Finite Element Methods at Realistic Complexities

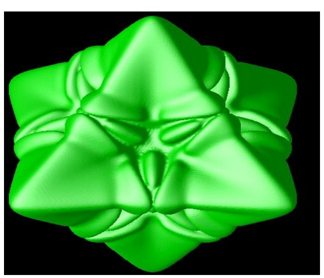
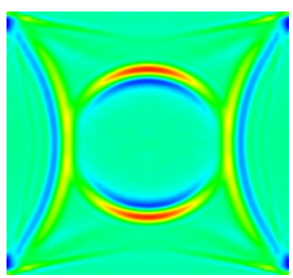
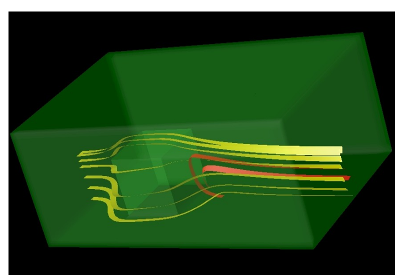
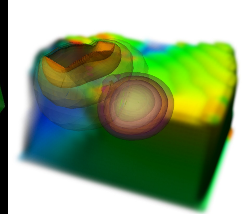
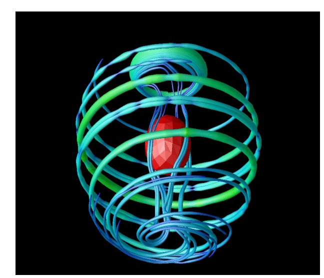
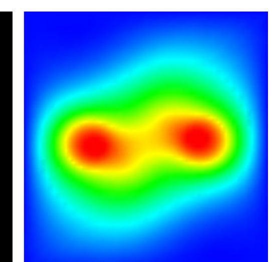
Wolfgang Bangerth

Texas A&M University

In collaboration with many many others around the world.

National Science Foundation
WHERE DISCOVERIES BEGIN

CIG COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

www.dealii.org

On Computational Science

This talk is about software.

Specifically:

How to write computational software for "real problems"?

...considering differences to "model problems" in:

- size
- complexity
- the way we develop it
- the way we teach it

Outline

- What we want
- How we can develop software that does this
- Experience with
  – the *deal.II* software library
  – the *ASPECT* mantle convection solver
- Conclusions

What we want

Goal: Simulate convection in Earth's mantle and elsewhere.

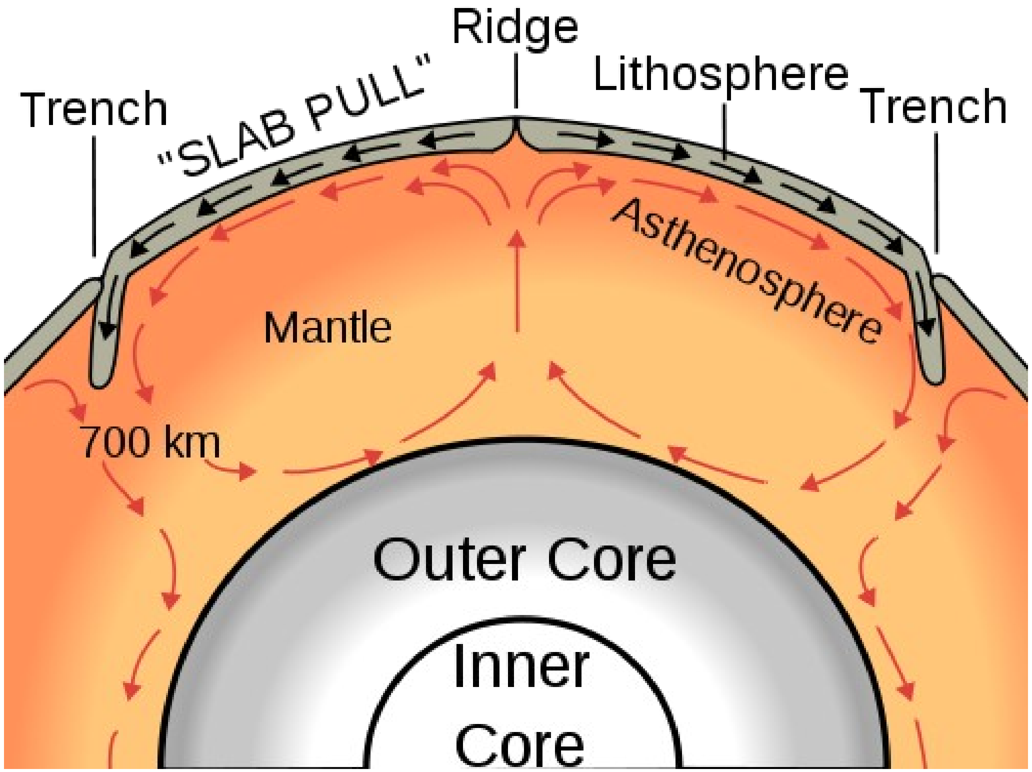
Our tool:

ASPECT – Advanced Solver for Problems in Earth's ConvecTion

http://aspect.dealii.org/

ASPECT

Goal: Simulate convection in Earth's mantle and elsewhere.



ASPECT

Goal: Simulate convection in Earth's mantle and elsewhere.

Questions:

- What drives plate motion?
- What is the thermal history of the earth?
- Do hot spots exist and how do they relate to global convection?
- Interaction with the atmosphere?
- When does mantle convection exist?
- What does that mean for other planets?

ASPECT – Challenges I

For convection in the earth mantle:

- Depth: ~35 – 2890 km
- Volume: ~10^{12} km^3
- Resolution required: <10 km
- Uniform mesh: ~10^9 cells
- Using Taylor-Hood (Q2/Q1) elements: ~$3 \cdot 10^{10}$ unknowns
- At 10^5–10^6 DoFs/processor: 30k-300k cores!

