BERKELEY LAB

LAWRENCE BERKELEY NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

The Impact of Community Codes in Astrophysics

Anshu Dubey

With slides from
Brian O'Shea (Enzo)
Matt Turk (Enzo and yt)

August 13, 2014

The Astrophysics Community

- Had an early culture of releasing research software starting in the early eighties
  - N-body codes for many-body gravitational interactions
    - Nbody_x went from Nbody_1 to Nbody_6
    - Barnes and Hut tree code
  - Hydrodymanics with ZEUS-2D, and later ZEUS-3D
  - SPH codes such as Hydra and Gadget
- Over time public codes became more sophisticated
  - AMR appeared in FLASH is early 2000
  - Shock-capturing MHD and radiation hydro also started to appear

The Astrophysics Community

- The tradition has continued with many more codes appearing over the years
  - Some codes are extensible
  - Capabilities added over time
- A comprehensive list can be found at the Astrophysics Source Code Library

  http://ascl.net/code/all/page/4/limit/100/order/title/listmode/full
  - Not only Astro codes, but all codes useful to Astro

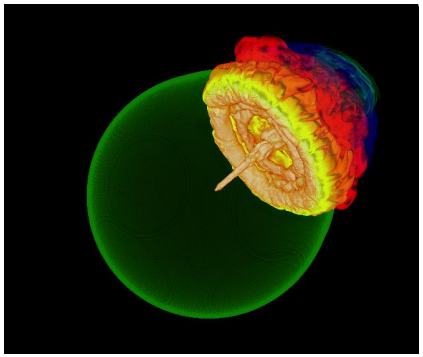
The Astrophysics Community

- Most codes started as research software and became public later
  - Were developed as closed and used by the core group for years before being released (Enzo)
- Others were developed with the intent of being made public as soon as possible (FLASH)
- With the arrival of “yt” there is yet another paradigm shift
  - An analysis and visualization packaged developed by the astrophysicists for astrophysicists
  - Building the community and development by the community from the very beginning

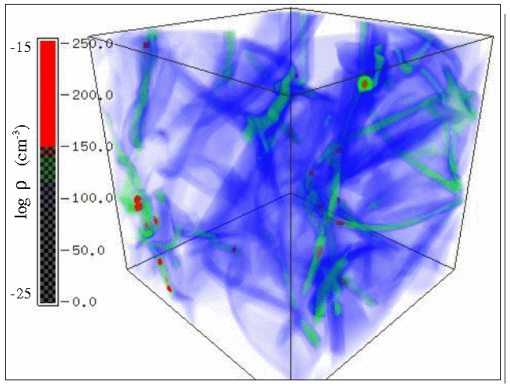
Why did Community Codes Flourish in Astrophysics

Why do some communities adopt the open source model while others do not ?

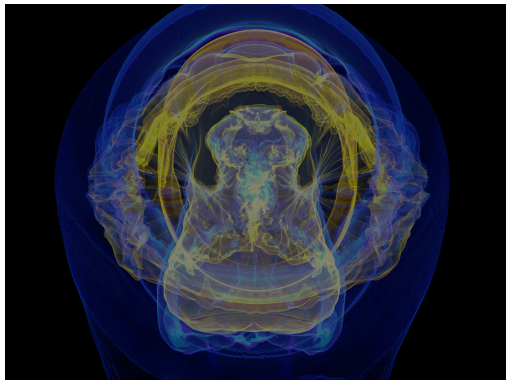
Astrophysics Simulations Need Multi-physics and Multi-scale



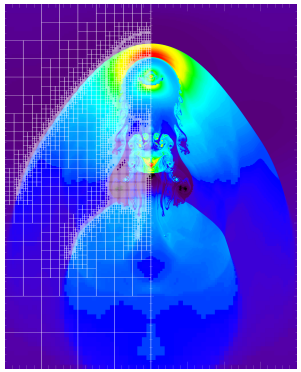
Type !a Supernova



Gravitational collapse/Jeans instability

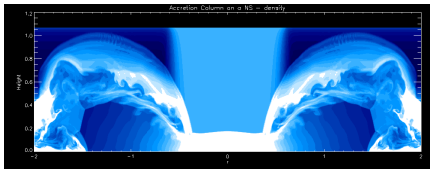


Galaxy Cluster Merger



Intracluster interactions

- Mesh methods: Explicit (gas dynamics), semi-Implicit (gravitational potential), and implicit (radiation)
- Particle methods: tracers, massive, charged
- Point-wise calculations: EOS, source terms
- AMR for data and computation compression



Shortly: Relativistic accretion onto NS

Developing and maintaining such complex codes is beyond the resources of capabilities of individuals or even small groups: Community codes are the solution

What About Other Communities ?

