrodynamics (SPH) simulation output and AMR grid structures.

`yt` can read and process output from a number of smoothed-particle hydrodynamics (SPH) simulations. These SPH simulations commonly store output in HDF files, and `yt` is enabled to read from and process particle data in chunks. While it may be possible to re-format many of these datasets in more cloud-ready formats like Zarr, it is also possible to obtain performant reads of existing cloud-hosted HDF files by using adding a simple fsspec metadata file that describes the HDF file and subseuqnetly loading a fsspec mapping object with Zarr (Signell, 2020). This approach should work well within yt's SPH data readers and allow an immediate avenue to utilizing cloud-hosted data for yt operations that subselect data without needing to change existing output formats.

In addition to particle-base data `yt` can ingest multi-resolution gridded data stored in grid patches of variable refinement as well as octree structures and we are investigating ways of representing such AMR structures within the Zarr framework. The OME-Zarr format was designed to store multi-resolution pyramidal image data (Moore et all, 2023) and has overlap with the goals here and initial experiments embedding a gridded `yt` dataset within the napari experimental Generative Zarr reader showed signficant promise (Havlin, 2023). But pyramidal image structures differ from AMR structures in that AMR structures do not necessarily contain data for every level of refinement at every position and so representing the potential sparseness within Zarr requires additional considerations. We are currently exporing the use of both ragged array and awkward array structures to store AMR hierarchies, both of which have recent or in-progress Zarr representations.

0.5 Summary

In summary

0.6 References

Ahlers, J., Althviz Moré, D., Amsalem, O., Anderson, A., Bokota, G., Boone, P., Bragantini, J., Buckley, G., Burt, A., Bussonnier, M., Can Solak, A., Caporal, C., Doncila Pop, D., Evans, K., Freeman, J., Gaifas, L., Gohlke, C., Gunalan, K., Har-Gil, H., … Yamauchi, K. (2023). napari: a multi-dimensional image viewer for Python (v0.4.18). Zenodo. https://doi.org/10.5281/zenodo.8115575

Global Modeling and Assimilation Office (GMAO) (2015), MERRA-2 inst3_3d_asm_Np: 3d,3-Hourly,Instantaneous,Pressure-Level,Assimilation,Assimilated Meteorological Fields V5.12.4, Greenbelt, MD, USA, Goddard Earth Sciences Data and Information Services Center (GES DISC), Accessed: 28 April 2024, 10.5067/QBZ6MG944HW0

M. Berger and I. Rigoutsos, "An algorithm for point clustering and grid generation," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 5, pp. 1278-1286, Sept.-Oct. (1991), doi: 10.1109/21.120081.

Havlin, C., Turk, M., Holtzman, B.K., Orf, L., Halbert, K., Naliboff, J.B., Kowalik, K., Munk, M. and Walkow, S., (2020), December. Visualization and Analysis of 3D Data in the Geosciences Using the yt Platform. In AGU Fall Meeting Abstracts (Vol. 2020, pp. IN037-13) with supplemental materials at https://github.com/chrishavlin/AGU2020

Havlin, C., Holtzman, B., Kowalik, K., Munk, M., Walkow, S. and Turk, M., (2021). 3D volume rendering of geophysical data using the yt platform. Earth and Space Science Open Archive ESSOAr.

Havlin, C. (2023), *3d progressive loading of yt dataset in napari*, Available at https://www.youtube.com/watch?v=ofoURuz-Cbw (Accessed 3 May 2024).

Moore, J., Basurto-Lozada, D., Besson, S., Bogovic, J., Bragantini, J., Brown, E.M., Burel, J.M., Casas Moreno, X., de Medeiros, G., Diel, E.E. and Gault, D., (2023). *OME-Zarr: a cloud-optimized bioimaging file format with international community support.* Histochemistry and Cell Biology, 160(3), pp.223-251. https://doi.org/10.1007/s00418-023-02209-1

Signell, R. (2020). *Cloud-Performant NetCDF4/HDF5 with Zarr, Fsspec, and Intake.* Available at https://medium.com/pangeo/cloud-performant-netcdf4-hdf5-with-zarr-fsspec-and-intake-3d3a3e7cb935 (Accessed 3 May 2024).

Schmandt, B. and E. Humphreys. (2010). "Complex subduction and small-scale convection revealed by body-wave tomography of the western United States mantle." Earth and Planetary Science Letters, 297, 435-445, https://doi.org/10.1016/j.epsl.2010.06.047. Avaialable at https://doi.org/10.17611/DP/9991760 (Accessed 1 May 2024)

0.7 Technical Appendix

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

0.7.1

development branches:

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

0.7.2 building this book

Recommended that you use `pdflatex` directive to build the pdf, which requires that you first install a texlive distribution, see

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 pdflatex
$ cp yt_xr_2024/_build/pdf/book.pdf ./yt_xr_2024.pdf
```

0.7.3 all the data