Adaptive mesh refinement: Theory, practice, and applications in geodynamics

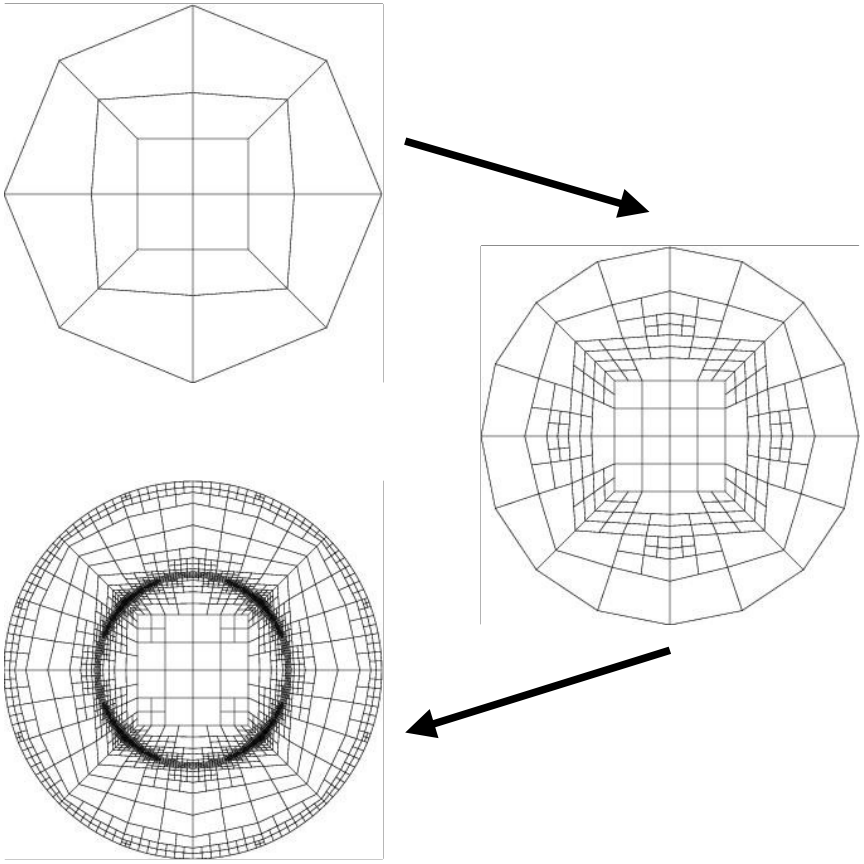
Wolfgang Bangerth
Department of Mathematics
Texas A&M University

Supported by: NSF, DHS, DoE, Sloan Foundation, CIG

Adaptivity modifies meshes as necessary

Philosophy of local mesh refinement:

- Solve on a rather coarse grid
- Compute an error criterion
- If error < tolerance, then stop
- Otherwise refine mesh
- Solve again on finer grid



Advantage: We can use meshes adapted to the solution and/or what we are interested in

Disadvantage: We have to solve more than once, and we need more sophisticated algorithms

Questions about adaptivity

- **Will we gain anything?** This depends on
  - whether we need meshes fitted to geometric features
  - whether we need fully adapted time varying meshes
  - the type of the equation

- **How can we generate adaptive meshes?**
  - mesh generators
  - adaptive mesh refinement using error estimators and indicators

- **How to use them in our codes?**
  - What do we need for existing codes?
  - What do we need for new codes?

- **A few examples**

Adaptivity is good for geodynamics!

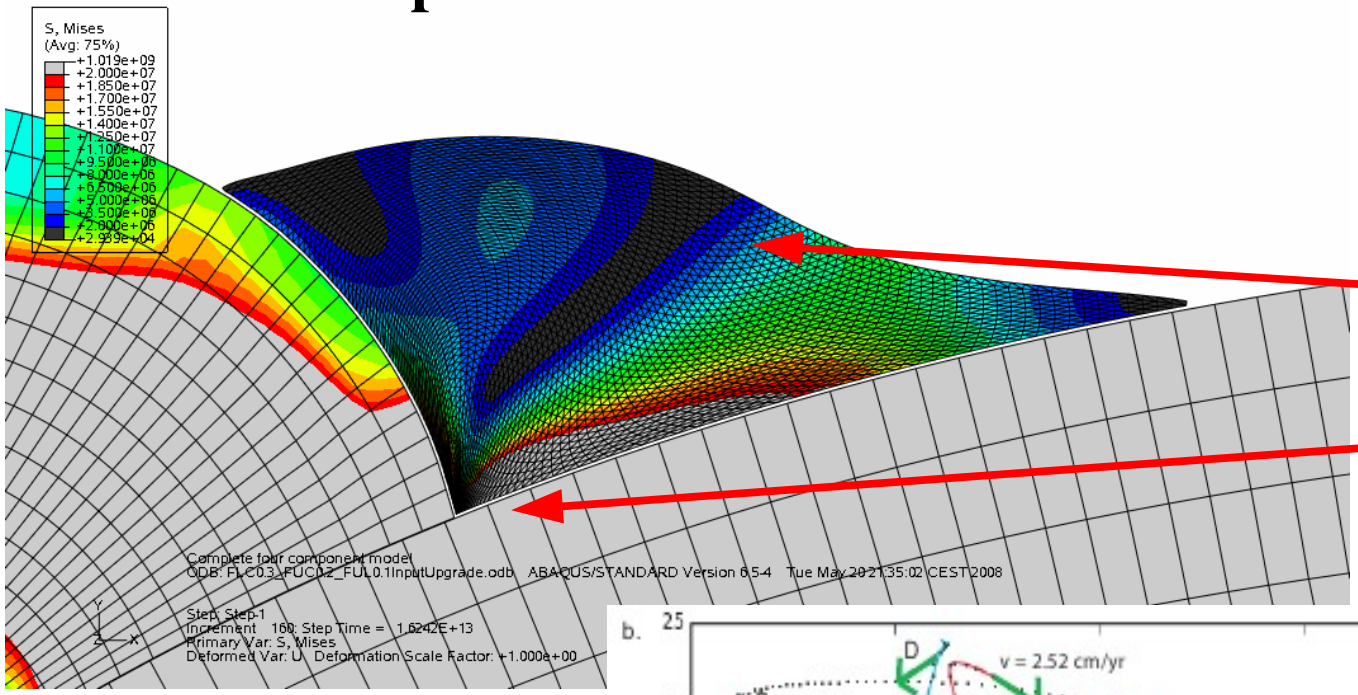
- Adaptive meshes promise to reach the same accuracy with (orders of magnitude) less effort
- Can focus degrees of freedom where the solution shows significant variation, achieving resolution otherwise impossible
- Avoid generation of fine meshes by generating them as necessary from coarse meshes

Adaptivity is good if and only if
"the action is localized"

Fortunately, that's frequently the case in geodynamics
(e.g. stress at fault tips/plate boundaries)!

Adaptivity is good for geodynamics!

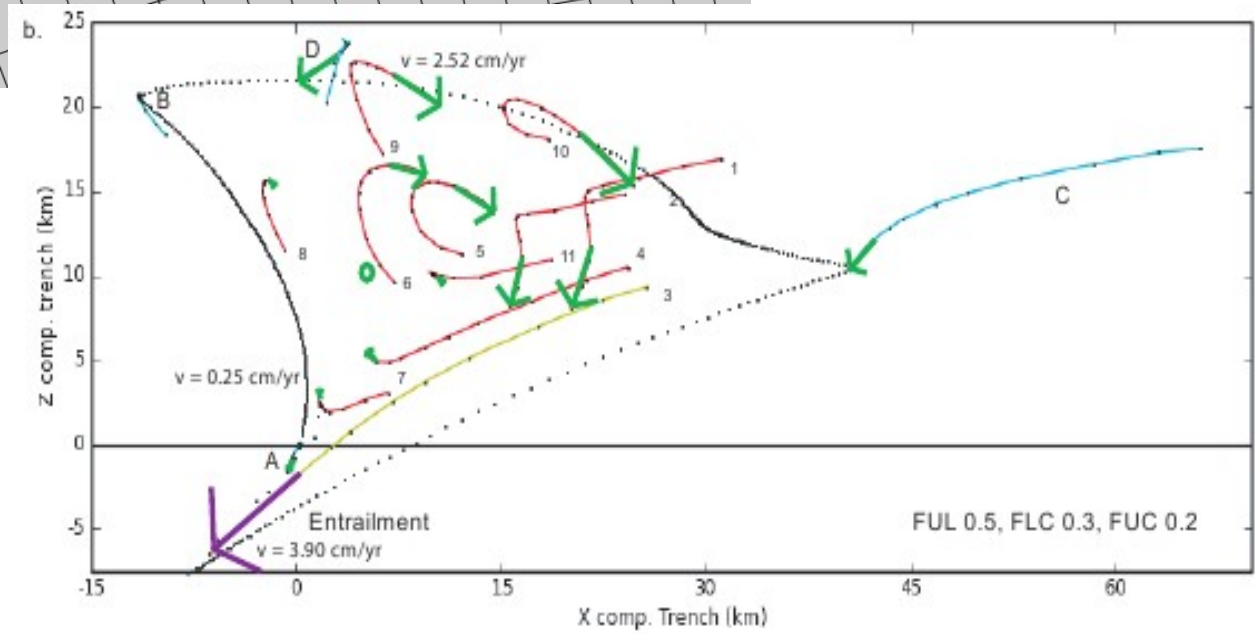
Positive example: Flow in domains with corners or separation



Nothing going on!