- Community/open-source approach more common in areas which need multi-physics and/or multi-scale
- A visionary sees the benefit of software re-use and releases the code
- Sophistication in modeling advances more rapidly in such communities
- Others keep their software close for perceived competitive advantage
  - Repeated re-invention of wheel
  - General advancement of model fidelity slower

Let us examine what does it take to build a community code

“Cathedral and the Bazaar”, Eric S. Raymond

- **The *Cathedral* model**
  - Code is available with each software release
  - Development between releases is restricted to an exclusive group of software developers.
    - GNU Emacs and GCC are presented as examples.
  - Central control models

- **The *Bazaar* model**
  - Code is developed over the Internet in view of the public.
  - Raymond credits Linus Torvalds, leader of the Linux kernel project, as the inventor of this process.
  - Distributed control models

Scientific codes

- Mostly follow the cathedral model
- Many reasons are given, some valid, others spring from bias
- The valid ones
  - Scientists tend to be skeptical of software engineering
  - The code quality becomes hard to maintain
  - Hard to find financial support for gate keeping and general maintenance
  - Typical user communities are too small to effectively support the bazaar model
  - The reward structure for majority of potential contributors is incompatible
- The not so valid ones
  - Codes are far too complex
  - Competitive advantage from owning the code

The real reason many times is simply the history of the development of the code and the pride of ownership

Scientific Community Codes Can Follow Several Different Paths :

- The most common path
  - Someone wrote a very useful piece of code that several people in the group started using
  - Collaborations happened
  - People moved and took the code with them
  - Critical mass of users achieved, code becomes popular
- No focused effort to build the code
  - Usually very little software process involved
  - For the whole code, limited shelf life

A More Sustained Path

- Sometimes enough like minded people take it a step further
  - Some long term planning might result in better engineered code
  - Thought given to extensibility and for future code growth
  - As the code grows so does its community supported model
- This model is still relatively rare.
  - The occurrences are increasing

A Desirable Path

- Explicit funding to build a code for a target community
- Implied support for the design phase
- The outcome is expected to be long lasting and well engineered
- The occurrences are even rarer
  - And it is getting increasingly harder
- When it works outcome is more capable and longer lasting codes

Open Source Benefits

- Nobody can pull the plug
  - Users have the source code, free to use and modify, in perpetuity.
- Users don’t have to pay
  - They pay with their time and attention and what they give back.
  - Stakeholders are power
  - Not all stakeholders are equivalent
    - User count may not be as helpful as vocal collaborators

Software Engineering and Process

- Strong interfaces and encapsulation (enforced by the language or build system) enables community participation.
  - Users can customize in many different ways
    - Depends somewhat on the code architecture
    - Add needed interfaces on top of infrastructure
    - Use derived classes
  - Open-source means they can control customization

The Benefits of the Bazaar model

- Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone
  - More varied test cases that demonstrate bugs
  - Debugging can be effectively parallelized.
  - The infrastructure limitations are quickly exposed
- Capability addition is rapid, codes can do more
  - A corollary to that is a good extensible design
  - Users always want something more and/or something different from what is available
  - Greater knowledge pool operating together, more possibility of innovation

The Pitfalls of the Bazaar model

- Many of the benefitting reasons can equally easily go the other way
  - Bigger knowledge pool can also mean more conflicting opinions
  - Prioritizations can become extremely challenging
- Gatekeeping can become a huge challenge for maintaining software quality
  - Scientific codes have their own peculiarities for verification and validation that can be extremely challenging
  - The orchestration of capability combination is harder when there is physics involved because many times it just won't play well together

Other Considerations : User Expertise

- Novice users – execute one of included applications
  - change only the runtime parameters
- Most users – generate new problems, analyze
  - Generate new Simulations with initial conditions, parameters
  - Write alternate and/or derived functions for specialized output
- Advanced users – Customize existing functions
  - Add small amounts of new code needed by their application
- Expert – new research
  - Completely new algorithms and/or capabilities
  - Can contribute to core functionality