ASPECT – Challenges II

Thermal convection is described by the relatively "simple" Boussinesq approximation:

$$-\eta \Delta u + \nabla p = g \rho(T)$$

$$\nabla \cdot u = 0$$

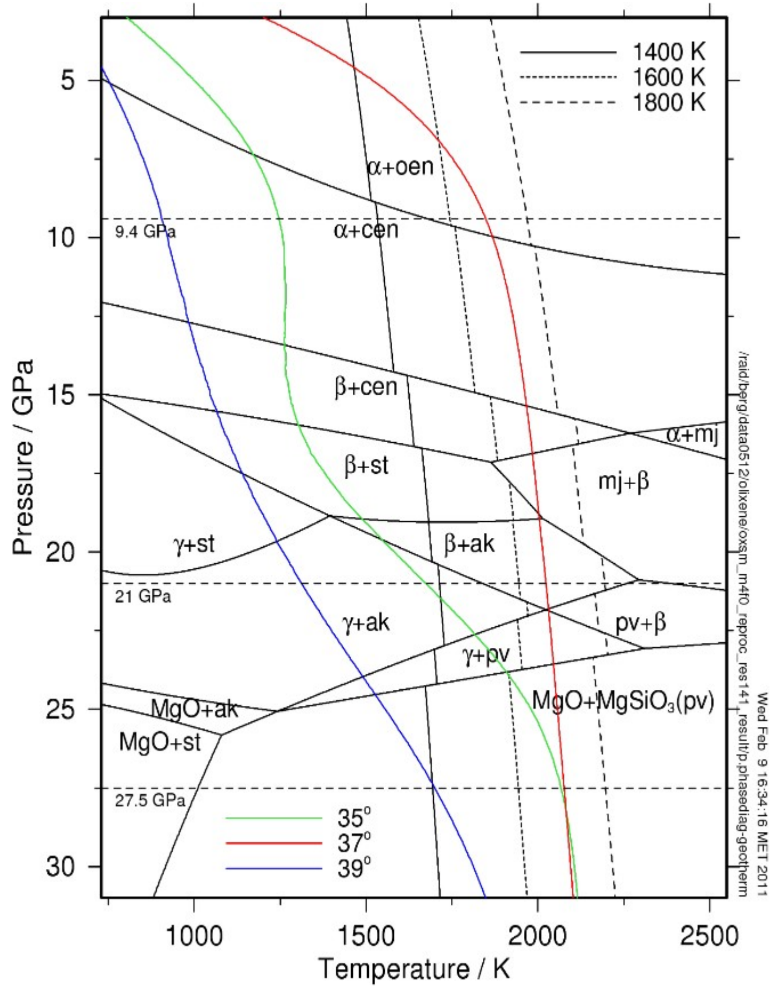
$$\frac{\partial T}{\partial t} + u \cdot \nabla T - \kappa \Delta T = \gamma + \alpha \left(\frac{\partial p}{\partial t} + u \cdot \nabla p \right) + 2 \eta \epsilon(u) : \epsilon(u)$$

This looks like a typical "model problem".

ASPECT – Challenges II

However, in reality:

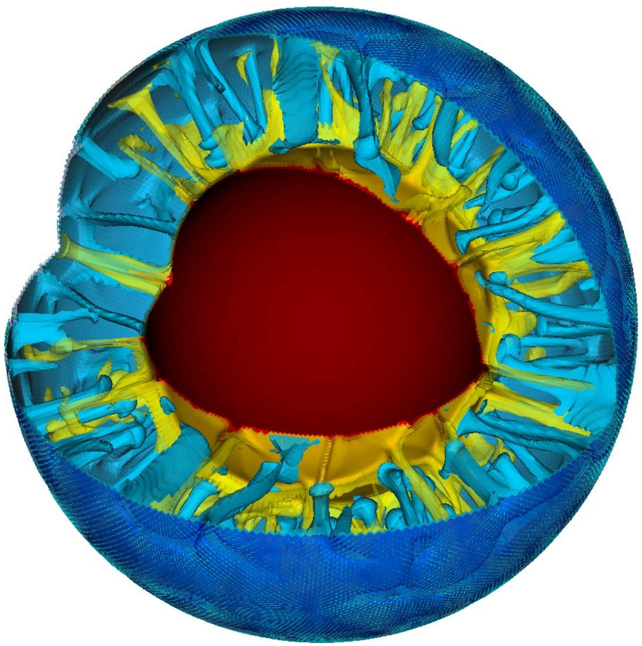
- All coefficients depend nonlinearly
  - pressure
  - temperature
  - strain rate
  - chemical composition
- Dependence is not continuous
- Viscosity varies by at least 10^{10}
- Material is compressible
- Geometry depends on solution

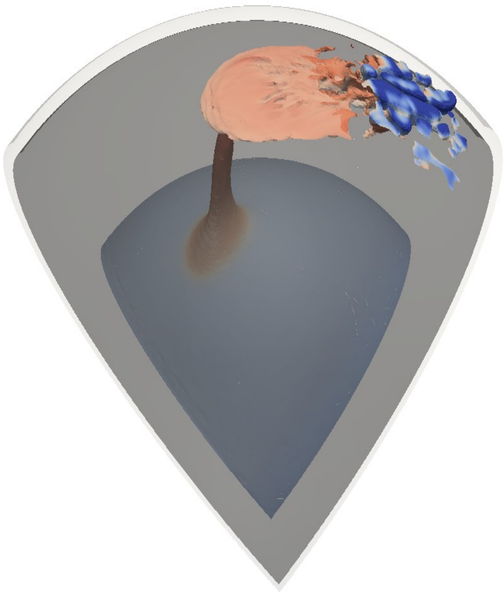


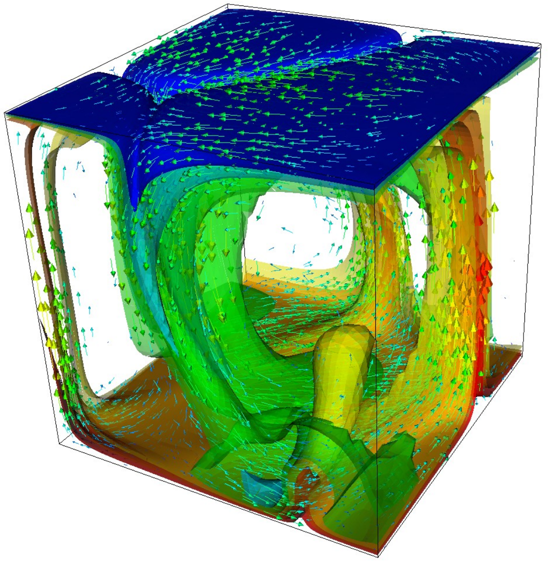
ASPECT – Challenges III

People want to change things:

- Geometries:
  - – global
  - – regional
  - – model problems







ASPECT – Challenges III

People want to change things:

- Geometries:
  – global
  – regional
  – model problems
- Material models:
  – isoviscous vs realistic
  – compressible vs incompressible
- Boundary conditions
- Initial conditions
- Add tracers or compositional fields
- … …

- What happens to the solution: postprocessing

ASPECT – Challenges IV

We need to think about the *whole* application:

- Adaptive meshes
- Nonlinear loops
- Scalable and efficient preconditioners
- Scale to 10,000s of cores
- Where we can cut corners to make things faster

Also, since this is a code for the community:

- Extensibility
- Ease of use
- Documentation
- Needs to fit into the geophysical workflow

ASPECT

Question:

How can we write such a code?

