Using yt for Visualization at OLCF

About yt

- yt is a python library for analysis and visualization of volumetric datasets (e.g., from hydrodynamics codes)
  - active, welcoming community of developers
  - http://yt-project.org/ (http://yt-project.org/)
- supports adaptive mesh refinement, unstructured grids, and more
- *frontends* provided for many common codes (e.g., *BoxLib*, *Chombo*, *Enzo*, *Flash*, *Gadget*, ...)
  - any 3-d NumPy array can also be used
  - generic HDF5 support as well

About yt

- unlike VisIt or other codes, there is no GUI
- you write your visualization as a python script
  - you can also use ipython / jupyter notebooks
- **excellent cookbook** with all sorts of examples to base your vis off of: http://yt-project.org/docs/dev/cookbook/index.html (http://yt-project.org/docs/dev/cookbook/index.html)

In [18]:
```
import yt
yt.__version__
```

Out[18]:
```
'3.4-dev'
```

In [20]:
```
ds = yt.load("smallplt16623")
ds.print_stats()
```

```
level   # grids        # cells       # cells^3
------------------------------------------------
  0        512        16777216             256
  1        615         6795264             190
------------------------------------------------
          1127        23572480


t = 1.63001201e+02 = 1.63001201e+02 s = 5.16519638e-06 yea
rs

Smallest Cell:
        Width: 6.482e-18 Mpc
        Width: 6.482e-12 pc
        Width: 1.337e-06 AU
        Width: 2.000e+07 cm
```

Fields

- fields in yt have units
  - these are usually defined in the frontend for the different codes
  - units are propagated through calculations using SymPy
- many common derived fields are also computed, depending on what exists on disk