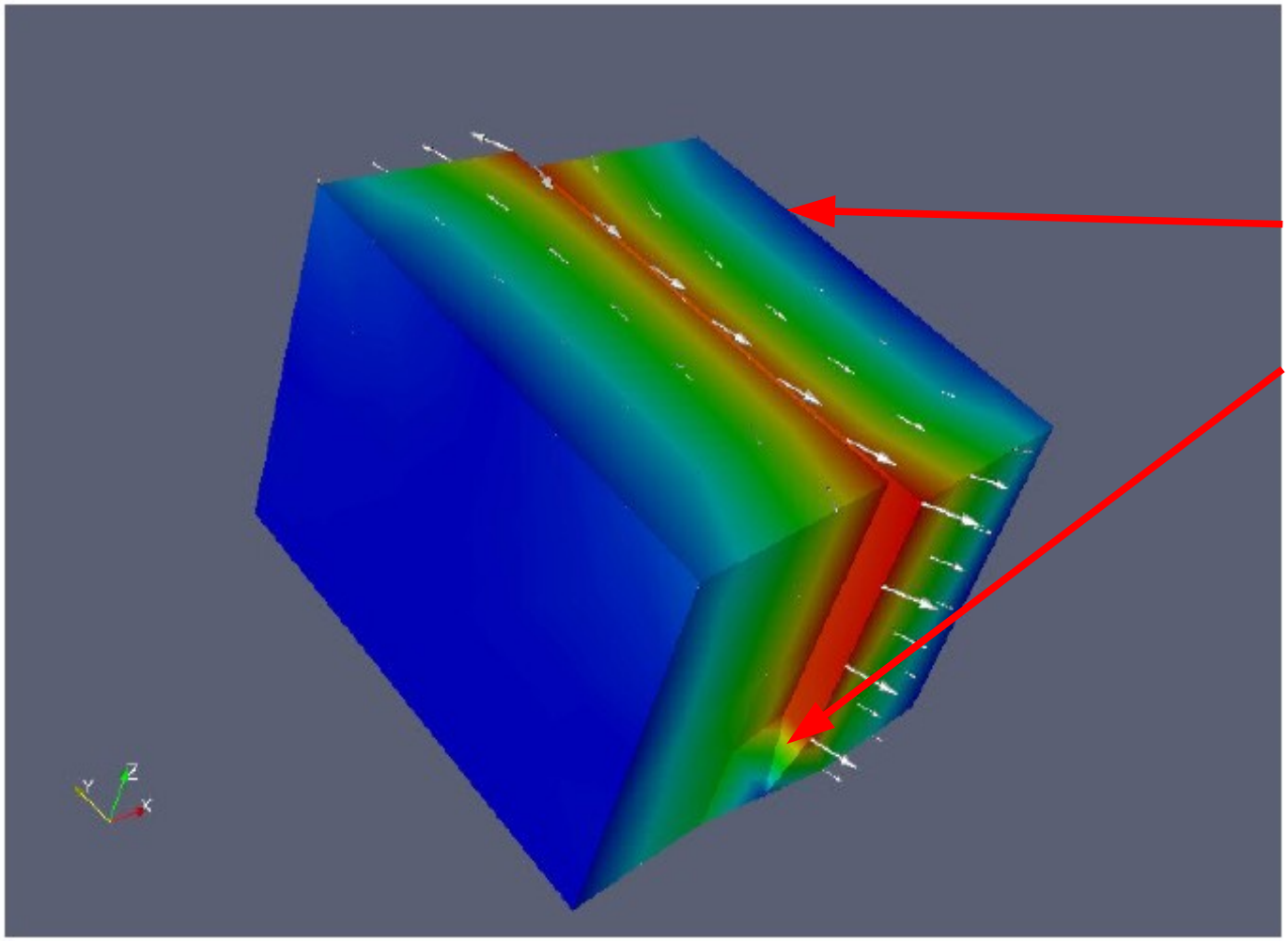
Singularity!

(Courtesy of
Ylona van Dinther,
ETH Zurich)



Adaptivity is good for geodynamics!

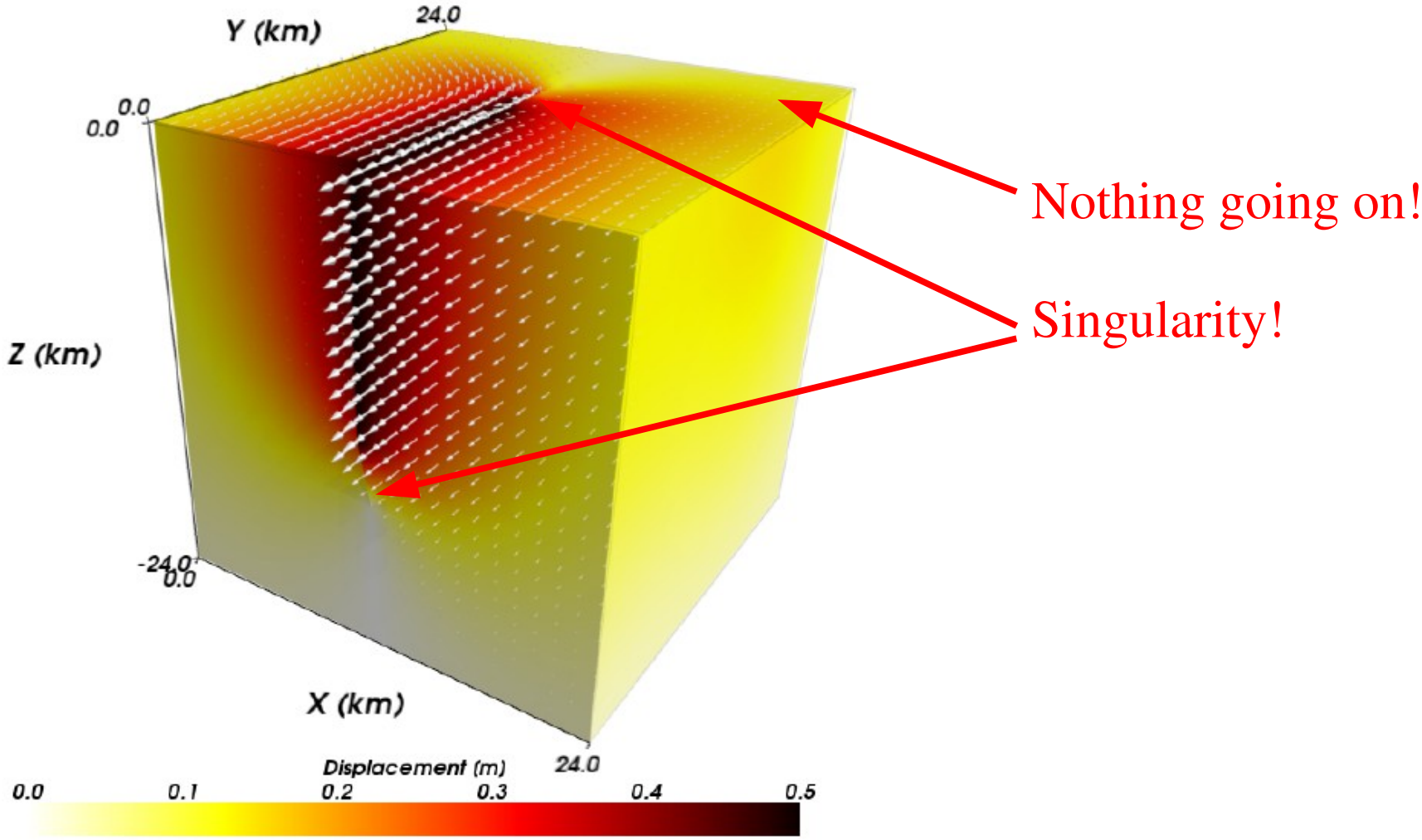
Positive example: Viscoelastic deformation



(From Aagard et al.: Pylith manual version 1.4.)

Adaptivity is good for geodynamics!

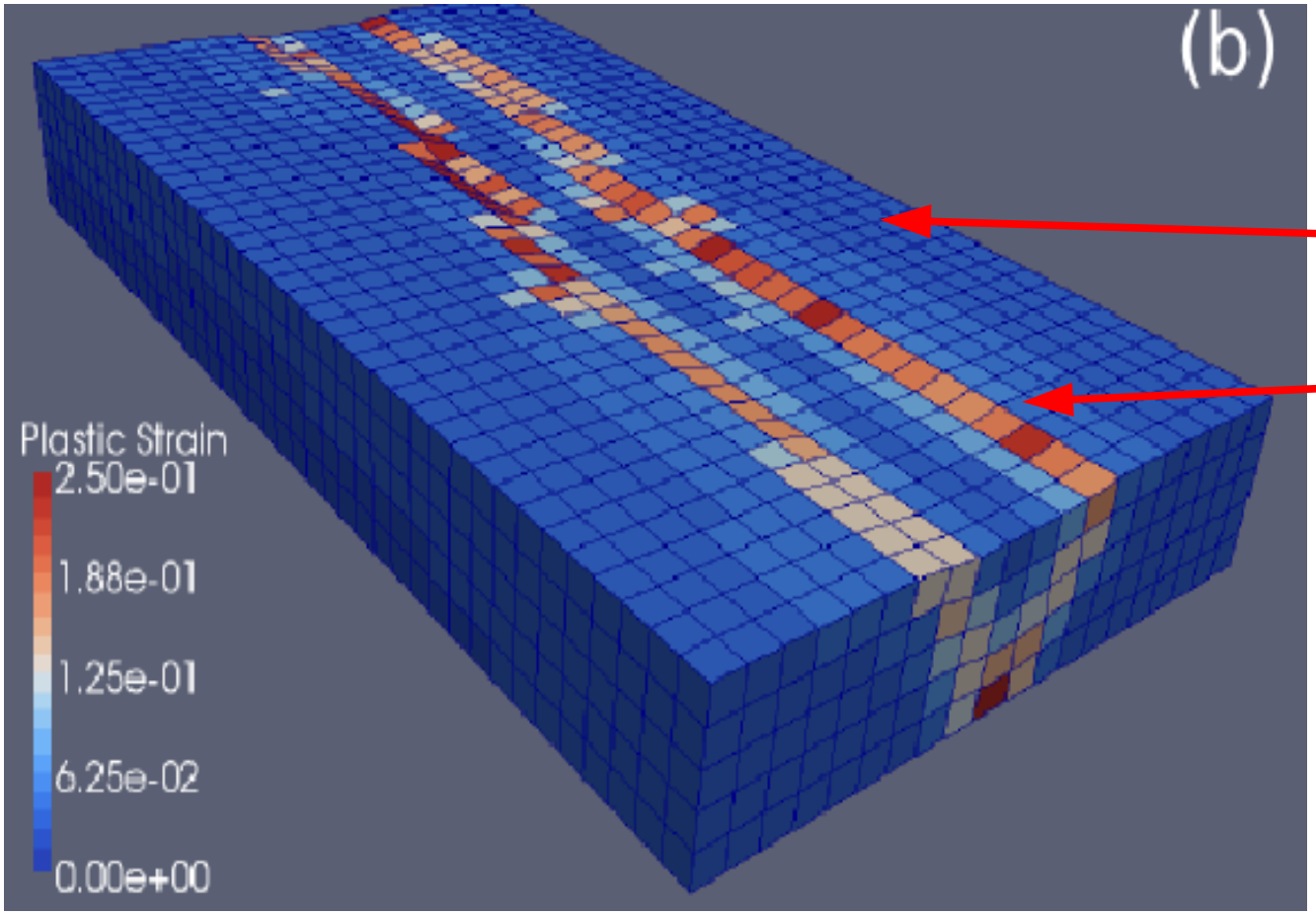
Positive example: Viscoelastic deformation



(From Aagard et al.: Pylith manual version 1.4.)

Adaptivity is good for geodynamics!