Surely, it will require 100,000s of lines of code!
(Recall: 20k lines of code per man-year.)

About software

Research software today:

- Typically written by graduate students
  – without a good overview of existing software
  – with little software experience
  – with little incentive to write high quality code
- Often maintained by postdocs
  – with little time
  – need to consider software a tool to write papers
- Advised by faculty
  – with no time
  – oftentimes also with little software experience

About software

Observation 1:

Most research software is not of high quality.

How does this affect our field? (Reproducibility? Archival? “Standing on the shoulders of giants”?)

About software

Observation 2:

There is a complexity limit to what we can get out of a PhD student.

About software

Solutions:

- Creating software is an art and science. So:

  What makes software successful?
  (Best practices? Lessons learned?)

  We could learn from the answers.

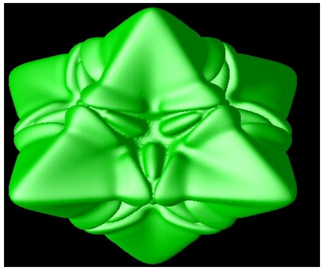
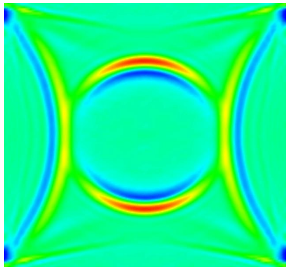
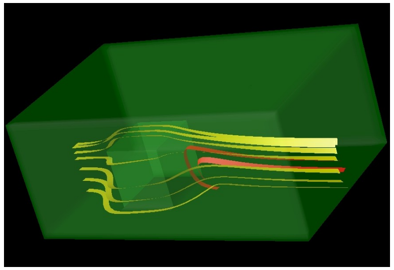
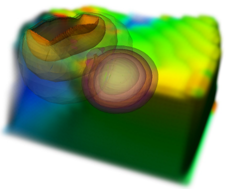
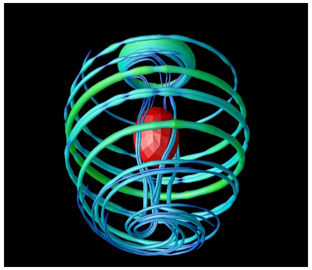
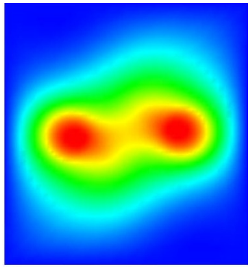
- Use what others have already done (and *use for free!*):
  – Matlab
  – Linear algebra packages like PETSc, Trilinos
  – Finite element packages like libMesh, FEniCS, deal.II
  – Optimization packages like COIN, CPLEX, SNOPT, …

- On this, build only what is application specific
- Use sound software design principles

Strategy 1: Libraries

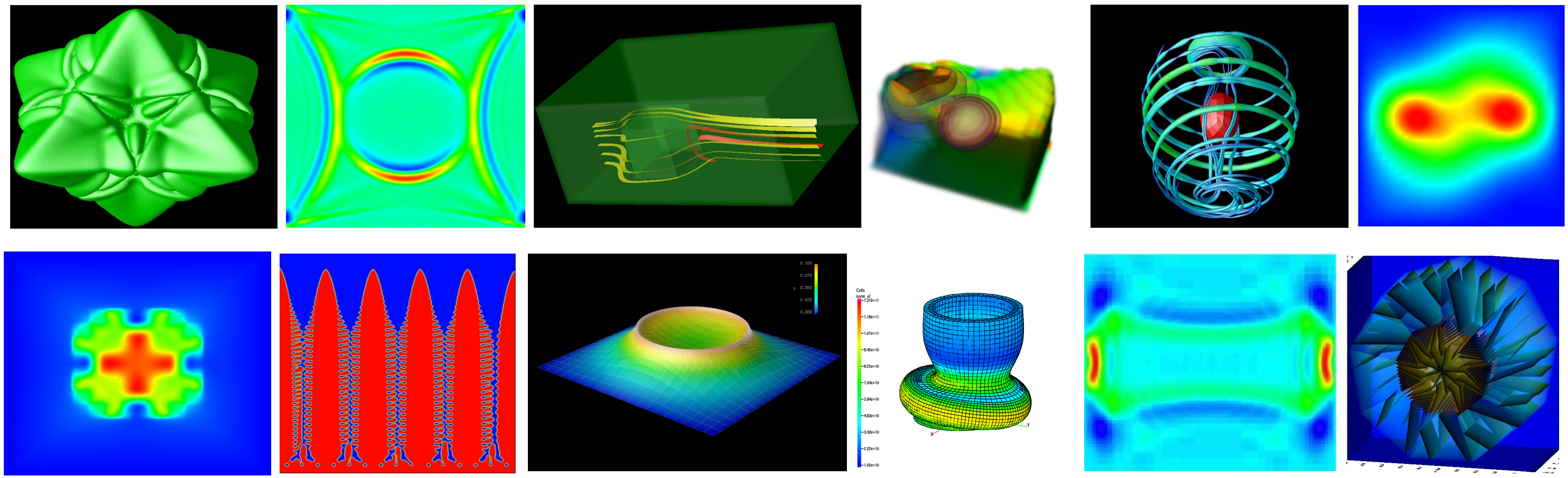
An example:

The *deal.II* library

deal.II

A library for finite element computations that supports...



...a large variety of PDE applications
tailored to non-experts.

www.dealii.org

We want a library that:

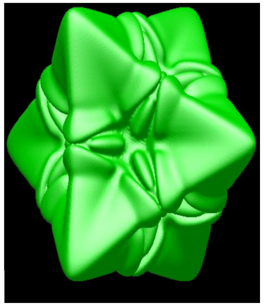
- Supports complex computations in many fields
- Is general (not area-specific)
- Has fully adaptive, dynamically changing 3d meshes
- Scales to 10,000s of processors
- Is efficient on today's multicore machines

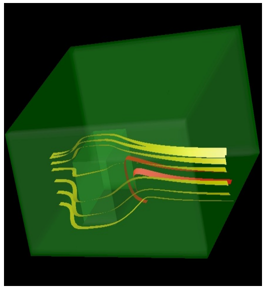
Fundamental premise:
Provide building blocks that can be used in many different ways, not a rigid framework.