Distribution Policies

- The licensing agreement
  - How restrictive ?
- Distribution control
  - Who can get the code
  - Should there be a registration requirement
- What is included in the release
  - The degree of support for released components
- How often to release
  - Trade-off between making capabilities available quickly and the overhead of releasing

Contribution Policies

- Balancing contributors and code distribution needs
  - Contributors want their code to become integrated with the code so it is maintained, but may not want it released immediately
    - Not exercised enough
    - Contributor may want some IP protection
- Maintainable code requirements
  - The minimum set needed from the contributor
    - Source code, build scripts, tests, documentation
- Agreement on user support
  - Contributor or the distributor
- Add-ons : components not included with the distribution, but work with the code

Community Building

- Popularizing the code alone does not build a community
- Neither does customizability – different users want different capabilities

So what does it take ?

- Enabling contributions from users and providing support for them
- Including policy provisions for balancing the IP protection with open source needs
- Relaxed distribution policies – giving collective ownership to groups of users so they can modify the code and share among themselves as long as they have the license

More inclusivity => greater success in community building

An investment in robust and extensible infrastructure, and a strong culture of user support is a pre-requisite

Examples : FLASH Developed and Distributed by One Institution

- Under sustained funding from the ASC alliance program
- One of the expected outcomes was a public code
  - Use the same code for many different applications
    - All target applications were for reactive flows
- Diverging camps from the beginning
  - Camp 1: Produce a well architected modular code
  - Camp 2: Let's build what can be used for science soon
- Both goals hard to meet in the near term
- Two parallel development paths started
  - Not enough resources to sustain both
  - Camp 2 won out
- Took three iterations of code refactoring to get robust framework built

FLASH's Community

- Originally designed for thermo-nuclear flashes
  - Expanded to include N-body capabilities through particles
  - Over the years many other physics capabilities got added
- Now serves many communities in Astrophysics, and also serves HEDP and CFD/FSI
  - Most rapid increase in modeling laser experiments
  - Only open code for the community
- Very little modification to the basic infrastructure needed to accommodate these capabilities
- Additions typically prove to be synergistic for all the communities

Some Statistics

- Releases roughly every 6-8 months
- In version 4.x now
- More than 850 papers have used it for obtaining results
- Increasingly more sophisticated contributions from outside the core group
  - Sink particles
  - Incompressible Navier-Stokes solver
  - Primordial Chemistry

Community Building

- Took several years
- Started with collaborations with the Center scientists
- Alumni of the center took the culture and the code with them
  - Their students and post-docs adopted the code
- Tutorials on-site and at scientific conferences to promote
  - Tutorials had hands-on sessions and help for user's specific problems
- Easy customizability built into the infrastructure helped
  - As did the included ready to run examples
- Increasing capabilities enable tackling more complex and higher fidelity modeling

The greatest impact in popularizing the code though was relative ease in getting started, quick turn-around for user's questions and hand holding provided through the mailing lists

Enzo : Transitioned from close to open source

- Started as a closed code
  - From 1996-2003
- First public release in March 2004
  - Mostly cathedral model
- Has now moved very close to a bazaar model
  - 25 contributors (~12 active developers) at >10 institutions
  - ~200 people on enzo-users mailing list (~50% active?)
  - Financial support from NSF (AST, OCI, PHY), NASA, and DOE

Complementary community: **yt** (http://yt-project.org)

Development Model

- Entirely distributed development model
  - Small number of developers per institution
- Use code forks / pull requests to move features from development branches to the main branch
- Almost all discussion on archived public mailing lists
  - And on Google docs

Community

- Most developers are astrophysicists “scratching their own itch”
- Development spurred by ~1.5 workshops/year
  - And periodic task-oriented “code sprints”
  - Many streams of funding
- Enthusiastic and heavily involved user/developer community

Challenges:

- No leader => hard to make major code revisions
- Part-time developers: distractions, less incentive to do “boring but important” infrastructure development
- Significant work required to build consensus and keep community together