Positive example: Elastoplastic deformation



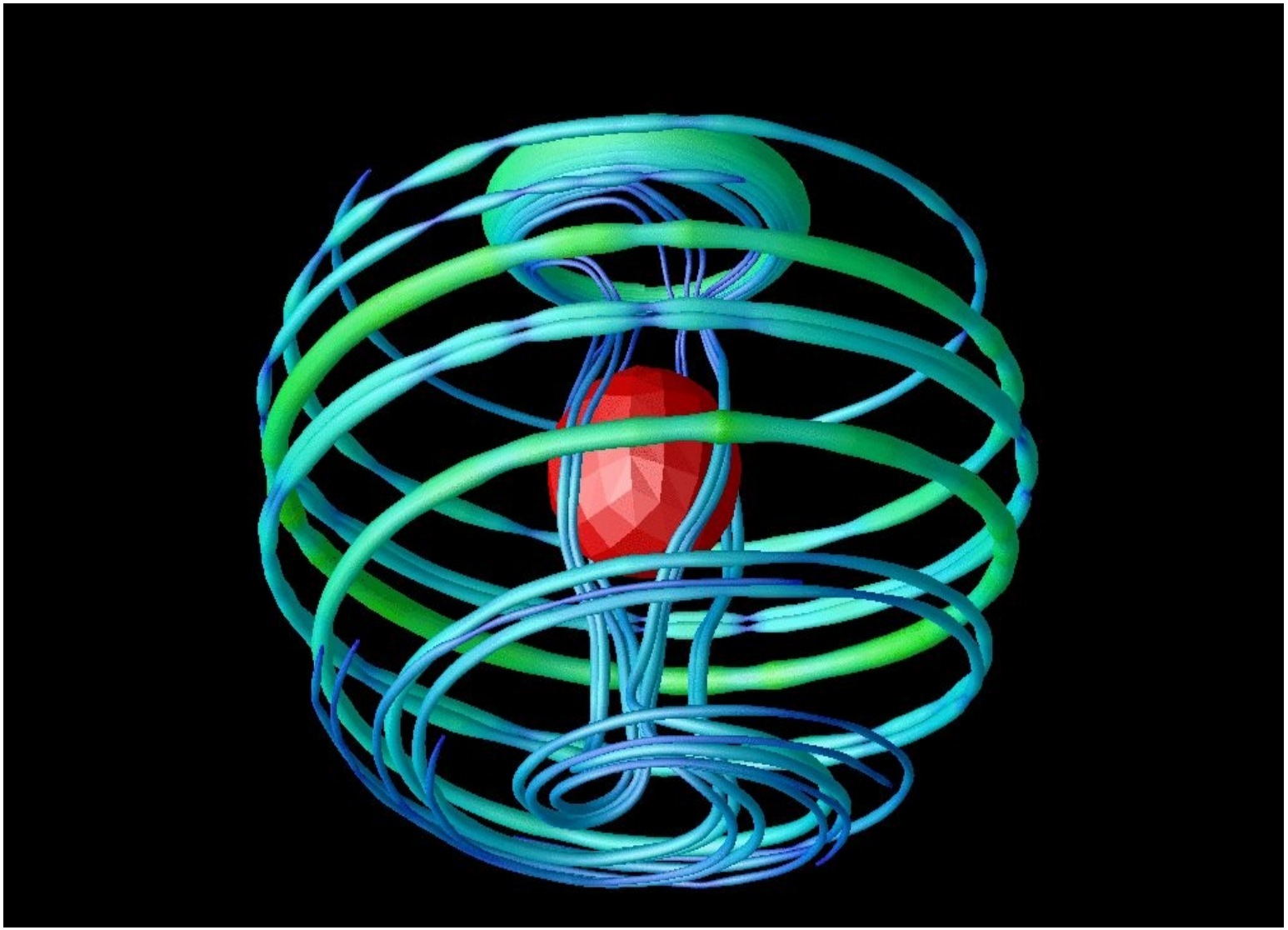
Nothing going on!

Singularity!

(From Choi et al.: SNAC manual version 1.1.)

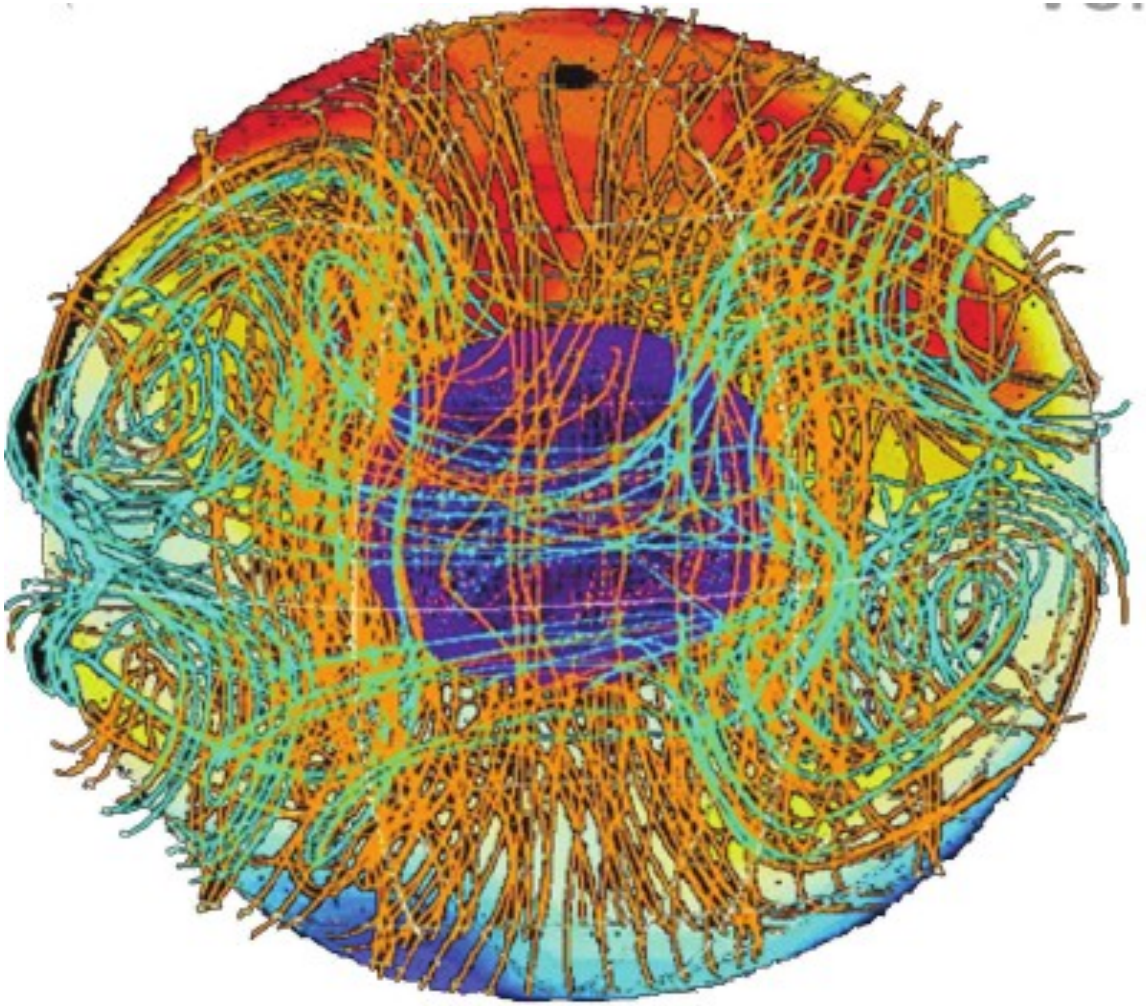
Adaptivity is also sometimes bad in geodynamics!

Counterexample: Geodynamo with its global turbulence and small-scale features



Adaptivity is also sometimes bad in geodynamics!

Counterexample: Geodynamo with its global turbulence and small-scale features



(From Olson et al.: MAG manual version 1.0.2.)

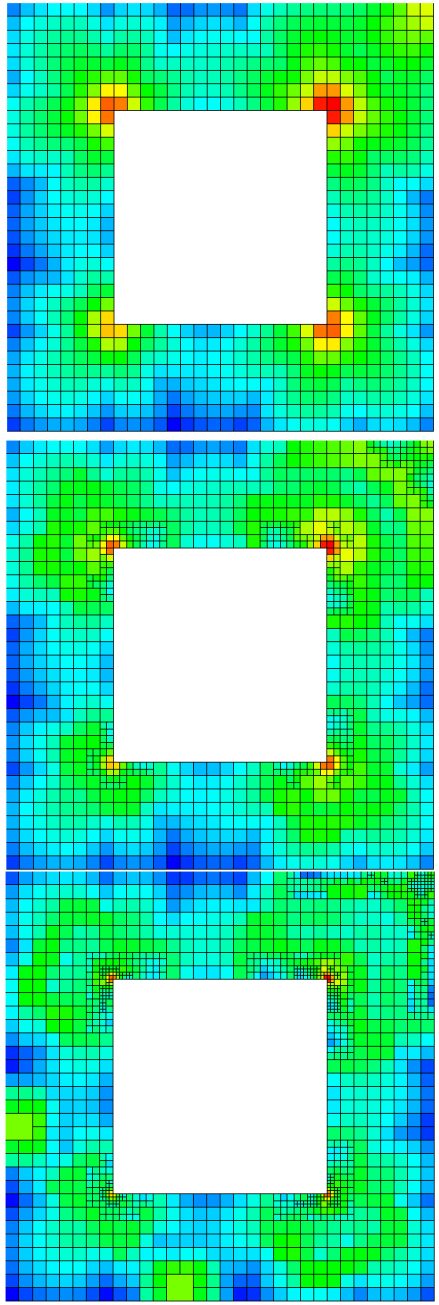
In theory, adaptivity is hard!

In order to use adaptivity you need:

- Software that can deal with meshes that change over time

- A criterion that can tell for every cell:
  - whether we want to refine it,
  - whether we want to coarsen it
  - whether we want to leave it alone

- This is typically done through *error estimators*:
  - calculate/estimate how big the error is for each cell
  - refine those cells with the largest errors
  - coarsen those with the smallest



In practice it's a lot simpler!