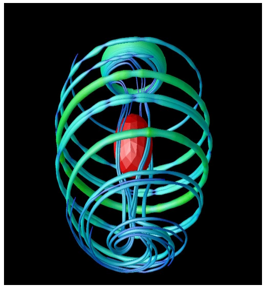
deal.II

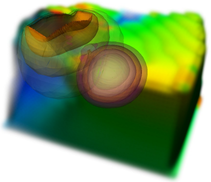
deal.II provides:

- Adaptive meshes in 1d, 2d, and 3d
- Interfaces to all major graphics programs
- Standard refinement indicators built in
- Many standard finite element types (continuous, discontinuous, mixed, Raviart-Thomas, ...)
- Low and high order elements
- Support for multi-component problems
- Its own sub-library for dense + sparse linear algebra
- Interfaces to PETSC, Trilinos, UMFPACK, ARPACK, ...
- Supports SMP + cluster systems





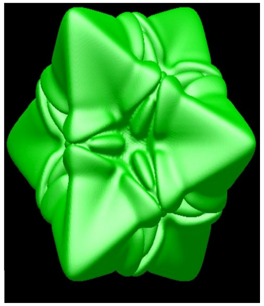


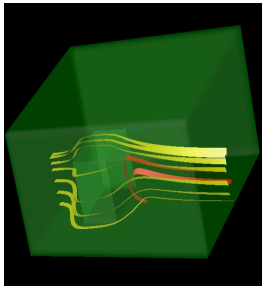


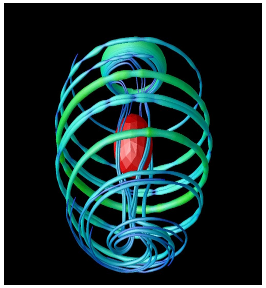
deal.II

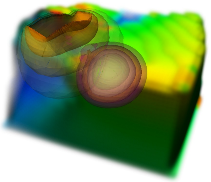
Status today:

- 500+ downloads per month
- 600,000 lines of code
- 10,000+ pages of documentation
- Portable build environment
- Used in teaching at many universities





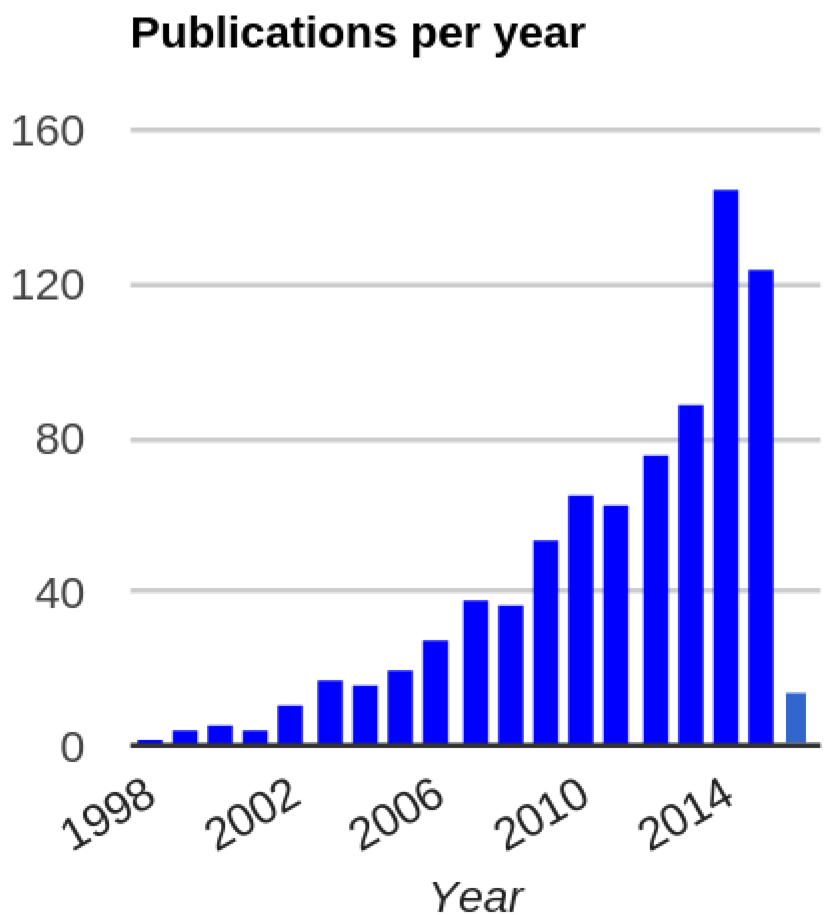




deal.II

www.dealii.org

Publications using deal.II:



Examples

Examples of what can be done with deal.II (2013 only):

- Biomedical imaging
- Brain biomechanics
- E-M brain stimulation

- Microfluidics
- Oil reservoir flow
- Fuel cells
- Transonic aerodynamics
- Foam modeling
- Fluid-structure interactions
- Atmospheric sciences

- Quantum mechanics
- Neutron transport
- Nuclear reactor modeling

- Numerical methods research

- Fracture mechanics
- Damage models
- Solidification of alloys
- Laser hardening of steel
- Glacier mechanics
- Plasticity
- Contact/lubrication models

- Electronic structure
- Photonic crystals

- Financial modeling

- Chemically reactive flow
- Flow in the Earth mantle

What makes such projects successful?

General observations:

Success or failure of scientific software projects is not decided on technical merit alone.

The *true* factors are beyond the code!
It is not enough to be a good programmer!

In particular, what counts:

- Utility and quality
- Documentation
- Community

All of the big libraries provide this for their users.

Ease of use

Take utility as an example:

- Lots of error checking in the code
- Extensive testsuites
- Meaningful error messages and assertions rather than cryptic error codes

- Cataloged use cases
- FAQs
- Well documented examples of debugging common problems

Documentation/Education

Take documentation and education as an example:

- Installation instructions/README
- Within-function comments
- Function interface documentation
- Class-level documentation
- Module-level documentation
- Worked “tutorial” programs
- Recorded, interactive demonstrations

Example: deal.II has 10,000+ HTML pages. 170,000 lines of code are actually documentation (~10 man years of work). There are 60 recorded video lectures on YouTube.