The yt Project
Growing & Engaging a community
of practice

Building a community …

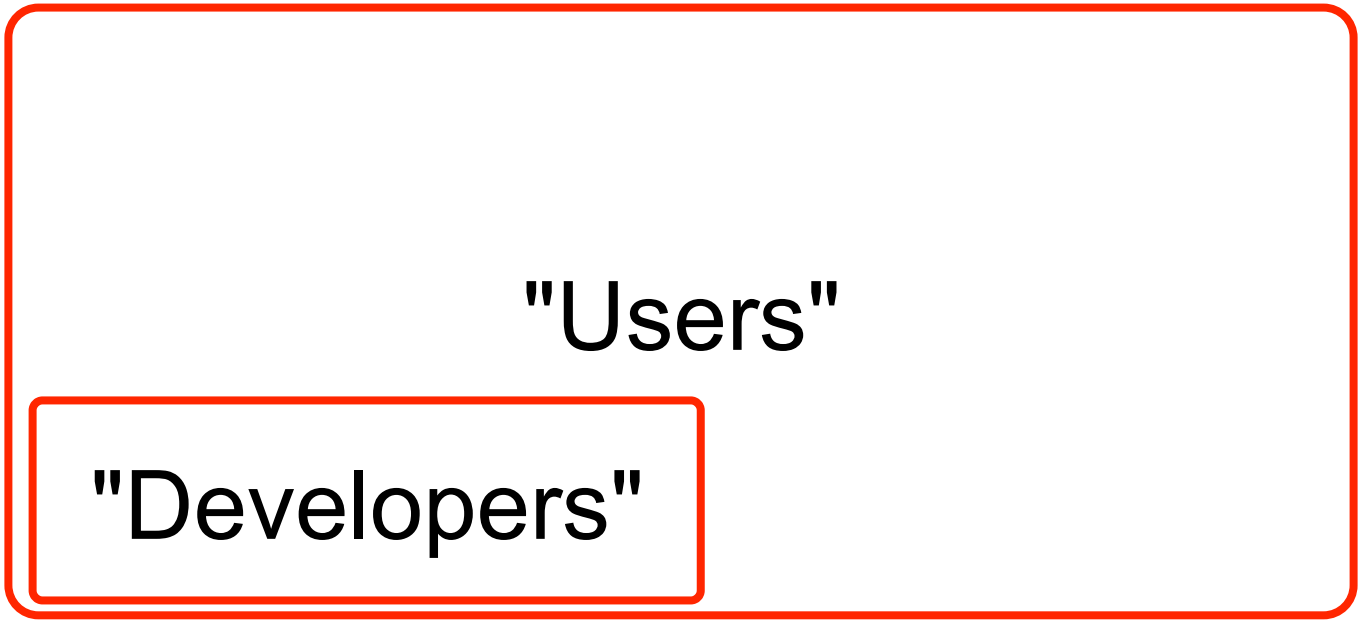
The technical and social aspects

Traditional View of Scientific Development

"Users"

"Developers"