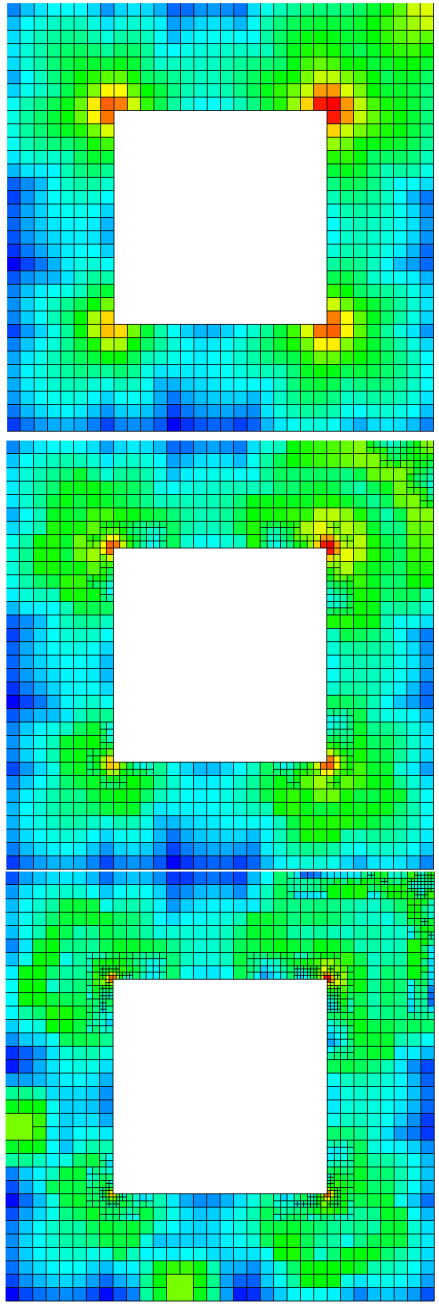
In order to use adaptivity you need:

- Software that can deal with meshes that change over time

  Use this!

- A criterion that can tell for every cell:
  - whether we want to refine it,
  - whether we want to coarsen it
  - whether we want to leave it alone

- This is typically done through *error estimators*.



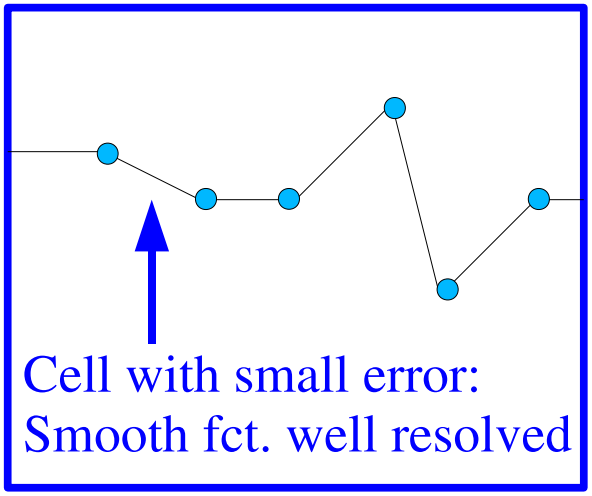
In practice it's a lot simpler!

In order to use adaptivity you need:

- Software that can deal with meshes that change over time

  Use deal.II!

- A criterion that can tell for every cell:
  - whether we want to refine it,
  - whether we want to coarsen it
  - whether we want to leave it alone

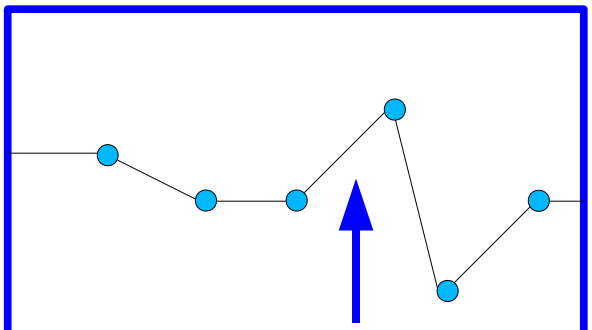


Cell with small error:
Smooth fct. well resolved

- This is typically done through *error estimators*.

  But in practice, this works just fine for almost all applications:

$$\eta_K = diam(K) \int_{\partial K} |\text{jump}(\nabla u_h)|^2 \, ds$$



Cell with big error:
Rough fct. badly resolved

In practice it's a lot simpler!

In order to use adaptivity you need:

- Software that can deal with meshes that change over time
  Use deal.II!

- An alternative is to re-generate the mesh every time using Cubit/ LaGrit/etc with a spatially variable mesh density.

  This loses a lot of the advantages of adaptivity, though.

I want to use adaptivity in my code!

What to do with existing codes:

- Converting existing codes is hard because changes in data structures and algorithms are so pervasive:
  - mesh data structures
  - finite elements/finite difference stencils
  - handling of hanging node constraints
  - linear solvers/preconditioners
  - top-level logic

- It may be simpler to write a new code from scratch

- Rewrite may be less painful than you think:
  - experience exists from previous codes (e.g.: what discretization, which solvers work, and which don't)
  - libraries exist that support adaptive finite elements

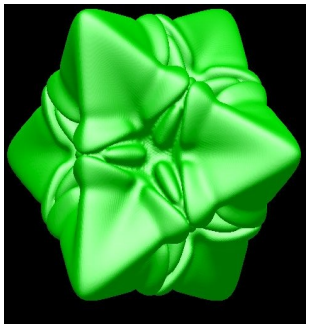
Available resources for new programs

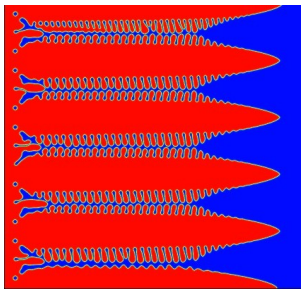
- PETSc (written in C):
  - everything for sequential and parallel linear algebra
  - direct and iterative solvers, algebraic multigrid
  - a bunch of other stuff
- Trilinos (written in C++):
  - everything for sequential and parallel linear algebra
  - direct and iterative solvers, algebraic multigrid
  - nonlinear solvers, automatic differentiation, optimization, ...
- deal.II (written in C++):
  - everything related to meshes, discretizations, etc
  - everything for sequential linear algebra
  - interfaces to PETSc, Trilinos, METIS, UMFPACK, ...
  - huge amount of documentation
  - tutorial programs of realistic complexity

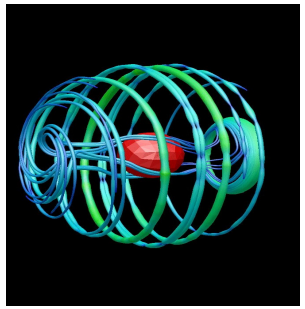
The deal.II library

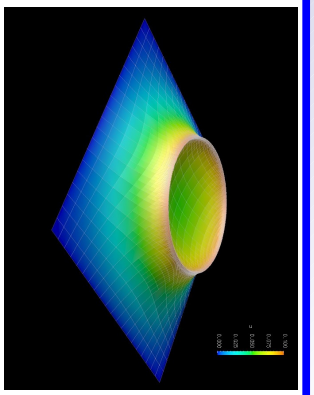
deal.II is a finite element software library:

- Provides support for adaptive meshes in 1d, 2d, and 3d through a unified interface
- Has standard refinement indicators built in
- Provides a variety of different finite element types (continuous, DG, mixed, Raviart-Thomas, Nedelec, ...)
- Low and high order elements
- Full support for multi-component problems
- Has its own sub-library for dense + sparse linear algebra
- But also comes with interfaces to PETSC, Trilinos, UMFPACK





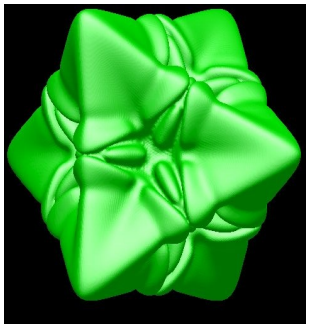
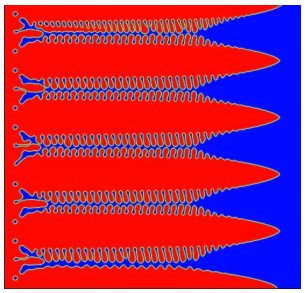
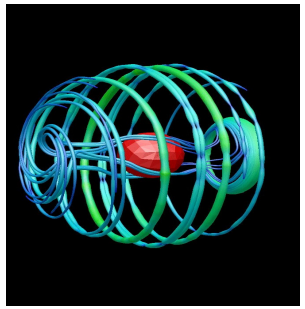
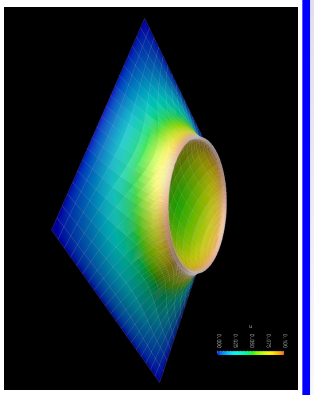




www.dealii.org

The deal.II library

- Supports SMP + cluster systems
- Interfaces to all major graphics programs
- Fairly widely distributed in the finite element/adaptivity community:
  - 200-300 downloads per month
  - 1000+ hits on homepage per month
  - 25-30 publications per year based on deal.II
- Supports a wide variety of applications in all sciences
- Presently over 450,000 lines of C++ code
- More than 5000 pages of documentation
- Open Source, active development
- Professional software methodology (testing, documentation, etc)

www.dealii.org

Examples of existing tutorial applications

Currently available and underway tutorial programs:

- There are currently 34 tutorial programs that explain the use of the library in detail, starting from very simple to quite complex applications
- For flow problems:
  . stationary Darcy solver (step-20)
  . two-phase time dependent Darcy flow (step-21)
  . Stokes solver (step-22)
  . Boussinesq solver (step-31)
  . parallel Boussinesq solver (step-32, almost finished)
  . Euler equations (step-33)
  . potential flow via boundary elements (step-34, almost finished)
  . projection scheme Navier-Stokes solver (step-35, coming)
  . 4 more programs from geophysical flow coming till 9/2009