Examples

deal.II comes with ~50 tutorial programs:

- From small Laplace solvers (~100s of lines)
- To medium-sized applications (~1000s of lines)
- Intent: – teach deal.II
  – teach advanced numerical methods
  – teach software development skills

What a student can expect

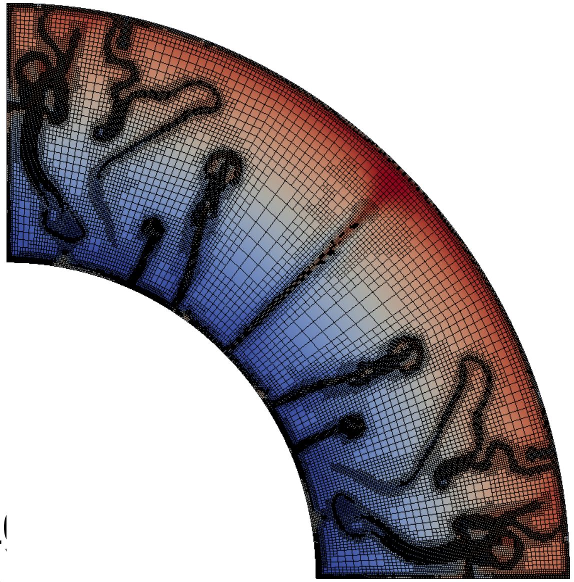
Because they no longer have to write most of their codes, a student can achieve in 3 years with deal.II:

- Solve a complex model
- With realistic geometries, unstructured meshes
- Higher order finite elements
- Multigrid-based solver
- Parallelization
- Output in formats for high-quality graphics
- Results almost from the beginning: a wide variety of tutorials allow a gentle start

Examples

There are also large applications (not part of deal.II):

- *Aspect:* Advanced Solver for Problems in Earth Convection
  – ~50,000 lines of code
  – Open source: *http://aspect.dealii.org/*

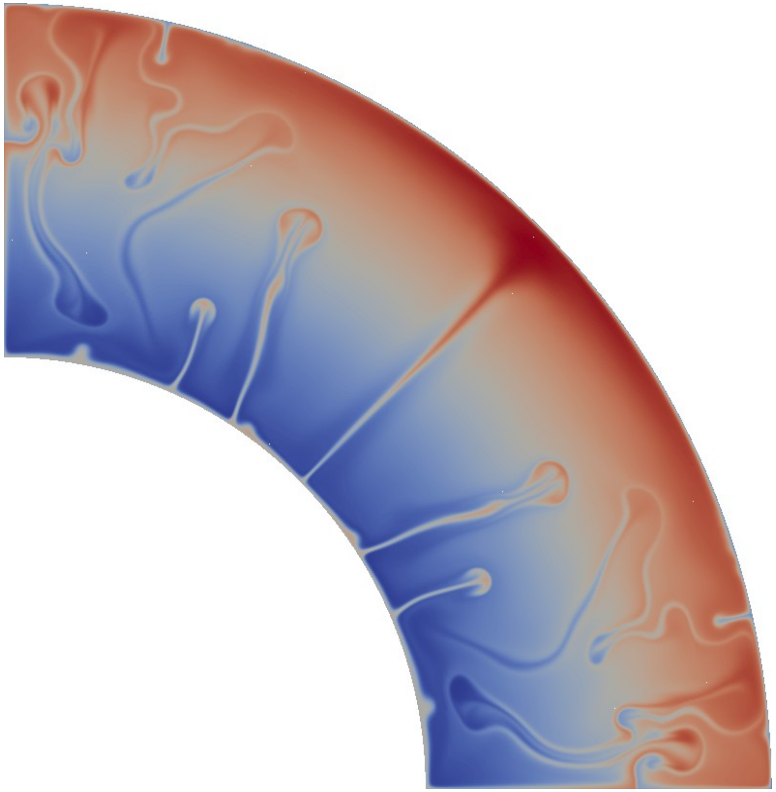


- *OpenFCST:* A fuel cell simulation packa
  – Supported by an industrial consortium
  – Open source: *http://www.openfcst.org/*

- Optical tomography code for medical imaging
  – 55,000 lines of code

- ...

ASPECT



Aspect:
Advanced Solver for Problems in Earth ConvecTion

(Bangerth, Heister, and many others around the world; funding by CIG and NSF)

ASPECT – Approaches

Among the mathematical techniques we use are:

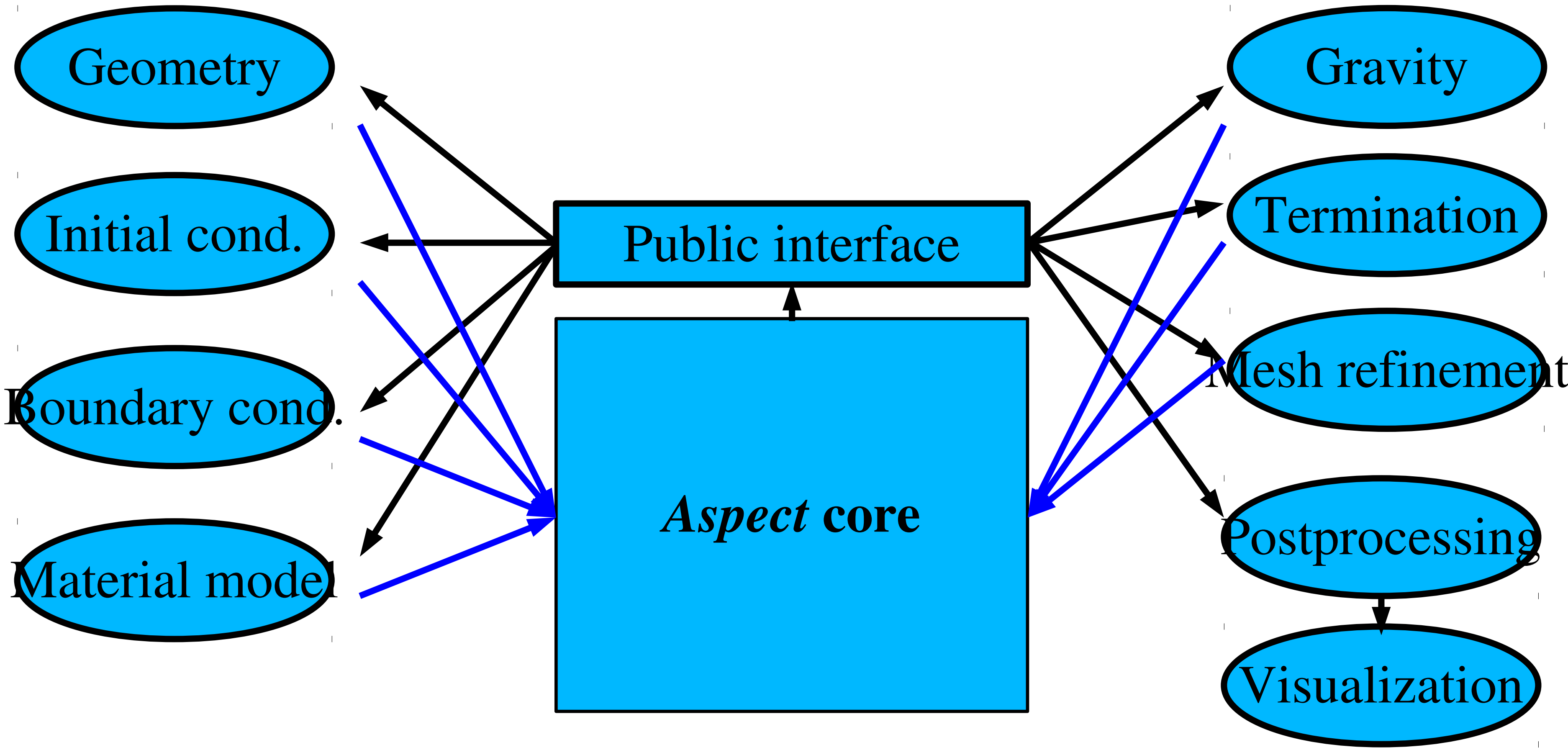
- Higher order time stepping schemes
- Higher order finite elements
- Fully adaptive, dynamically changing 3d meshes
- Newton's method for the nonlinearity
- Silvester/Wathen-style block preconditioners with FGMRES
- Algebraic multigrid for the elliptic part
- Parallelization using MPI, threads, and tasks