Most Scientific Development

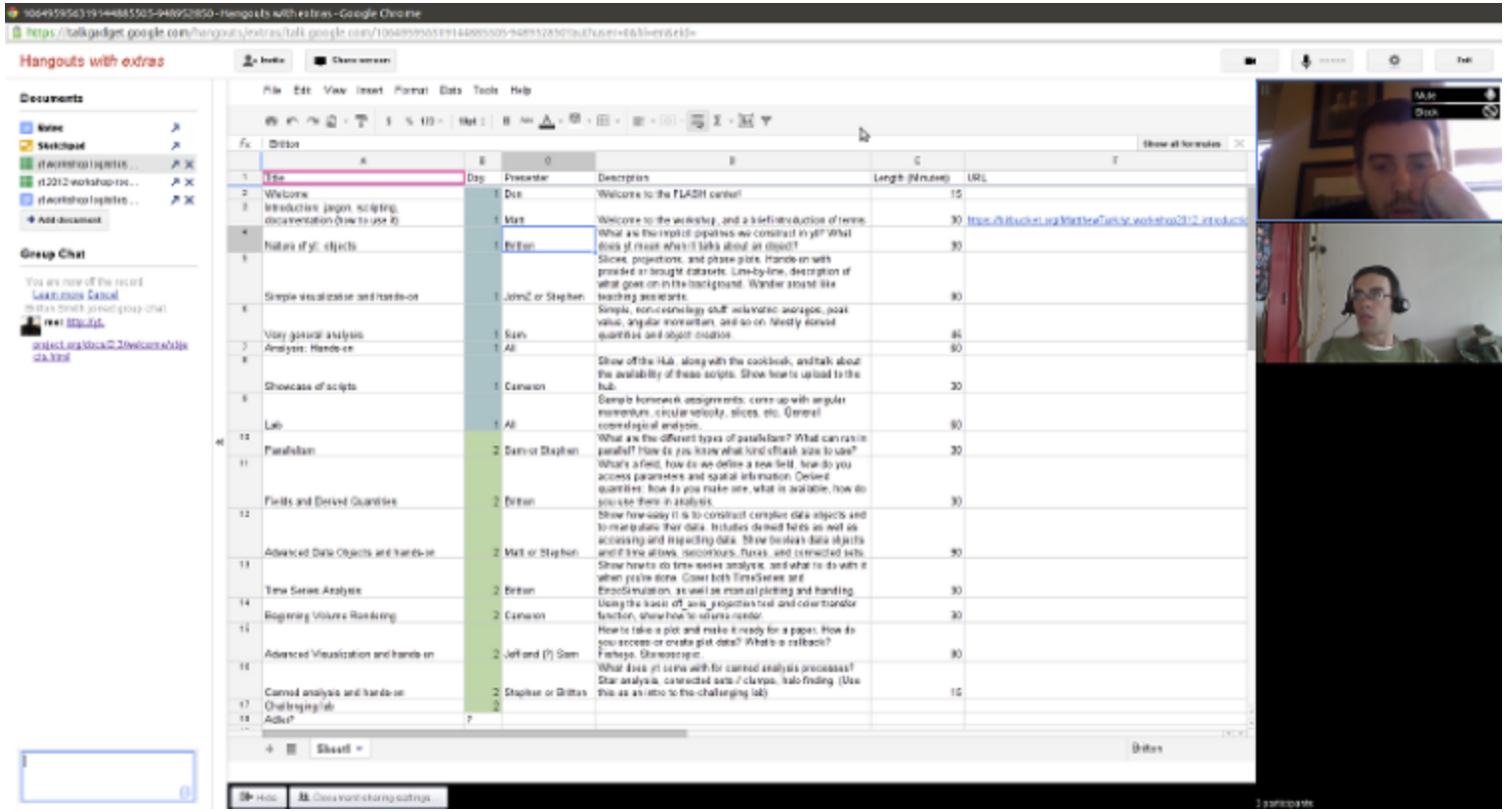


Community of Practice

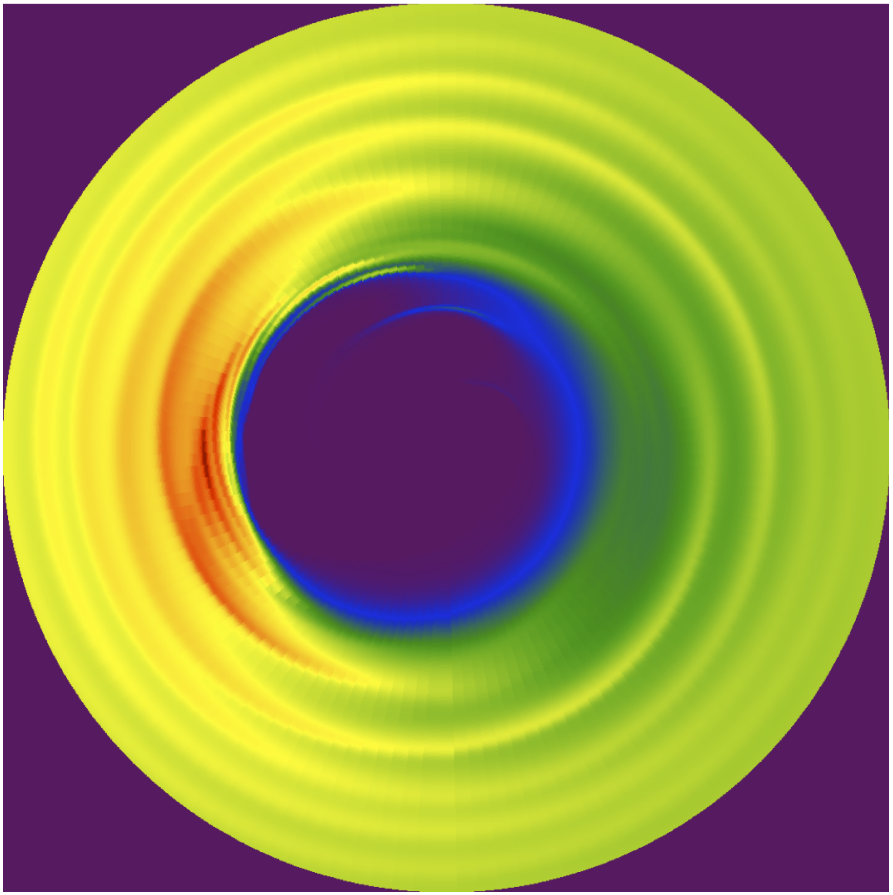
"Devusers"