Examples of existing tutorial applications

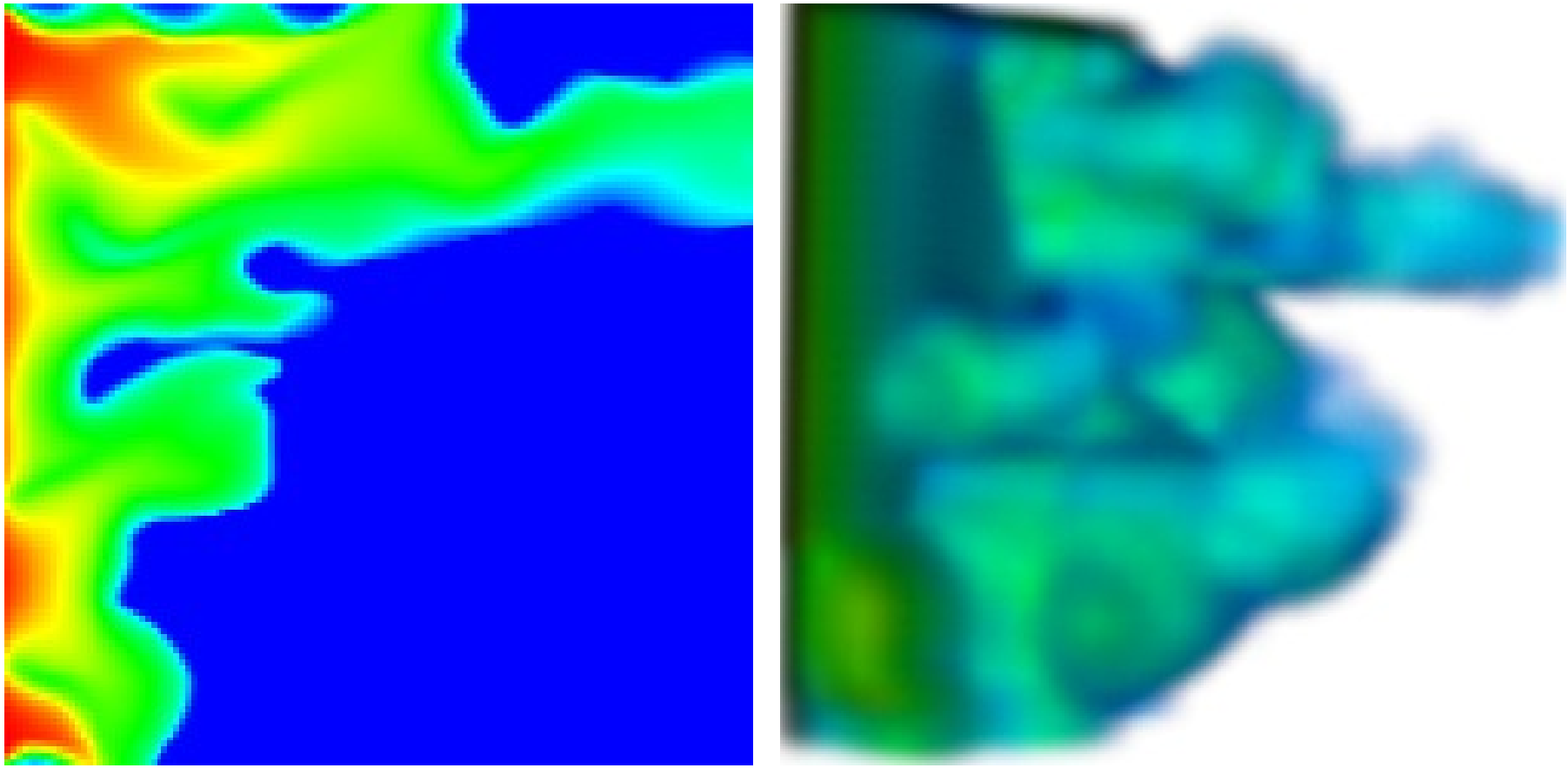
Currently available and underway tutorial programs:

- For elasticity:
  . 2d/3d linear elasticity (step-8, step-17)
  . 3d quasistatic (nonlinear) time dependent elasticity (step-18)
- Other tutorial programs solve wave equations, soliton equations, neutron transport...
- Tutorials are extensively documented pieces of code
- We encourage contribution of more codes and are quite willing to work with authors!

Examples of existing tutorial applications

deal.II's step-21 tutorial program:

A solver for time dependent two-phase flow (340 lines of code)

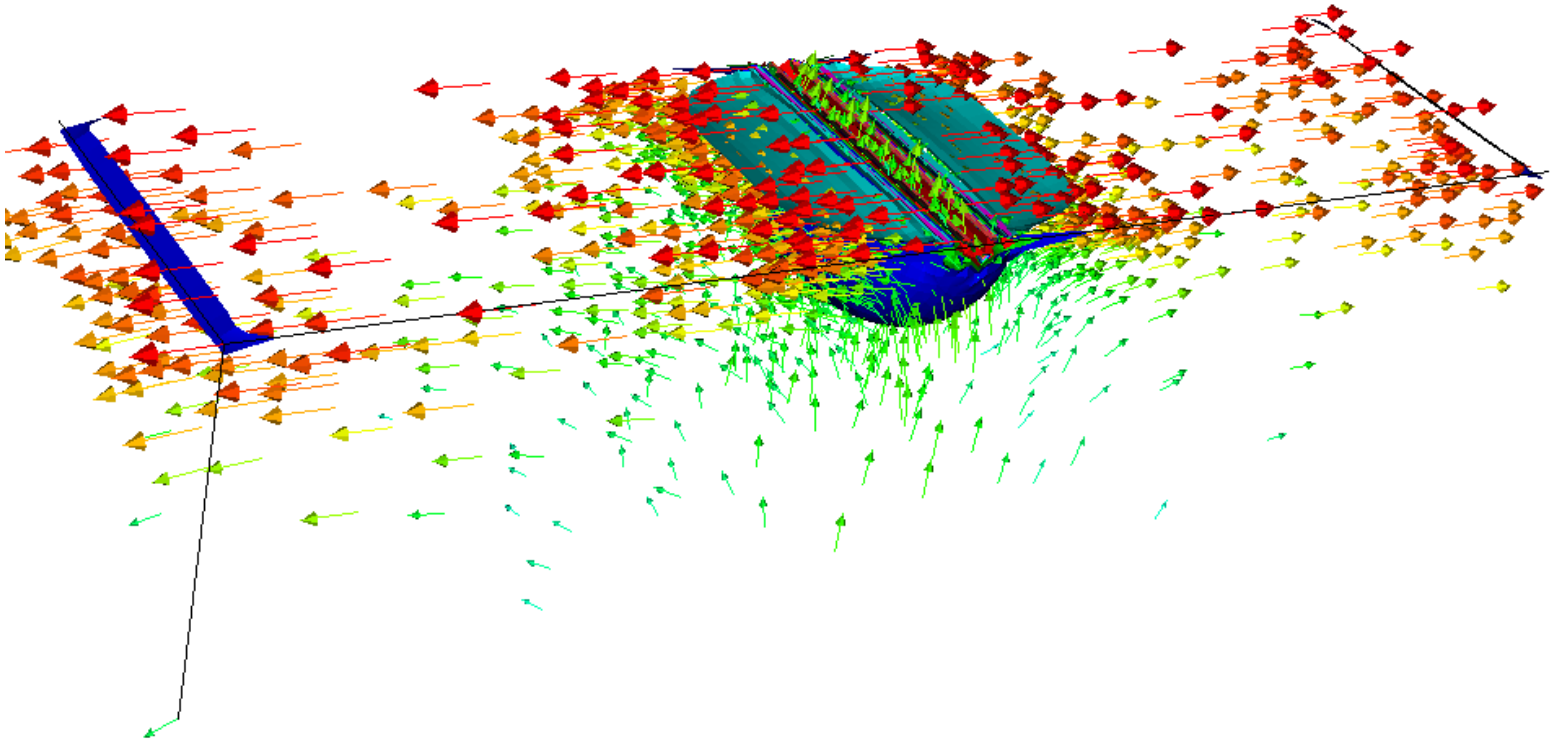


(Li, Bangerth)

Examples of existing tutorial applications

deal.II's step-22 tutorial program:

An adaptive solver for the Stokes equations (210 lines of code)

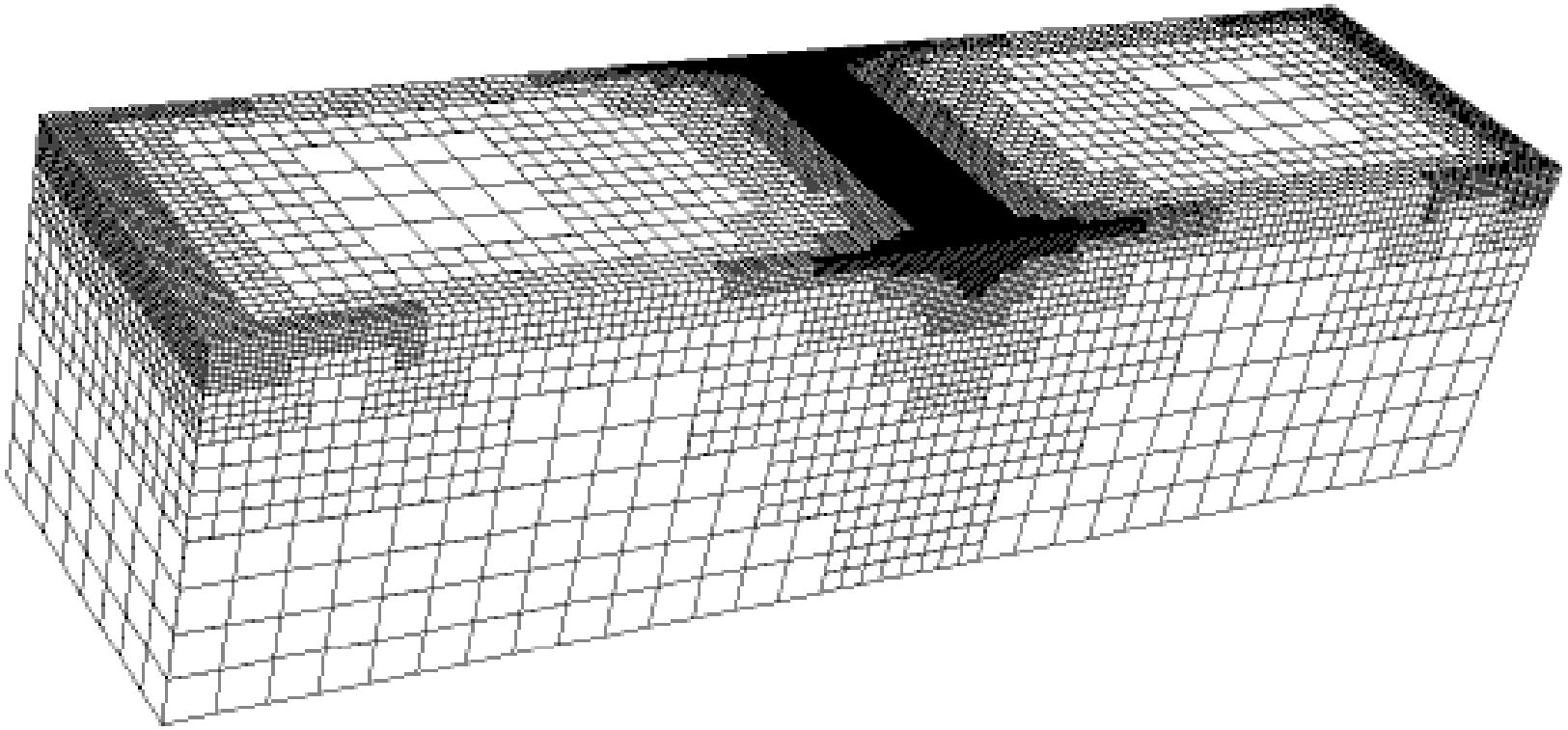


(Kronbichler, Bangerth)

Examples of existing tutorial applications

deal.II's step-22 tutorial program:

An adaptive solver for the Stokes equations (210 lines of code)

