The University of Chicago

Visualizing FLASH with yt

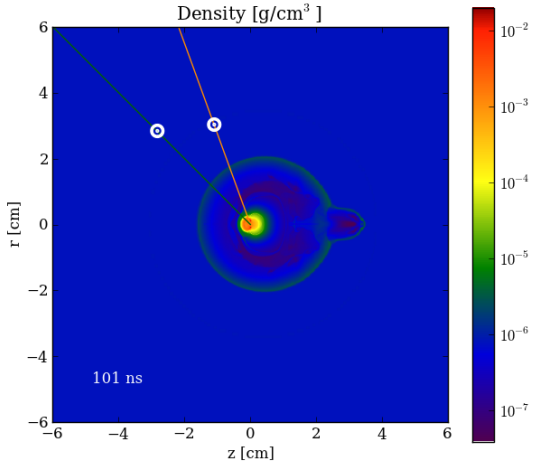
June 1st, 2012, RAL

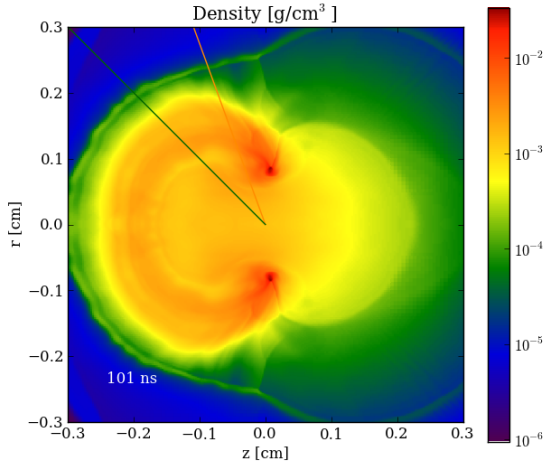
Anthony Scopatz - The FLASH Center
scopatz@flash.uchicago.edu

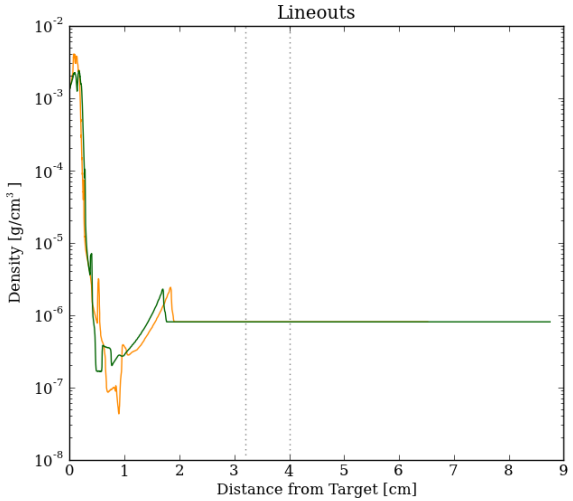
Goal

Goal

Make *easy*, *reproducible*, *publication-quality* figures.







Context

The existing solutions were inappropriate for the following reasons:

Context

The existing solutions were inappropriate for the following reasons:

- **VisIt:** Difficult to make publication worthy and even more difficult to reproduce.

Context

The existing solutions were inappropriate for the following reasons:

- **VisIt:** Difficult to make publication worthy and even more difficult to reproduce.
- **yt:** Lack of full control over basic objects (labels, legends, color maps, etc).

Context

The existing solutions were inappropriate for the following reasons:

- **VisIt:** Difficult to make publication worthy and even more difficult to reproduce.
- **yt:** Lack of full control over basic objects (labels, legends, color maps, etc).

We are going to need a bigger boat...

Key Points

- Though it is marketed as a visualization tool, yt is a fully-fledged analysis platform for FLASH.

Key Points

- Though it is marketed as a visualization tool, yt is a fully-fledged analysis platform for FLASH.
- Since yt is well-factored, the visualization & analysis feature sets are distinct.

Key Points

- Though it is marketed as a visualization tool, yt is a fully-fledged analysis platform for FLASH.
- Since yt is well-factored, the visualization & analysis feature sets are distinct.
- Thus we can replace yt's plotting functionality with something easier and more empowering to the user.

Key Points

- Though it is marketed as a visualization tool, yt is a fully-fledged analysis platform for FLASH.
- Since yt is well-factored, the visualization & analysis feature sets are distinct.
- Thus we can replace yt's plotting functionality with something easier and more empowering to the user.
- (*cough* matplotlib)

Enter: FLASH Python Library

In a separate effort to provide a FLASH workflow management tool, we have Python package which lives in the source. This is a natural place for the new visualization tools to live.

Install via:

```
$ cd flash4/tools/
$ python setup.py install --user
```

Documentation is available on our website.

Output Module

In the `flash` namespace we now have access to the `output` module which contains several functions which return raw data that is suitable for plotting:

```
from flash.output import *

lineout(p1, p2, field, pf, **kwargs)
shock_on_lineout(p1, p2, field, pf, threshold=1e-06, min_threshold=1e-36, **kwargs)
slice(axis, coord, field, pf, bounds=None, resolution=600, method='nearest', **kwargs)
slice_gradient(axis, coord, field, pf, bounds=None, resolution=600, method='nearest', **kwargs)
```

Output Module

In the `flash` namespace we now have access to the `output` module which contains several functions which return raw data that is suitable for plotting:

```
from flash.output import *

lineout(p1, p2, field, pf, **kwargs)
shock_on_lineout(p1, p2, field, pf, threshold=1e-06, min_threshold=1e-36, **kwargs)
slice(axis, coord, field, pf, bounds=None, resolution=600, method='nearest', **kwargs)
slice_gradient(axis, coord, field, pf, bounds=None, resolution=600, method='nearest', **kwargs)
```

Lineouts can be piped to the matplotlib `plot()` function while the slices can be sent to `imshow()`.