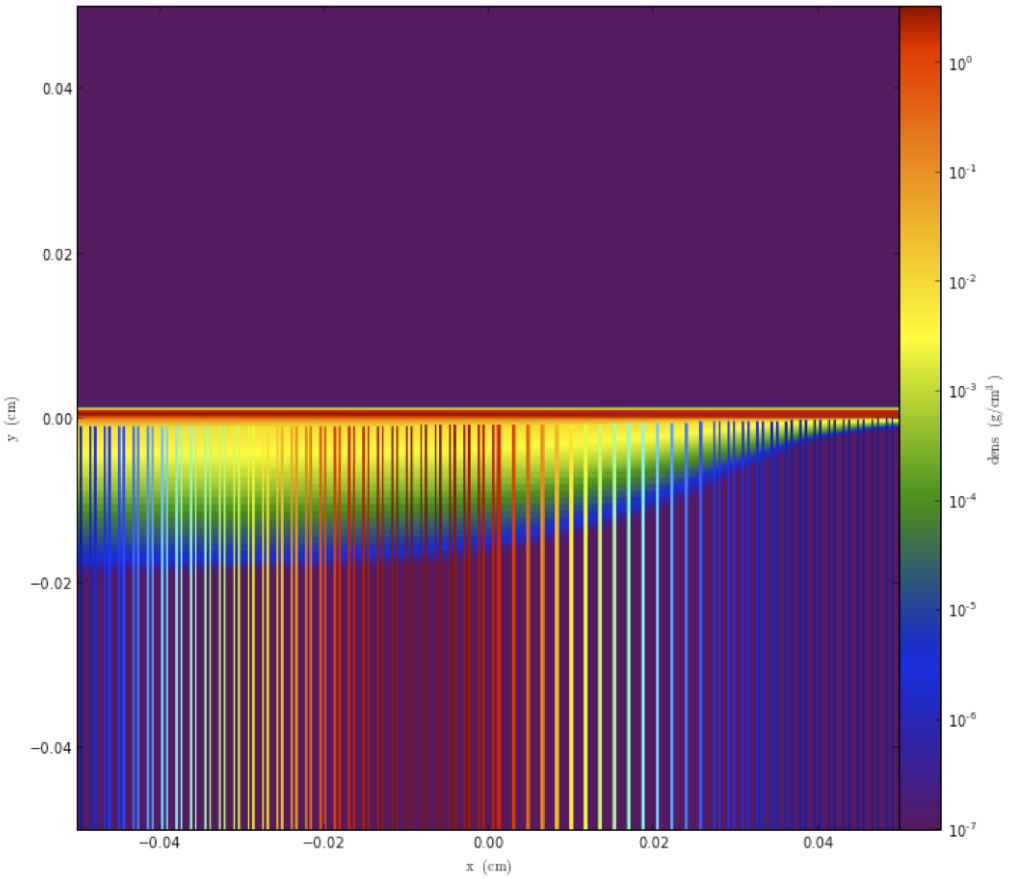
Foster a community of peers,
not a community of elites.