In [21]:
```
ds.field_list
```

Out[21]:
```
[('boxlib', 'Temp'), ('boxlib', 'density')]
```

In [23]:
```
dd = ds.all_data()
dd.quantities.extrema(("boxlib", "density"))
```

Out[23]:
```
YTArray([  3.20258364e-03,   1.85814420e+07]) g/cm**3
```

Working with Datasets

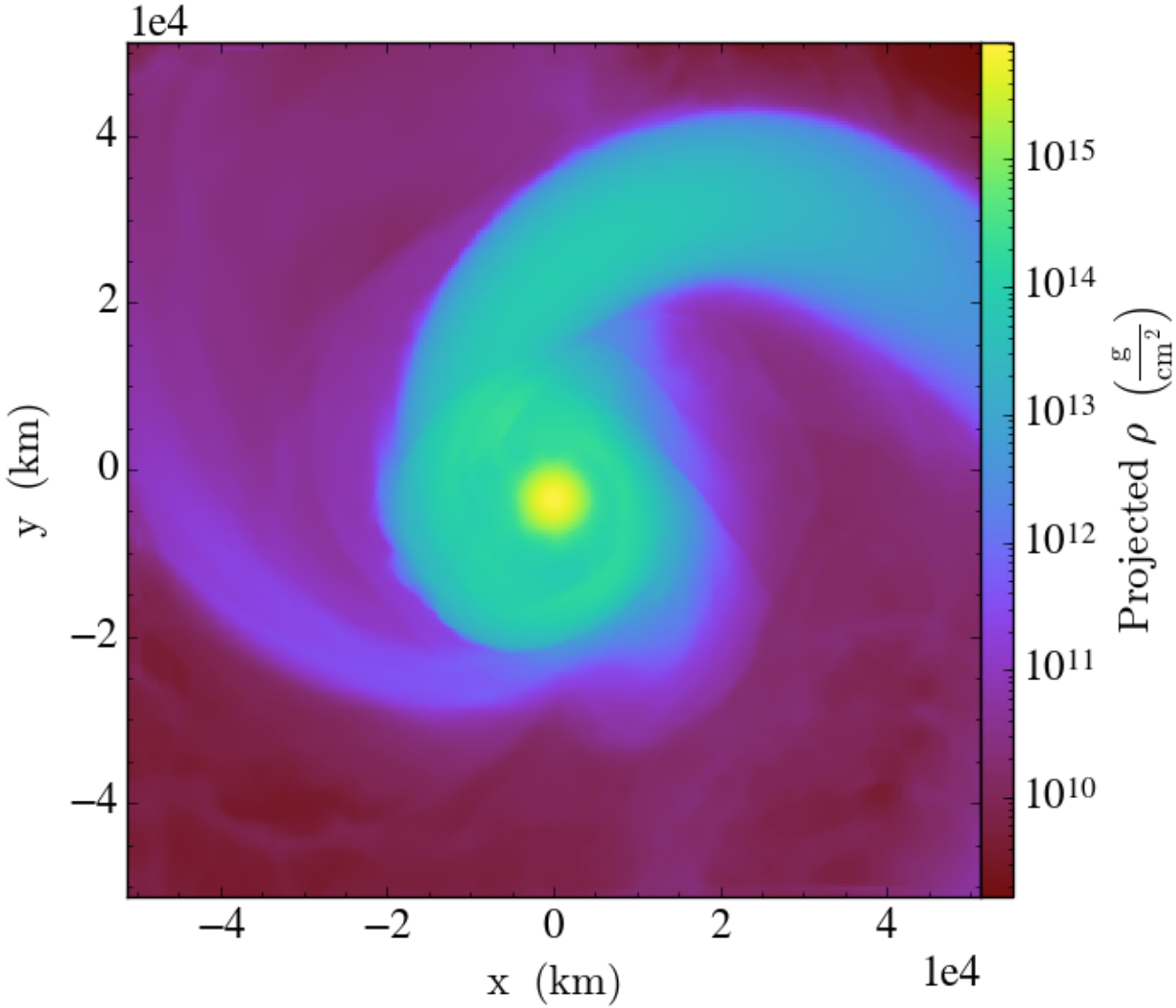
- yt can explore the mesh structure for AMR datasets
- you can easily create new fields via algebraic operations on existing fields
- you can select sub-regions (e.g. a sphere) to explore
- direct data exploration (e.g., inspect the data values directly)
- interface with other libraries (e.g., take an FFT to generate kinetic energy power spectrum)