To make the code usable by the community:

- Use object-oriented programming
- Make it modular, separate concerns
- Extensive documentation
- Extensive and frequent testing

ASPECT – Approaches

***Aspect* is very modular:** It is extended by a number of isolated "plugin" sub-systems:



ASPECT – Approaches

How hard is this in practice:

- Core simulator:
  – ~5,000 lines of code (1,100 semicolons)
- Runtime parameters, checkpoint/restart, etc:
  – ~1,800 lines of code (350 semicolons)

- Problem statement (geometry, materials, boundary and initial conditions, postprocessing, etc):
  – ~43,000 lines of code (8,600 semicolons)

ASPECT – Approaches

How hard is this in practice:

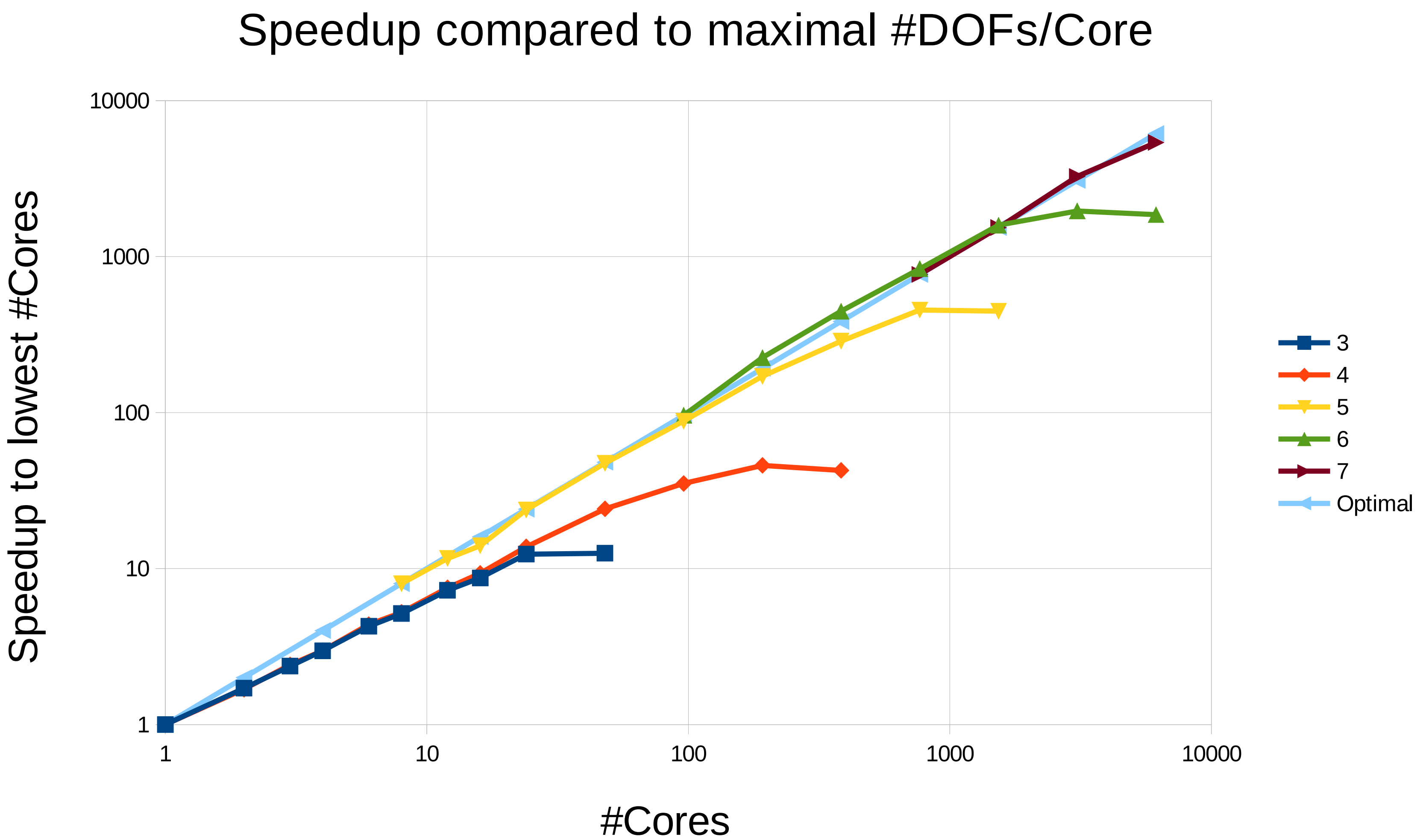
- Core simulator:
  – ~5,000 lines of code (1,100 semicolons) – mostly stable
- Runtime parameters, checkpoint/restart, etc:
  – ~1,800 lines of code (350 semicolons) – mostly stable
- Problem statement (geometry, materials, boundary and initial conditions, postprocessing, etc):
  – ~43,000 lines of code (8,600 semicolons) – growing

Conclusions:

- Using advanced numerical methods does not lead to prohibitive code growth
- Developers can focus on application specifics
- Most parts are self-contained and accessible to "newbies"