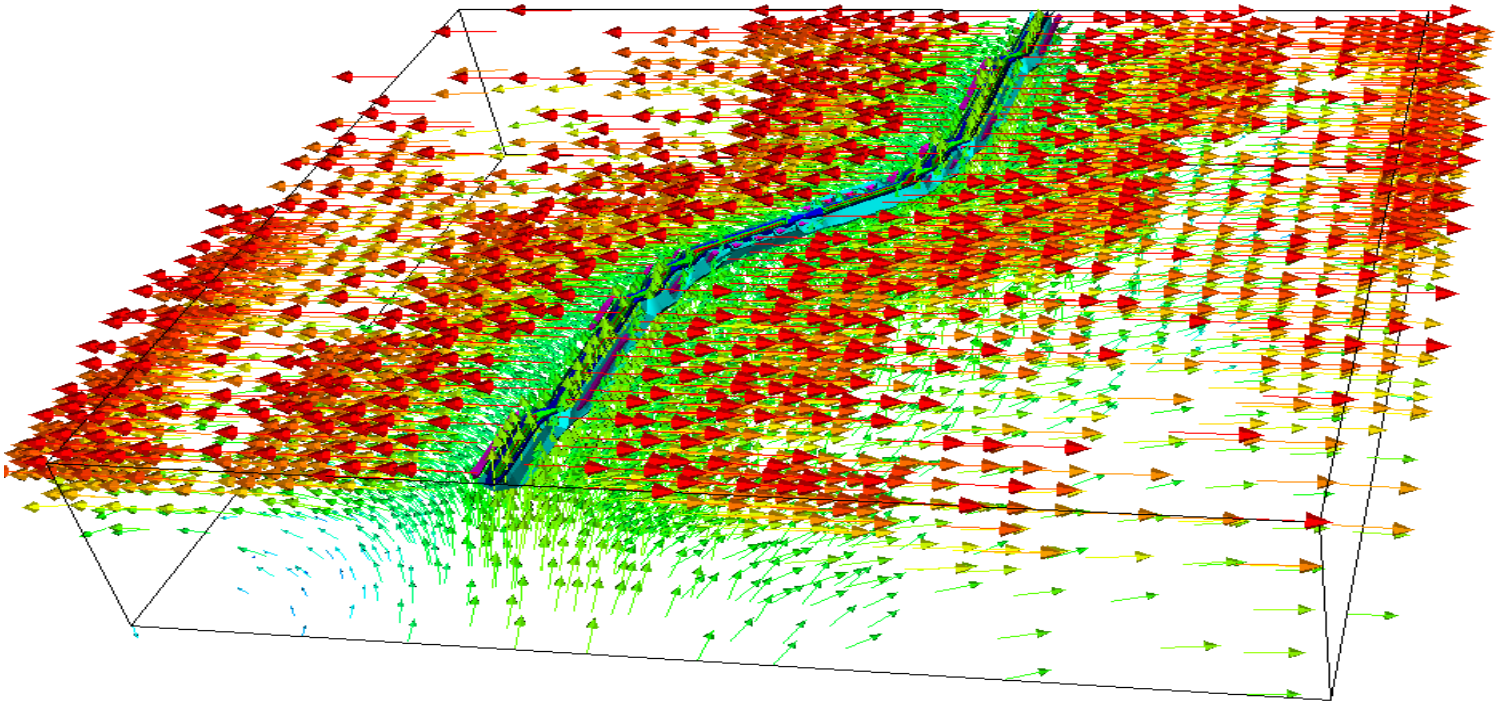


(Kronbichler, Bangerth)

Examples of existing tutorial applications

deal.II's step-22 tutorial program:

An adaptive solver for the Stokes equations (210 lines of code)



(Kronbichler, Bangerth)

Examples of existing tutorial applications

deal.II's step-22 tutorial program:

An adaptive solver for the Stokes equations (210 lines of code)

Code is an extensively documented testbed for numerical methods:

- Uses Q2/Q1 (Taylor-Hood) elements, but Q1/Q1+stabilization takes only changing ~20 lines of code
- Compares solving $\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}\begin{pmatrix} U \\ P \end{pmatrix}=\begin{pmatrix} F \\ 0 \end{pmatrix}$ using

  - the pressure Schur complement $S=B^T A^{-1} B$ with CG

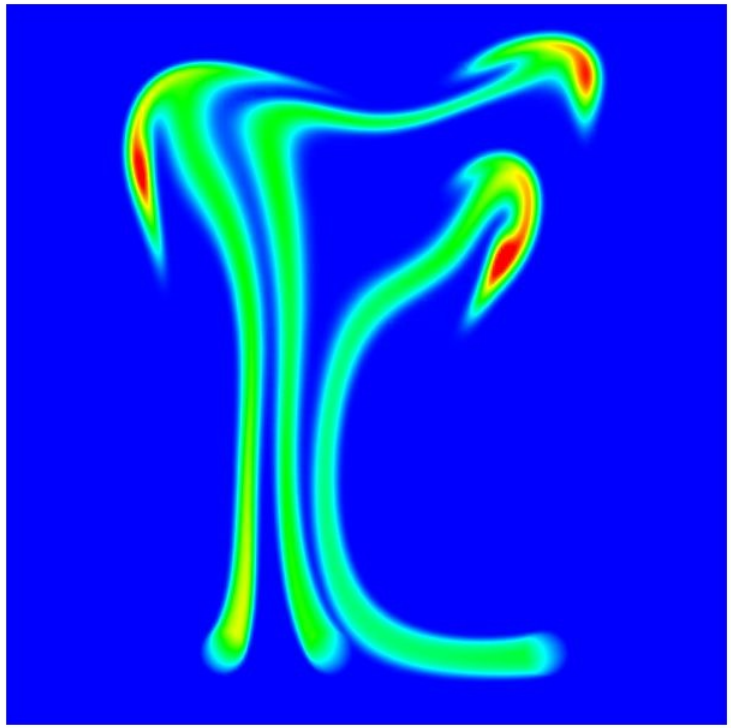
  - GMRES + block preconditioner $\begin{pmatrix} A^{-1} & 0 \\ B^T & S^{-1} \end{pmatrix}$ using Trilinos'

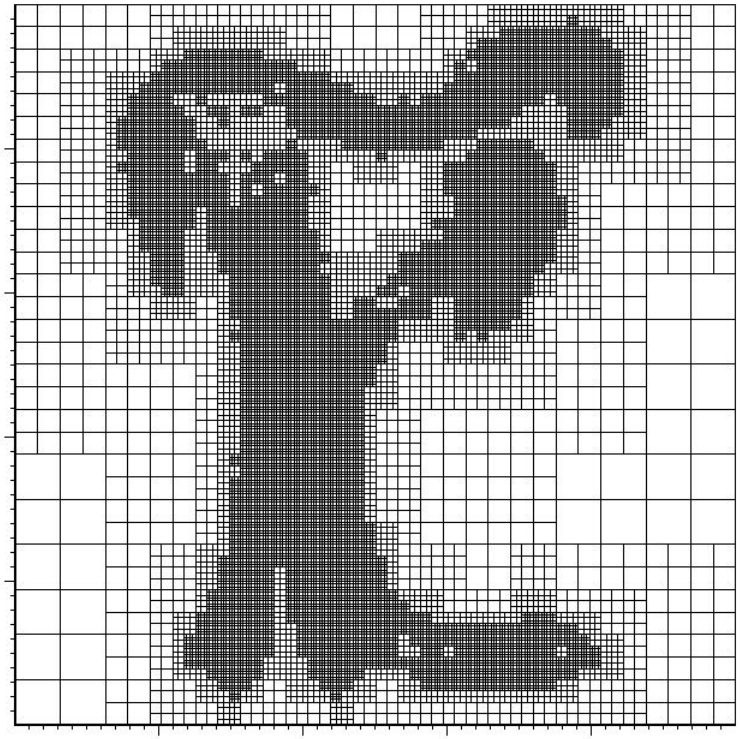
    algebraic multigrid implementation for the Laplace block

Examples of existing tutorial applications

deal.II's step-31 tutorial program:

An adaptive solver for the Boussinesq equations (530 lines of code)



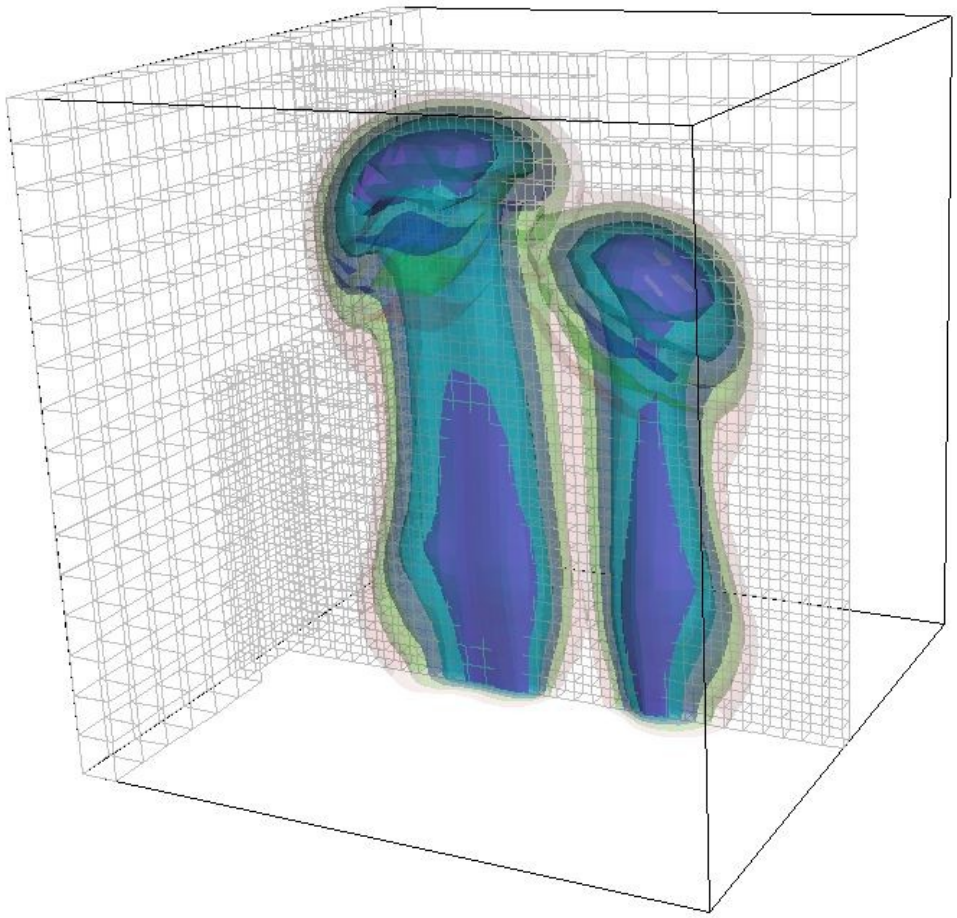


(Kronbichler, Bangerth)

Examples of existing tutorial applications

deal.II's step-31 tutorial program:

An adaptive solver for the Boussinesq equations (530 lines of code)