Immediate



Immediate

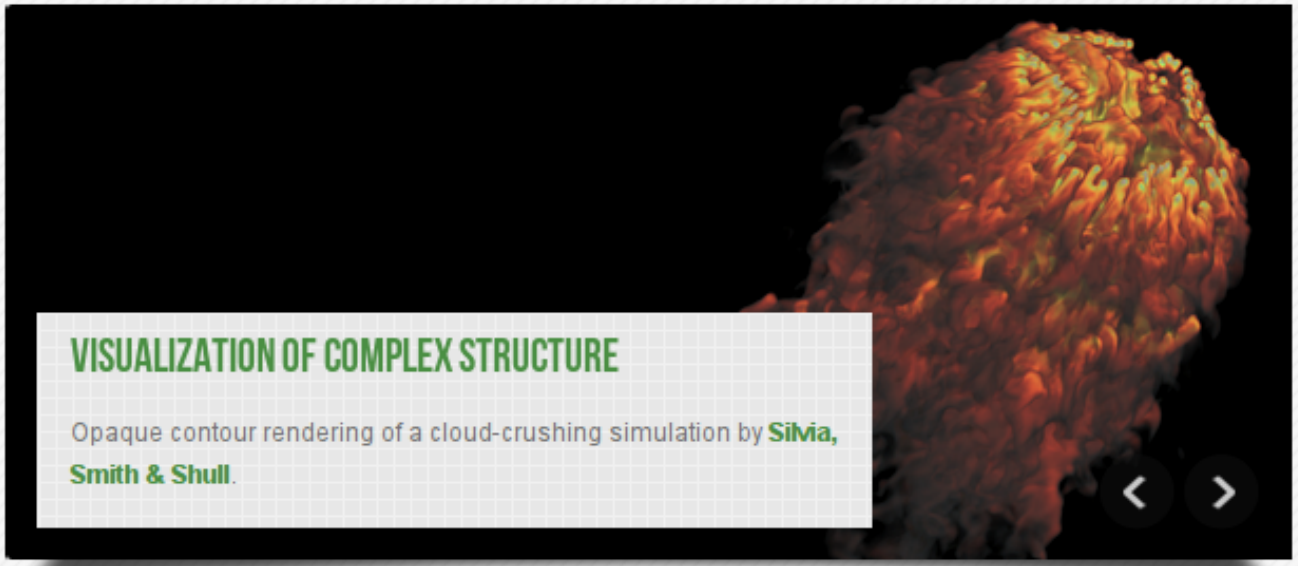




BERKELEY LAB
LAWRENCE BERKELEY NATIONAL LABORATORY

THE YT PROJECT

ASTROPHYSICAL SIMULATION ANALYSIS AND VIZ



VISUALIZATION OF COMPLEX STRUCTURE

Opaque contour rendering of a cloud-crushing simulation by **Silvia, Smith & Shull**.

DETAILED DATA ANALYSIS AND VISUALIZATIONS, WRITTEN BY WORKING ASTROPHYSICISTS AND DESIGNED FOR PRAGMATIC ANALYSIS NEEDS.

DATA-DRIVEN

Inspect your data

yt is designed to provide a consistent, cross-code interface to analyzing and visualizing

COMMUNITY

Participants welcome!

yt is composed of a friendly community of users and developers. We want to make it

FREE SOFTWARE

Open Source, Open Science

yt is developed completely in the open, released under the GPL license. The developers are

In a Nutshell, yt...

...has had 7,345 commits made by 42 contributors
representing 113,588 lines of code

...is mostly written in Python
with an average number of source code comments

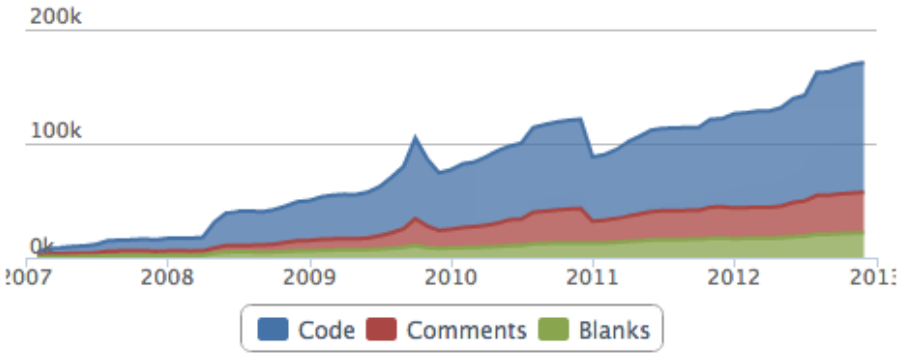
...has a well established, mature codebase
maintained by a large development team
with stable year-over-year commits