Plotting

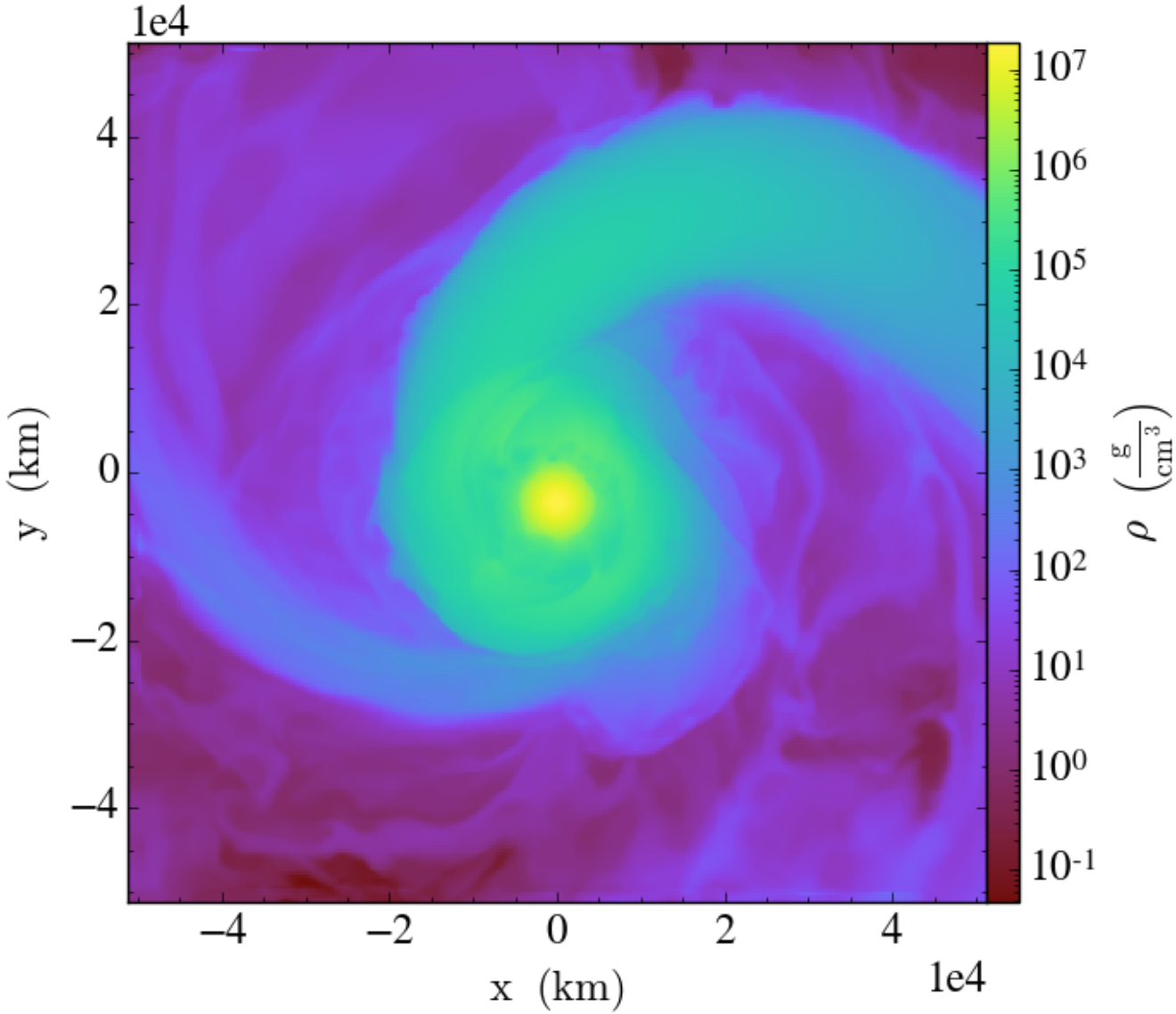
Common Plot Types

- *projection plot* : integrate through volume along a line (coordinate direction or off-axis). Can optionally be weighted
- *slice plot* : take a slice through the volume
- *volume rendering* : assign an opacity and color to different data values and render an image by shooting lightrays through the volume
- *phase plot* : plot two fields against one-another, optionally coloring by a different quantity

In [26]:
```
p = yt.ProjectionPlot(ds, "z", "density")
p.set_figure_size(5)
d = p.show()
```



In [12]:
```
s = yt.SlicePlot(ds, "z", "density")
s.set_figure_size(5.0)
s.show()
```



Volume rendering

Volume rendering requires a little bit of setup. The key pieces are:

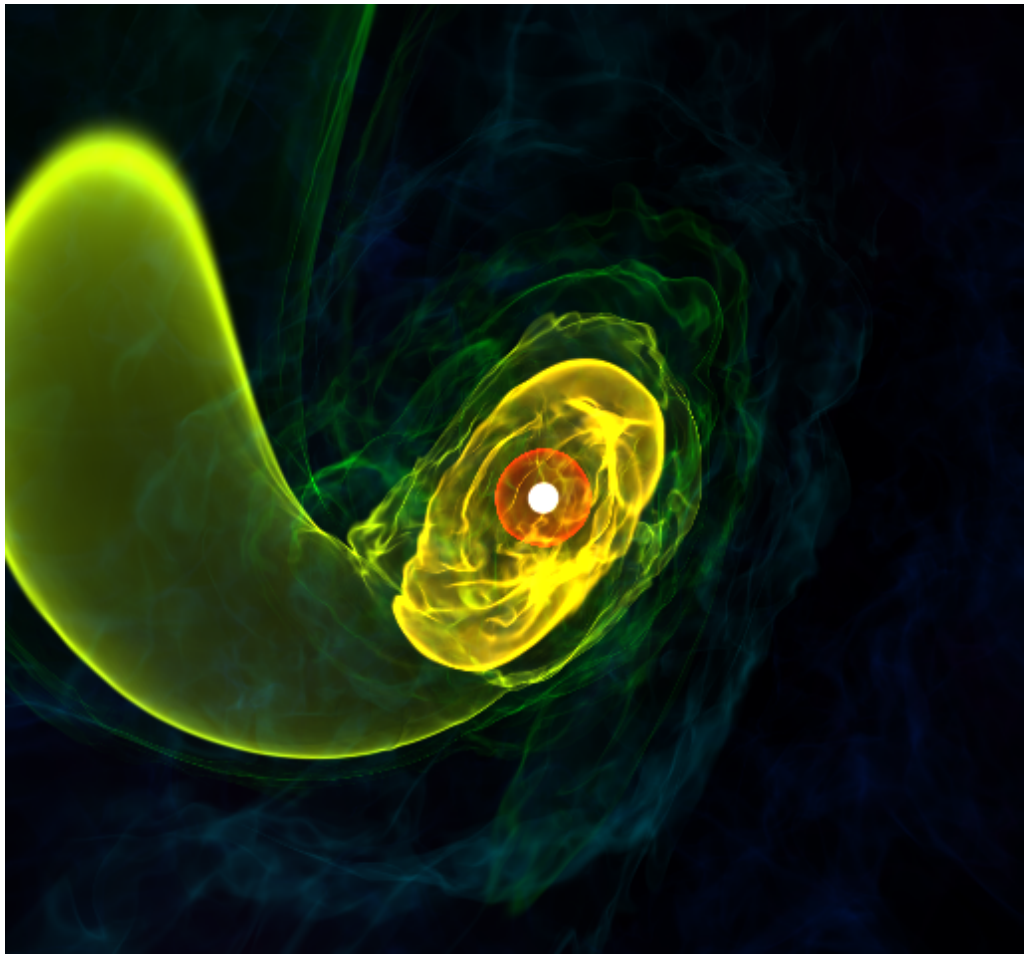
- *transfer function* : this maps the data values into colors and opacities
- *scene* : this is the "box" containing the volume that will be rendered. It contains the camera and volume sources
- *camera* : this is the viewing point -- lightrays cast through the volume are imaged by the camera, optionally altered by a lens
- *sources* : this includes the volume source (e.g. our data set), as well as annotation sources (like lines, etc.)

In [14]:
```
from yt.visualization.volume_rendering.api import Scene, V
olumeSource

sc = Scene()

vol = VolumeSource(ds.all_data(), "density")
sc.add_source(vol)
cam = sc.add_camera(ds, lens_type="perspective")
im = sc.render()
sc.show(sigma_clip=5.0)
```

Out[14]:



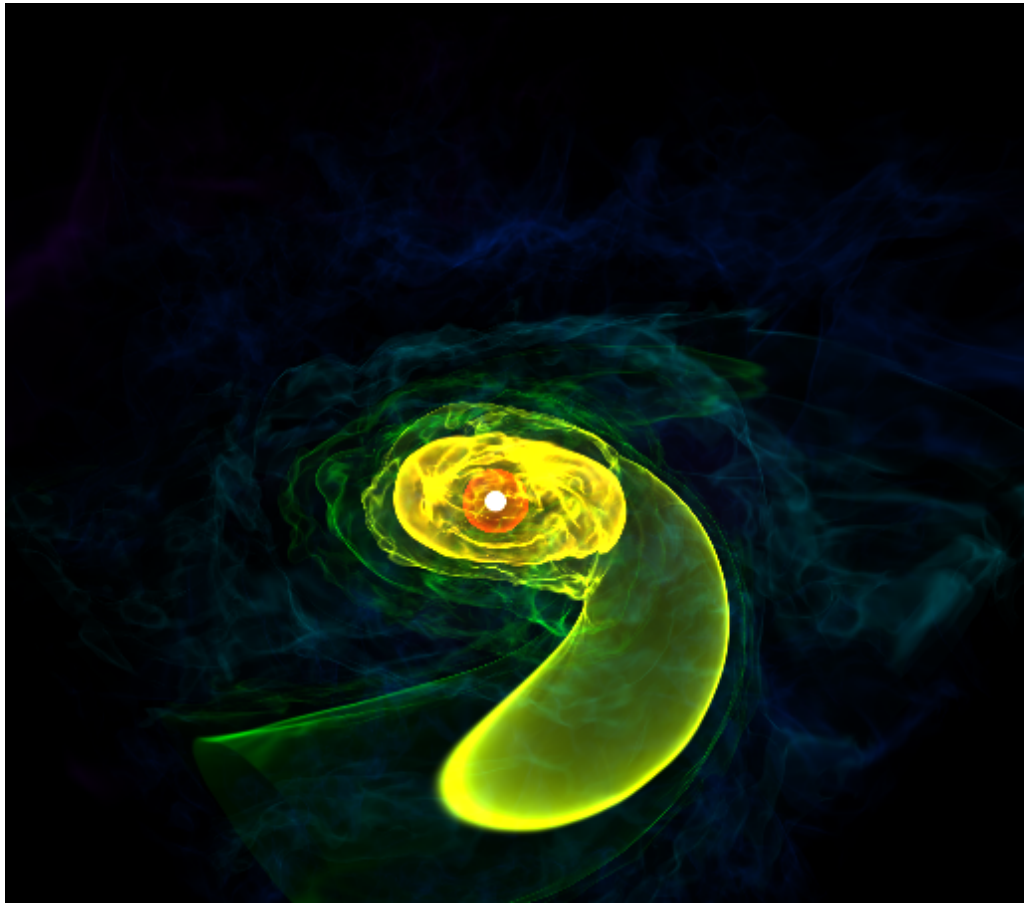
In [15]:
```
cam.position = 1.5*ds.domain_right_edge
center = 0.5*(ds.domain_left_edge + ds.domain_right_edge)

normal = (center - cam.position)
normal /= np.sqrt(normal.dot(normal))

cam.switch_orientation(normal_vector=normal, north_vector=
[0., 0., 1.])

im = sc.render()
sc.show(sigma_clip=5.0)
```

Out[15]:



Using yt at OLCF

Installing the Dev Version on Titan

- **You must be on a lustre filesystem so the compute nodes will see yt**
- Good place to install -- you project work directory
  - create a subdirectory, `yt/` in your project space for the install

In your project `yt/` directory, get the install script:

```
wget https://bitbucket.org/yt_analysis/yt/raw/yt/doc/inst
all_script.sh
```

edit the script and set:

```
INST_CONDA=1
INST_YT_SOURCE=1
```

This will use Conda to install the dependencies and build yt from source

```
source install_script.sh
```

This will install all the dependencies in a sub-directory `yt-conda`. When it is done, it will tell you to modify your `PATH` variable in your `.bashrc`, and then you are set.

Note: make sure that you are not loading any other python environments via modules, or else things will be confused.

Testing the Installation

Launch `ipython` and at the prompt do

```
In [1]: import yt
```

if no errors are produced, then yt was found and your installation was successful

Batch Visualization on Titan

Create a simple script, `vol.py`:

```
#!/usr/bin/env python

import yt

ds = yt.load("smallplt16623")

p = yt.ProjectionPlot(ds, "z", "density")
p.save()
```

and make it executable:

```
chmod o+x vol.py
```

Batch Visualization on Titan

We'll run on a single node.

Sample submission script:

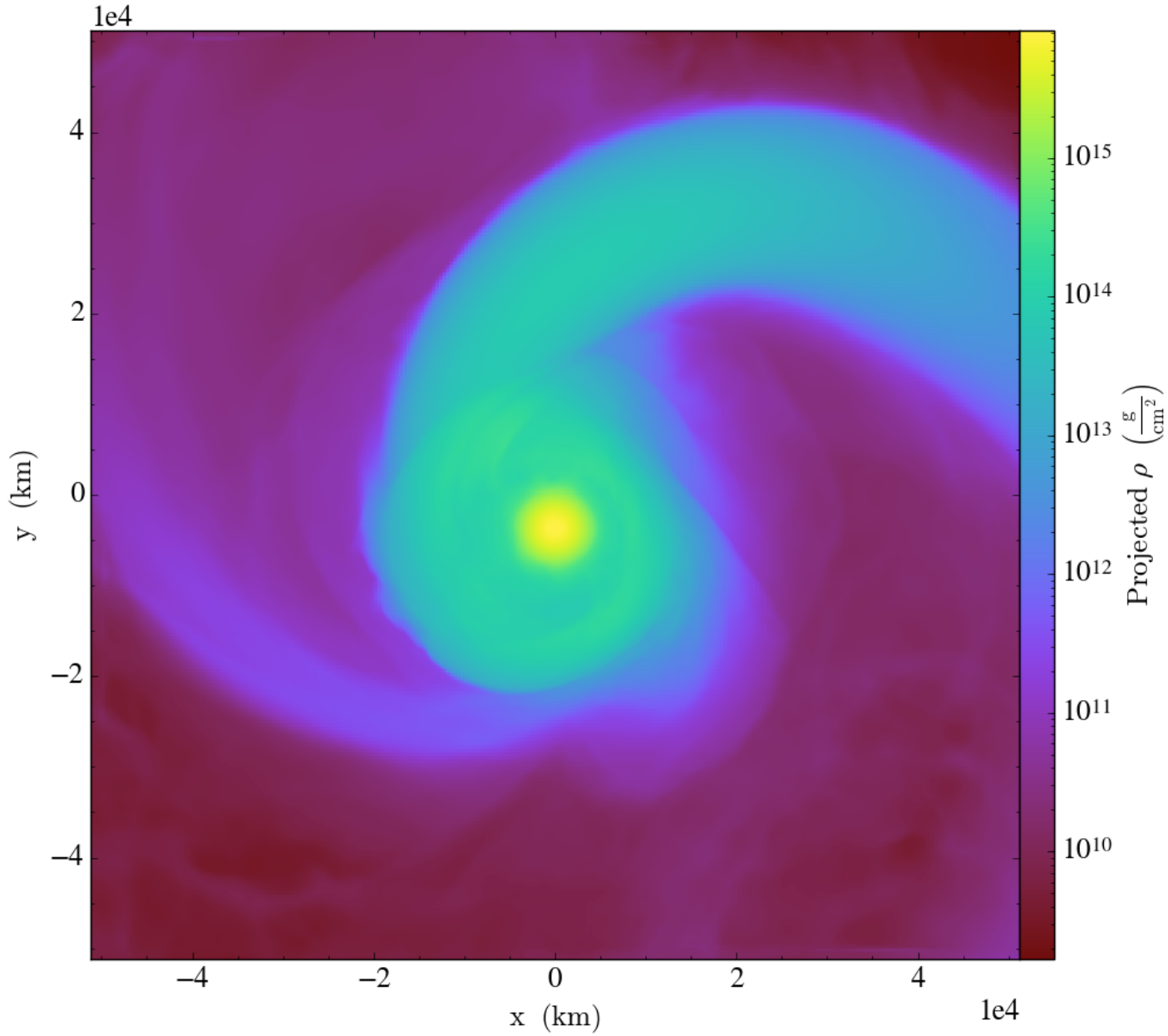
```
#!/bin/sh
#PBS -A XXXXXX
#PBS -N visualization
#PBS -j oe
#PBS -q debug
#PBS -l walltime=00:05:00,nodes=1

# this script runs with yt to volume render

export OMP_NUM_THREADS=16
cd $PBS_O_WORKDIR

aprun ./vol.py
```

Note that you still launch via `aprun` so it is executed on the compute node. Submit this as usual.


1e4
4
2
0
−2
−4
y (km)
−4
−2
0
2
4
1e4
x (km)
10^{15}
10^{14}
10^{13}
10^{12}
10^{11}
10^{10}
Projected ρ $\left(\frac{\mathrm{g}}{\mathrm{cm}^2}\right)$

Other Notes

- yt can use OpenMP (critical parts are written in Cython and are threaded)
- you can use something like `wraprun` to do trivial-parallel rendering of multiple output files on many nodes
- you can do the same install process on rhea
  - there we've also had luck compiling mpi4py and running yt using MPI across nodes

https://www.youtube.com/watch?v=0AAPwsST9WQ
(https://www.youtube.com/watch?v=0AAPwsST9WQ)