(Kronbichler, Bangerth)

Examples of existing tutorial applications

deal.II's step-31 tutorial program:
An adaptive solver for the Boussinesq equations (530 lines of code)

Testbed for numerical methods for thermal convection:

- GMRES with block triangular preconditioners, Trilinos' ML as inner preconditioner on the form

  $(\nabla u, \nabla v)$ + constraints from boundary conditions

- IMPES-like scheme to decouple Stokes and advection
- Adaptive time step BDF-2 for time discretization
- Nonlinear artificial viscosity to stabilize transport:

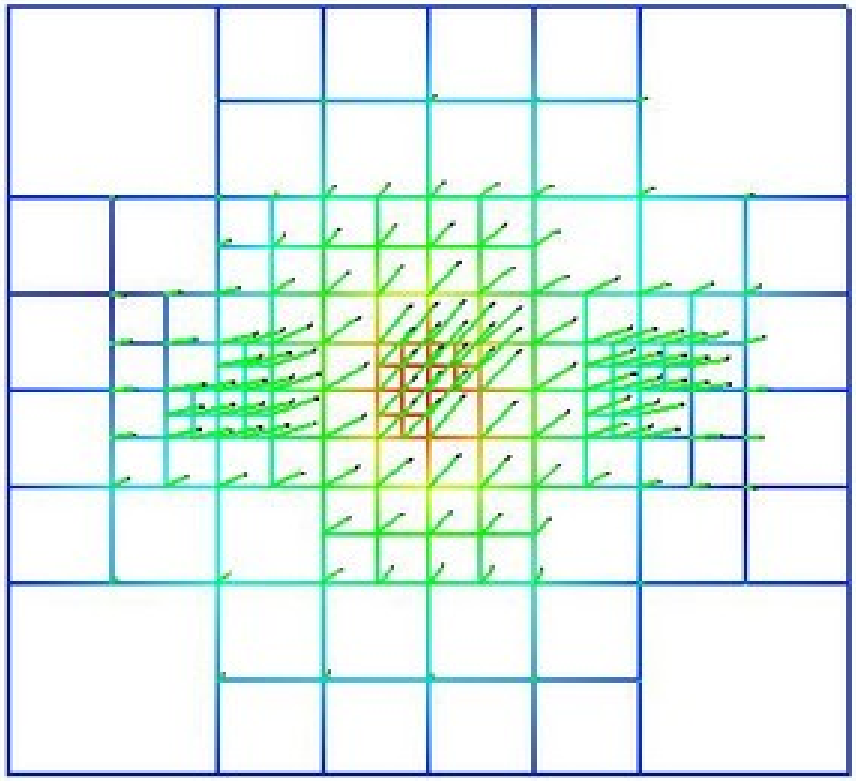
$$\frac{\partial T}{\partial t} + u \cdot \nabla T - (\kappa + \nu) \Delta T = q$$

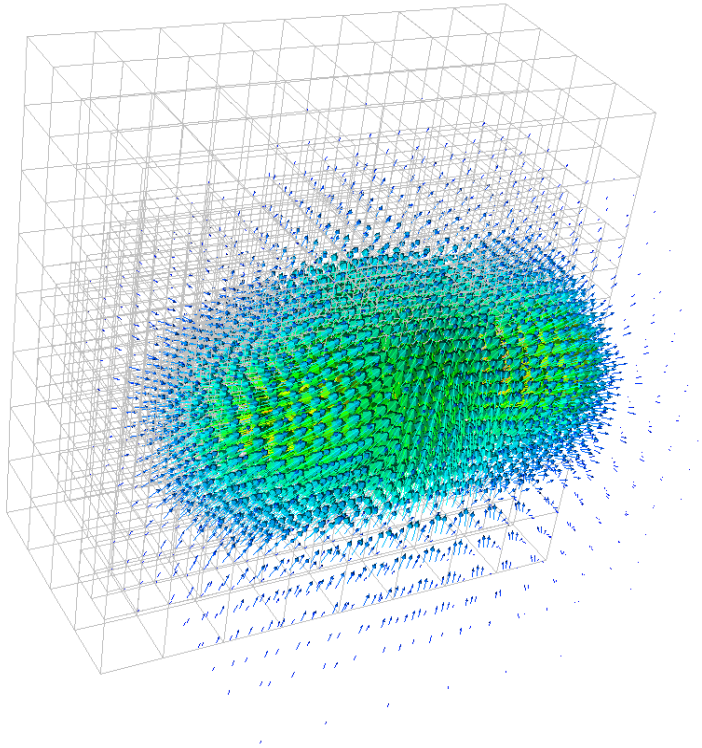
$$\nu_K = C_1 h_K \min\left(C_2, \|R_K\|\right)$$

Examples of existing tutorial applications

deal.II's step-8/step-17 tutorial program:

Adaptive sequential and parallel solvers for the elasticity equations (150 lines of code)

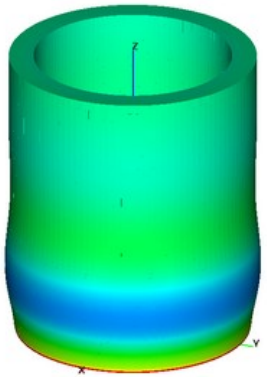
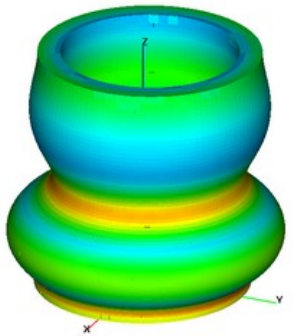
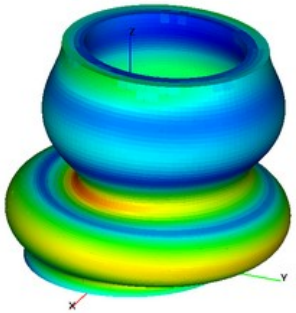
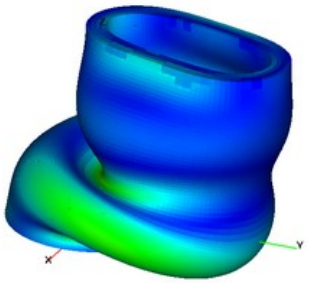




Examples of existing tutorial applications

deal.II's step-18 tutorial program:

A parallel solver for quasi-static time dependent elasticity
(350 lines of code)

A concrete example: the essence of step-4

```
Triangulation<2>   triangulation;
FE_Q<2>            fe(1);
DoFHandler<2>      dof_handler(triangulation);

SparsityPattern      sparsity_pattern;
SparseMatrix<double> system_matrix;

Vector<double>       solution;
Vector<double>       system_rhs;

int main () {
  GridGenerator::hyper_cube (triangulation, -1, 1);
  triangulation.refine_global (4);

  dof_handler.distribute_dofs (fe);

  sparsity_pattern.reinit (dof_handler.n_dofs(), dof_handler.n_dofs(),
                           dof_handler.max_couplings_between_dofs());
  DoFTools::make_sparsity_pattern (dof_handler, sparsity_pattern);
  sparsity_pattern.compress();

  system_matrix.reinit (sparsity_pattern);
```

A concrete example: the essence of step-4

```
QGauss<2>  quadrature_formula(2);
FEValues<2> fe_values (fe, quadrature_formula,
                       update_values   | update_gradients |  update_JxW_values);

FullMatrix<double>   cell_matrix (fe.dofs_per_cell, fe.dofs_per_cell);
Vector<double>       cell_rhs (fe.dofs_per_cell);

std::vector<unsigned int> local_dof_indices (fe.dofs_per_cell);
```

```
DoFHandler<2>::active_cell_iterator cell = dof_handler.begin_active(),
                                    endc = dof_handler.end();
for (; cell!=endc; ++cell)    {
   fe_values.reinit (cell);
   cell_matrix = 0;
   cell_rhs = 0;
```

```
   for (unsigned int q_point=0; q_point<n_q_points; ++q_point)
     for (unsigned int i=0; i<dofs_per_cell; ++i)  {
        for (unsigned int j=0; j<dofs_per_cell; ++j)
          cell_matrix(i,j) += (fe_values.shape_grad (i, q_point) *
                               fe_values.shape_grad (j, q_point) *
                               fe_values.JxW (q_point));
```

A concrete example: the essence of step-4

```
            cell_rhs(i) += (fe_values.shape_value (i, q_point) *
                            1.0 *
                            fe_values.JxW (q_point));
        }

      cell->distribute_local_to_global (cell_matrix, system_matrix);
      cell->distribute_local_to_global (cell_rhs, system_rhs);
    }
```

```
std::map<unsigned int,double> boundary_values;
VectorTools::interpolate_boundary_values (dof_handler,
                                          0,
                                          ZeroFunction<2>(),
                                          boundary_values);
MatrixTools::apply_boundary_values (boundary_values,
                                    system_matrix,
                                    solution,
                                    system_rhs);
```

A concrete example: the essence of step-4

```
  SolverControl           solver_control (1000, 1e-12);
  SolverCG<>              cg (solver_control);
  cg.solve (system_matrix, solution, system_rhs,
          PreconditionIdentity());
```

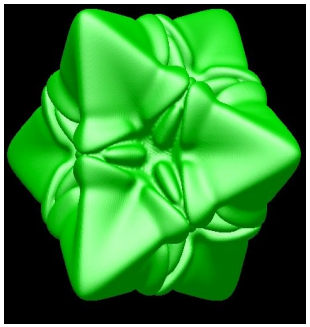
```
  DataOut<2> data_out;
  data_out.attach_dof_handler (dof_handler);
  data_out.add_data_vector (solution, "solution");
  data_out.build_patches ();
  std::ofstream output ("solution.vtk");
  data_out.write_vtk (output);
}
```

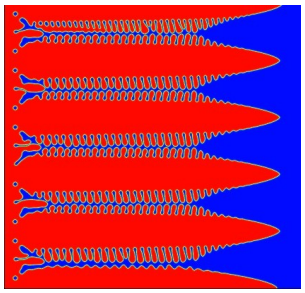
New stuff in step-6:

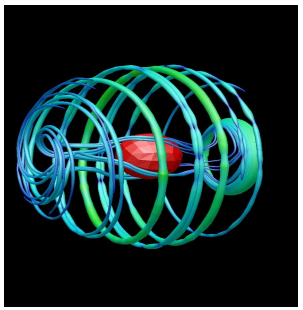
Adding hanging nodes to this program takes 6 more lines, refining the mesh adaptively around 10.

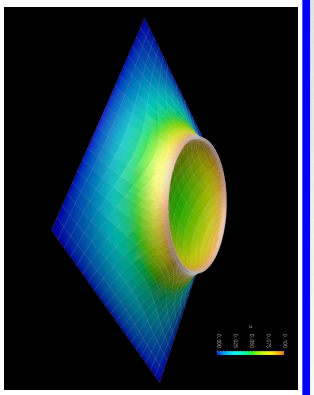
I want to write a deal.II application!