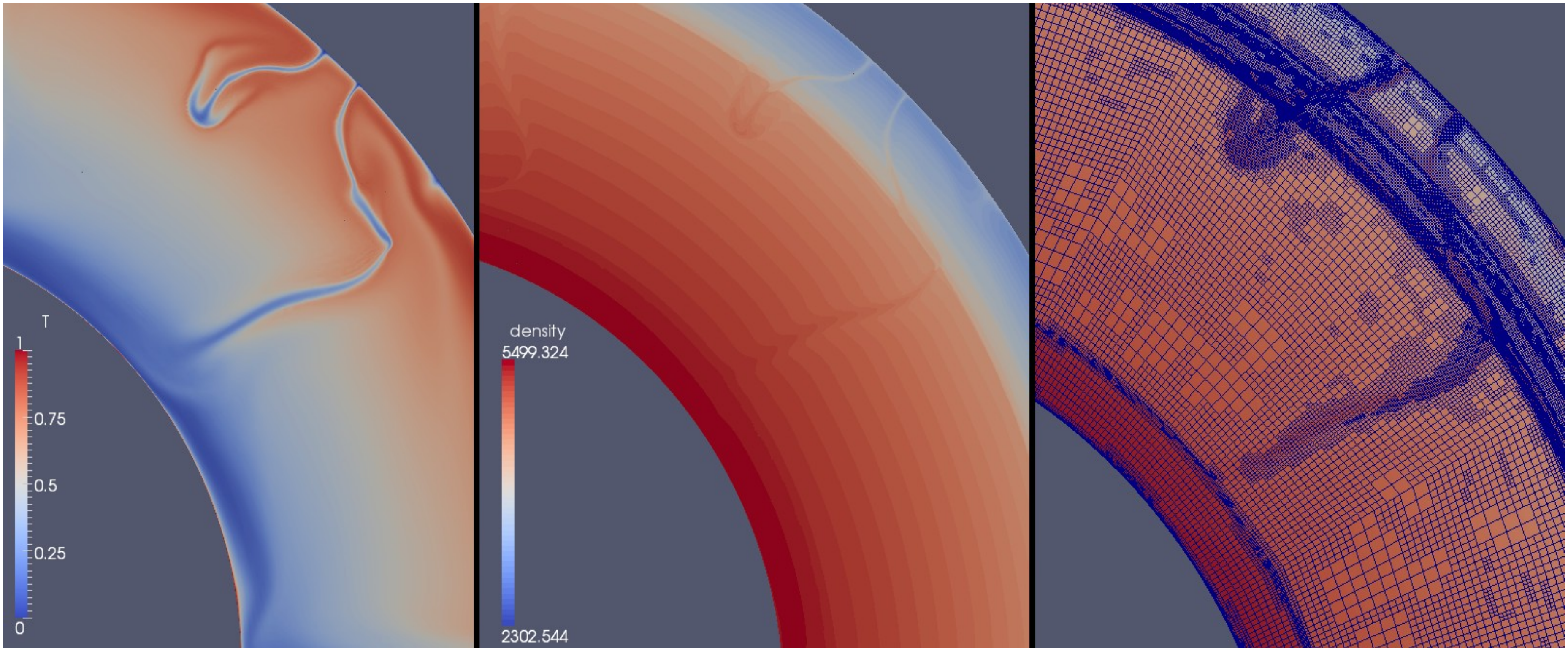
ASPECT – Results

Scalability:



ASPECT – Results

Example of a subducting slab lying on the 660km discontinuity:



Temperature

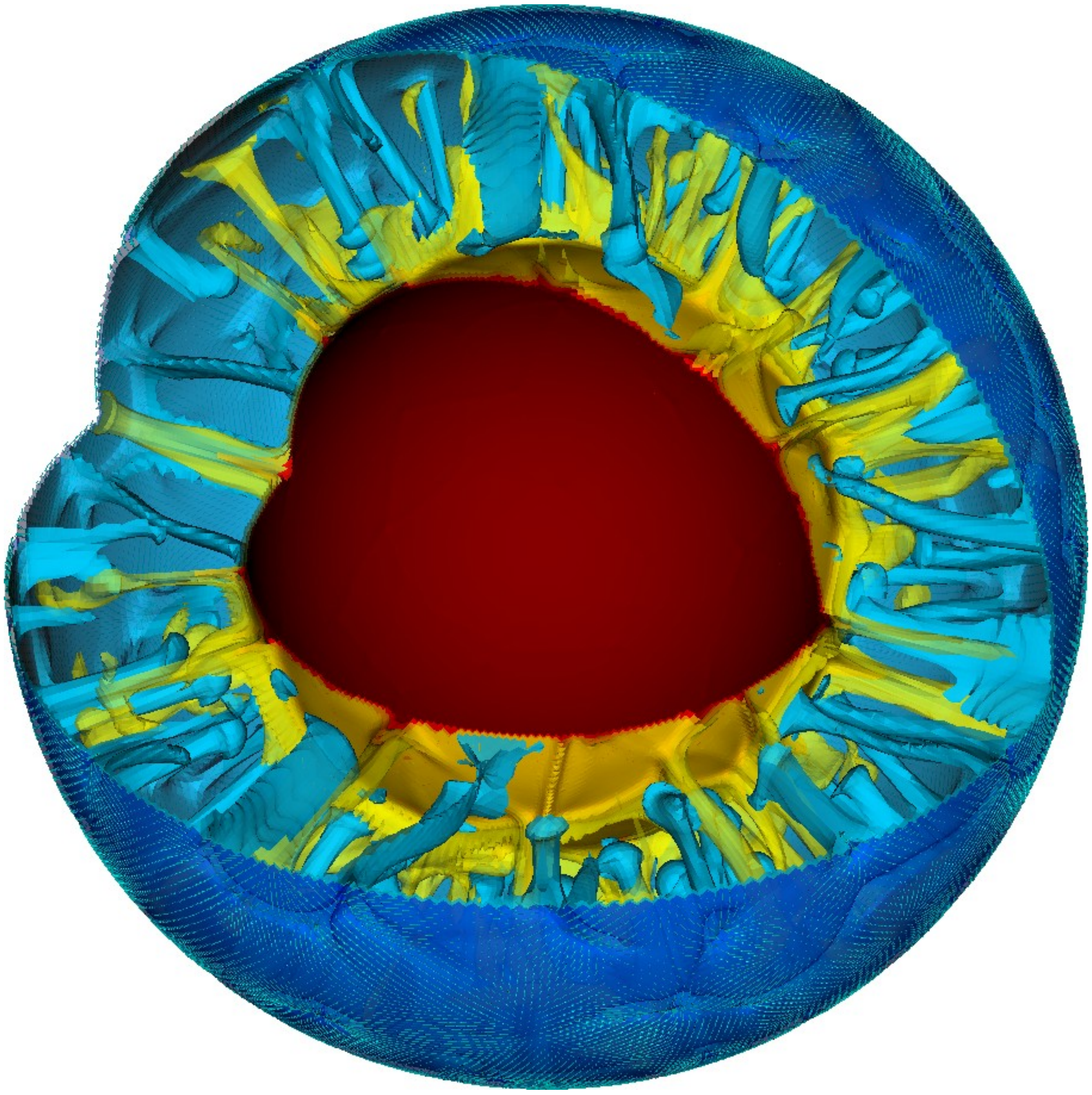
Density

Mesh

Credit: Thomas Geenen, University of Utrecht

ASPECT – Results

Example of 3d computations:



Effects

What this development model means for us:

- We can solve problems that were previously intractable
- Methods developers can demonstrate applicability
- Applications scientists can use state of the art methods
- Our codes become far smaller:
  – less potential for error
  – less need for documentation
  – lower hurdle for “reproducible” research (publishing the code along with the paper)
- More impact/more citations when publishing one's code

Effects

What this development model means for our community:

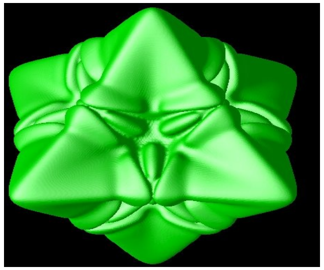
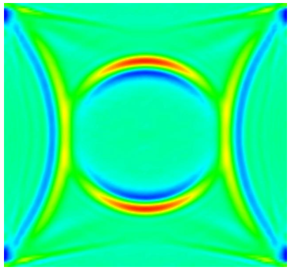
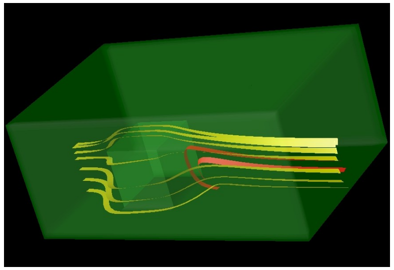
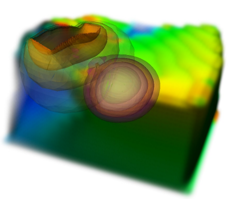
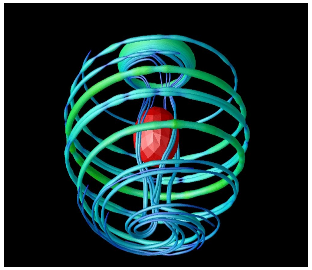
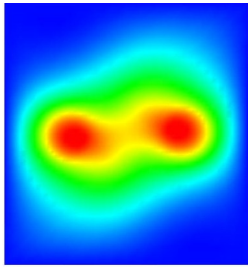
- Faster progress towards “real” applications
- Leveling the playing field – excellent online resources are there for *all*
- Raising the standard in research:
  – can't get 2d papers published any more
  – reviewers can require state-of-the-art solvers
  – allows for easier comparison of methods

Conclusions

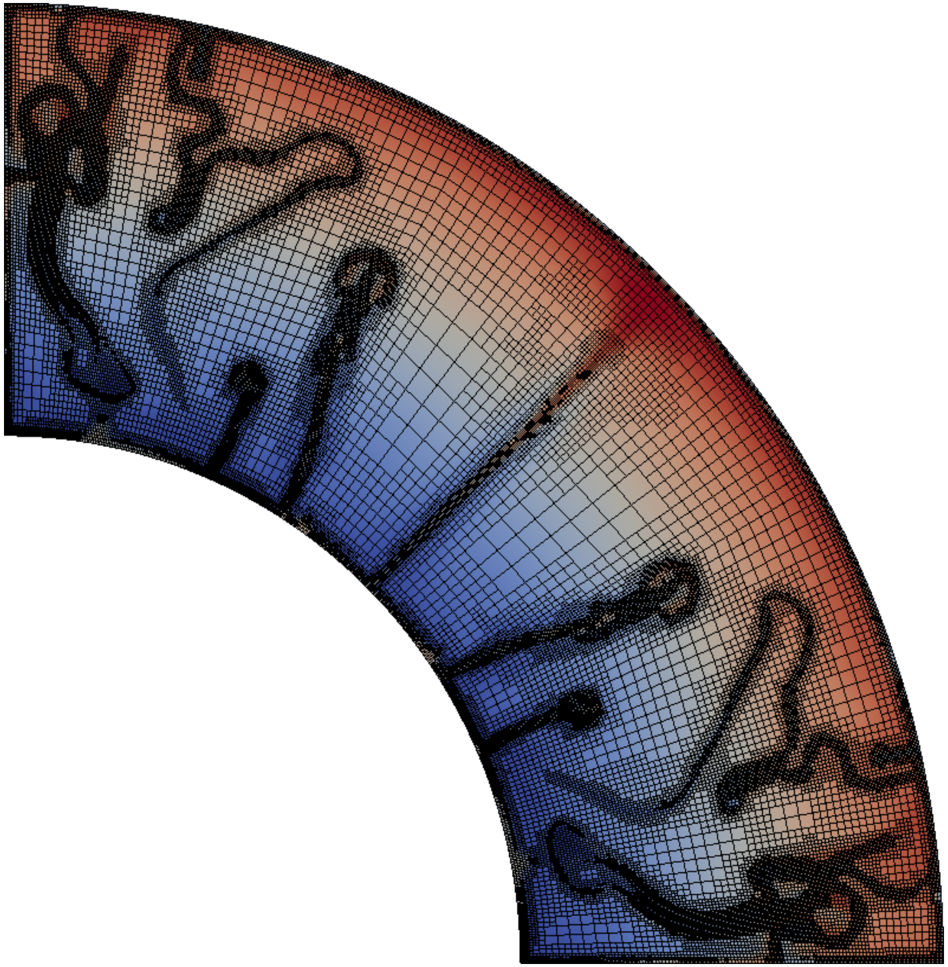
Computational science has spent too much time where everyone writes their own software.

By building on existing, well written and well tested, software packages:

- We build codes *much* faster
- We build better codes
- We can solve more realistic problems

www.dealii.org



More information:

http://www.dealii.org/

http://aspect.dealii.org/