Output Module

In the `flash` namespace we now have access to the `output` module which contains several functions which return raw data that is suitable for plotting:

```
from flash.output import *

lineout(p1, p2, field, pf, **kwargs)
shock_on_lineout(p1, p2, field, pf, threshold=1e-06, min_threshold=1e-36, **kwargs)
slice(axis, coord, field, pf, bounds=None, resolution=600, method='nearest', **kwargs)
slice_gradient(axis, coord, field, pf, bounds=None, resolution=600, method='nearest', **kwargs)
```

Lineouts can be piped to the matplotlib `plot()` function while the slices can be sent to `imshow()`.

Projections could be easily added.

A Quick Example

In a terminal, run:

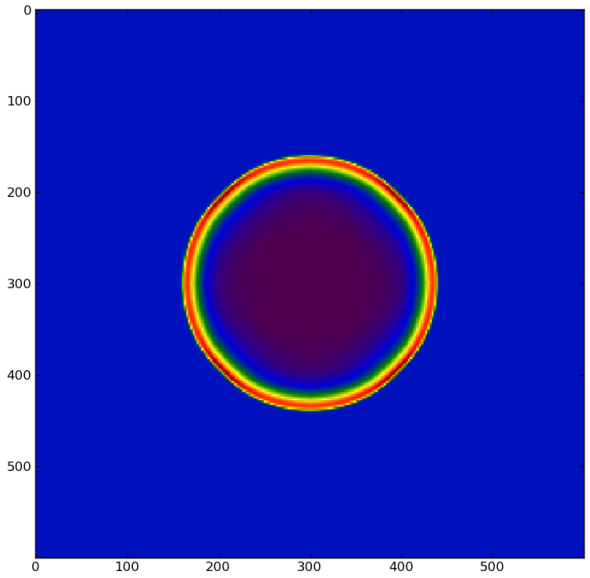
```
$ ./setup -auto Sedov; cd object/
$ make -j 20
$ mpirun -n 20
```

Then in Python, run:

```
from flash import output
import matplotlib.pyplot as plt
x, y, z = output.slice(2, 0.0, 'dens', "<path to chk>")
plt.imshow(z)
```

A Quick Example

You should see something like:



What is yt doing?

- Under the covers, yt has file handlers called plotfiles (`pf`) which live in plot collections (`pc`).

What is yt doing?

- Under the covers, yt has file handlers called plotfiles (`pf`) which live in plot collections (`pc`).
- On the `pf` live Hierarchy objects (aliased `h`) which provide a common interface for common operations (ray, slice, projection, etc) for all supported file type.

What is yt doing?

- Under the covers, yt has file handlers called plotfiles (`pf`) which live in plot collections (`pc`).
- On the `pf` live Hierarchy objects (aliased `h`) which provide a common interface for common operations (ray, slice, projection, etc) for all supported file type.
- These operations follow a pattern whereby they return special mappings keyed by fields (dens, etc). For flash, `pf.h.slice()` will return an `amr_slice[field]`.

What is yt doing?

If this wasn't confusing enough, these mapping are *lazily evaluated*. The fields don't necessarily exist until you ask for them:

```
In [7]: amr_slice.fields
Out[7]: ['dens', 'px', 'py', 'pz', 'pdx', 'pdy', 'pdz', 'x', 'y', 'z']

In [8]: amr_slice['targ']
Out[8]: array([ 0.47239542,  0.47150037,  0.4828257 , ...,  0.        ,
                0.        ,  0.        ])

In [9]: amr_slice.fields
Out[9]: ['dens', 'px', 'py', 'pz', 'pdx', 'pdy', 'pdz', 'x', 'y', 'z', 'targ']
```

What is the output module doing?

- The point of the output module is to abstract a lot of these under-the-cover yt issues.

What is the output module doing?

- The point of the output module is to abstract a lot of these under-the-cover yt issues.
- Moreover, it is faster than pure yt because it caches the special hierarchy mappings to prevent excessive re-reads (*ie* changing the resolution will only read in all the slice data the first time).

What is the output module doing?

- The point of the output module is to abstract a lot of these under-the-cover yt issues.
- Moreover, it is faster than pure yt because it caches the special hierarchy mappings to prevent excessive re-reads (*ie* changing the resolution will only read in all the slice data the first time).

```
output.ray_cache
output.slice_cache
```

What is the output module doing?

Furthermore since we are sitting on the yt analysis layer, we have access to all of their capabilities - including derived fields.

What is the output module doing?

Furthermore since we are sitting on the yt analysis layer, we have access to all of their capabilities - including derived fields.

```
from yt.data_objects.field_info_container import add_field

# register electron density field
def _edens(field, data):
    return data['ye'] * data['dens'] * data['sumy'] * 6.022E23

add_field ('edens', function=_edens, take_log=True)

# use this field with output functions
x, y, z = output.slice(2, 0.0, 'edens', "<path to chk>")
```

Summary

- The yt back-end is great and gets us 90% of the way there. However, its front end visualization is a little too crippled for daily use.

Summary

- The yt back end is great and gets us 90% of the way there. However, its front end visualization is a little too crippled for daily use.
- Using matplotlib instead gives us the perfect combination of data model and view.

Summary

- The yt back end is great and gets us 90% of the way there. However, its front end visualization is a little too crippled for daily use.
- Using matplotlib instead gives us the perfect combination of data model and view.
- Some convenience functions which glue these two together have already been written. More can be added and already have a place to live!

Questions

Image source: http://www.fotopedia.com/items/flickr-2200500024