What does it take to write an application based on deal.II?









I want to write a deal.II application!

In the simplest case, writing new applications is trivial:

- If the new method is a variant of one that is already used in existing tutorial programs, then the latter can easily be modified.
- **Advantages:**
  . Quick startup: A student can start from a working, well-tested program

  . Graphical proof of correctness is available from the start and provides extra motivation

  . Possible errors can only be in the most recently introduced new codes, not “everywhere”
- Almost all existing applications are based on one tutorial program or another.
- Tutorial programs are relatively small and very well documented

I want to write a deal.II application!

In more complex cases:

- Applications can be written from scratch
- deal.II is organized as a library of components that can be combined at will, without an imposition of top-level program structures
- deal.II's 450,000 lines of code provide high-level tools, data structures, and algorithms for most any purpose you can think of
- New codes basically only have to implement the things that are specific to this application

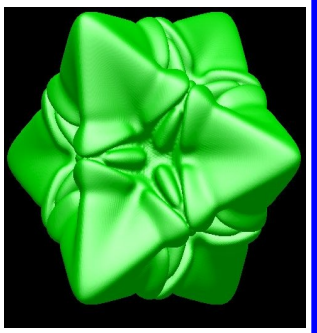
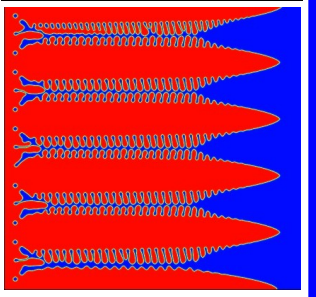
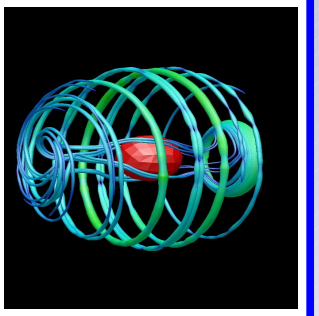
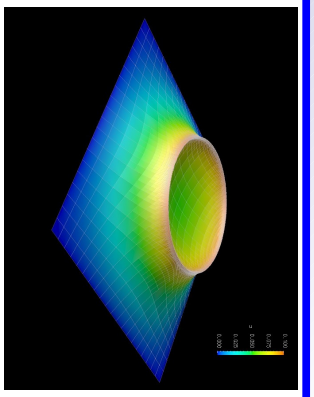
I want to write a deal.II application!

Q: What does it take to write new applications?

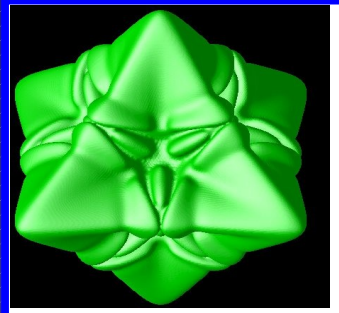
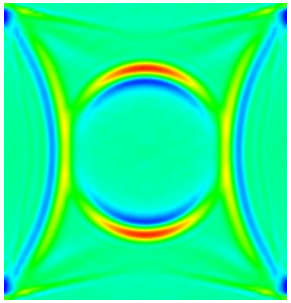
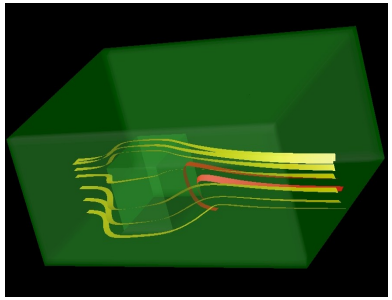
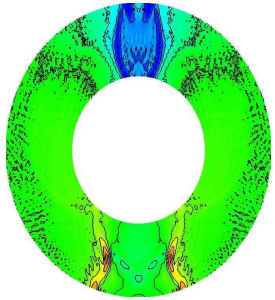
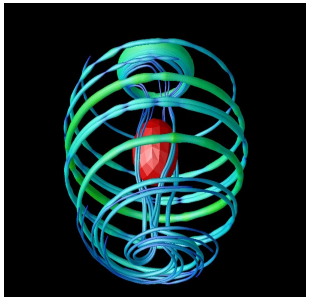
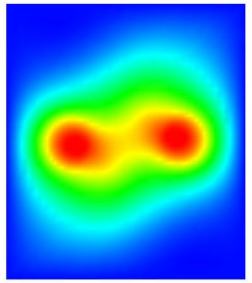
A: Not a whole lot, actually!

And if you try:

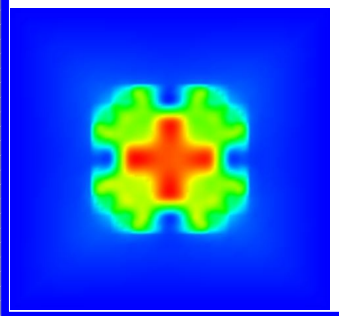
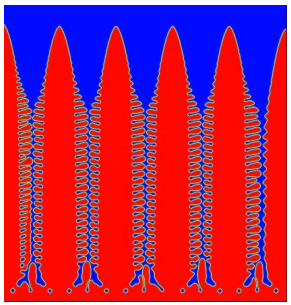
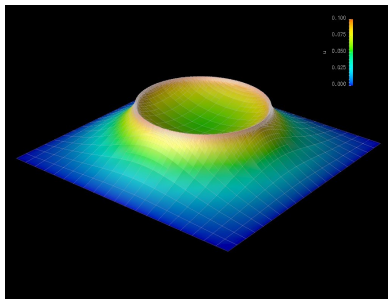
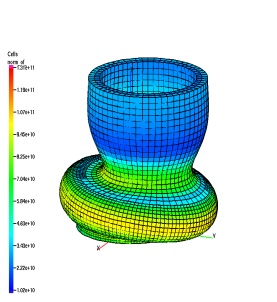
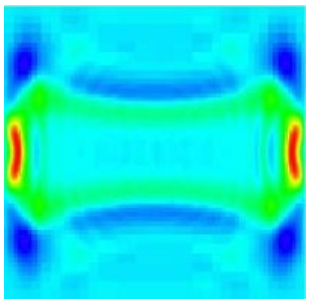
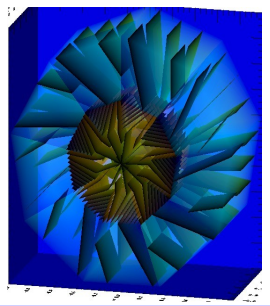
. an active developer community will be at your side to assist in case of questions
. we will be happy to help you get the code into shape for inclusion into deal.II (and give proper credit – this could count as a publication) so that others can use it as well!
. we are working on other venues for citable publishing of codes based on deal.II

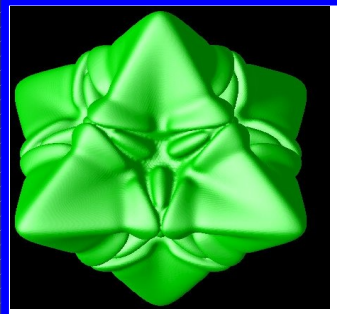
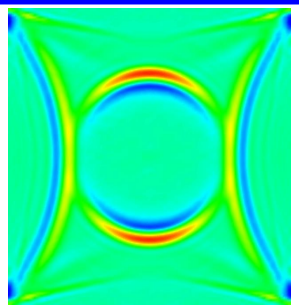
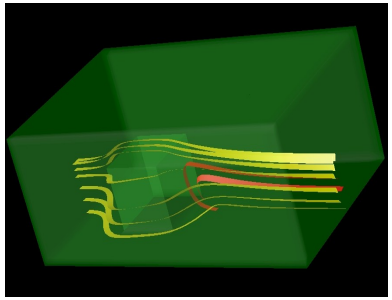
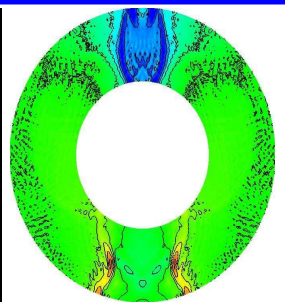
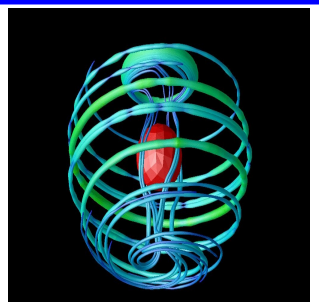
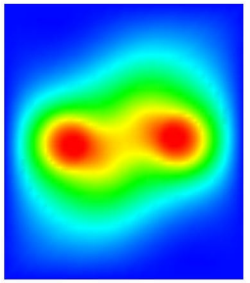
Conclusions

- Programs for nontrivial applications exist and are freely available
- Students and faculty can use these as starting points for their own research
- The threshold for new applications is not very high
- We are quite willing to help with questions & answers!

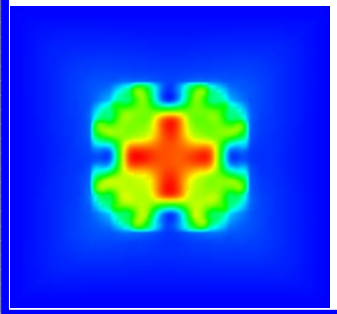
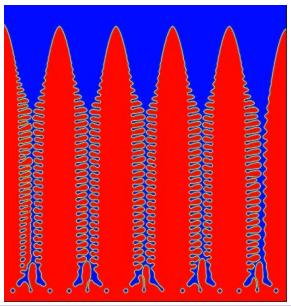
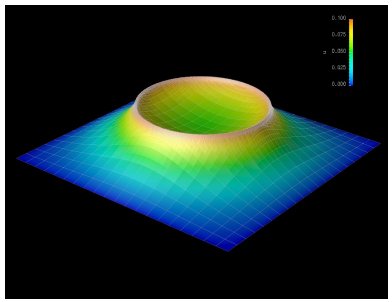
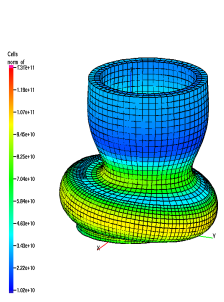
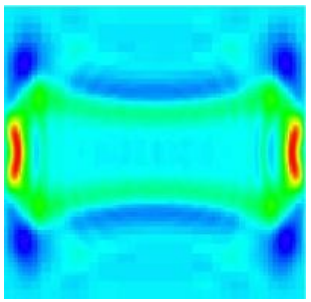
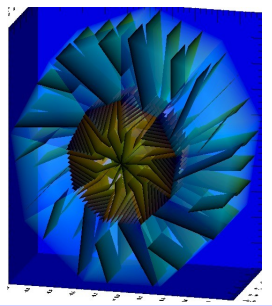







The deal.II library

Visit the deal.II library:

http://www.dealii.org

www.dealii.org