...took an estimated 29 years of effort (COCOMO model)
starting with its first commit in February, 2007
ending with its most recent commit about 17 hours ago

Languages

Python	77%	C	12%
JavaScript	5%	8 Other	6%

Lines of Code



Activity

30 Day Summary

Nov 14 2012 — Dec 14 2012

219 Commits
12 Contributors
including 1 new contributor
1 New Language :
TeX/LaTeX added Dec 13

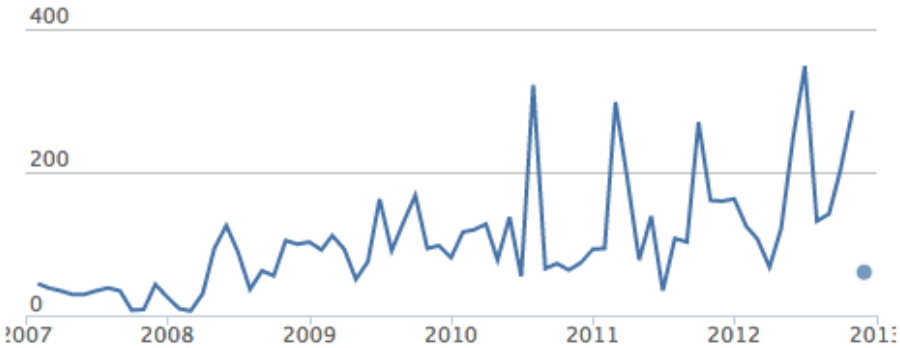
12 Month Summary

Dec 14 2011 — Dec 14 2012

2057 Commits
Up +384 (22%) from previous 12 months

26 Contributors
Up +3 (13%) from previous 12 months

Commits per Month



Common Threads

- Open source with a governance structure in place
- Trust building among teams
- Commitment to transparent communications
- Strong commitment to user support
- Either an interdisciplinary team, or a group of people comfortable with science and code development
- Attention to software engineering and documentation
- Understanding the benefit of sharing as opposed to being secretive about the code

General Impact

- Scientists can focus on developing for their algorithmic needs instead of getting bogged down by the infrastructural development
- Graduate students do not start developing codes from scratch
  - Look at the available public codes and converge on the ones that most meet their needs
  - Look at the effort of customization for their purposes
  - Select the public code, and build upon it as they need

Important to remember that they still need to understand the components developed by others that they are using, they just don't have to actually develop everything themselves. And this is particularly true of pesky detailed infrastructure/solvers that are too well understood to have any research component, but are time consuming to implement right

General Impact

- Researchers can build upon work of others and get further faster, instead of reinventing the wheel
  - Code component re-use
  - No need to become an expert in every numerical technique
- More reliable results because of more stress tested code
  - Goes back to argument for the bazaar model, enough eyes looking at the code will find any errors faster
  - New implementations take several years to iron out the bugs and deficiencies
  - Different users use the code in different ways and stress it in different ways
- Open-source science results in more reproducible results
  - Generally good for the credibility

General Impact

- Involvement in this community has strongly affected young scientists' career trajectories
  - They have produced more significant and far-reaching advances in their fields
  - They have understood the importance of rigorous verification and validation in the scientific process
  - They have learned to use the good software practices to their advantage
- Unexpected side benefits:
  - Arrival of yt spurred the development in Enzo
  - FLASH morphed into community code for areas that have very little to do with its original target
    - Took addition of a few physics models and there was a working code in a few months
  - New developments helped the Astro community

Conclusions

- There are many reasons why community codes are good and should be encouraged
  - Science and engineering by simulation needs more scrutiny into the methods and software
  - There is no need to keep reinventing the wheel
    - Starting from scratch for any computational project where implementations exist is a waste
    - This is especially true of book-keeping work
  - Scientists aren't trained to write good maintainable and performant software
    - Optimization blockers
    - Spaghetti codes
    - No rigorous verification program

The Solution : Let the professionals do the software design and engineering with a lot of input from the target user community (ideally some users will also be developers)

Culture self-propagates. So, it must be seeded directly.

"... it seems likely that significant software contributions to existing scientific software projects are not likely to be rewarded through the traditional reputation economy of science. Together these factors provide a reason to expect the over-production of independent scientific software packages, and the underproduction of collaborative projects in which later academics build on the work of earlier ones."

Howison & Herbsleb (2011)