Adaptive Mesh Refinement

Wolfgang Bangerth

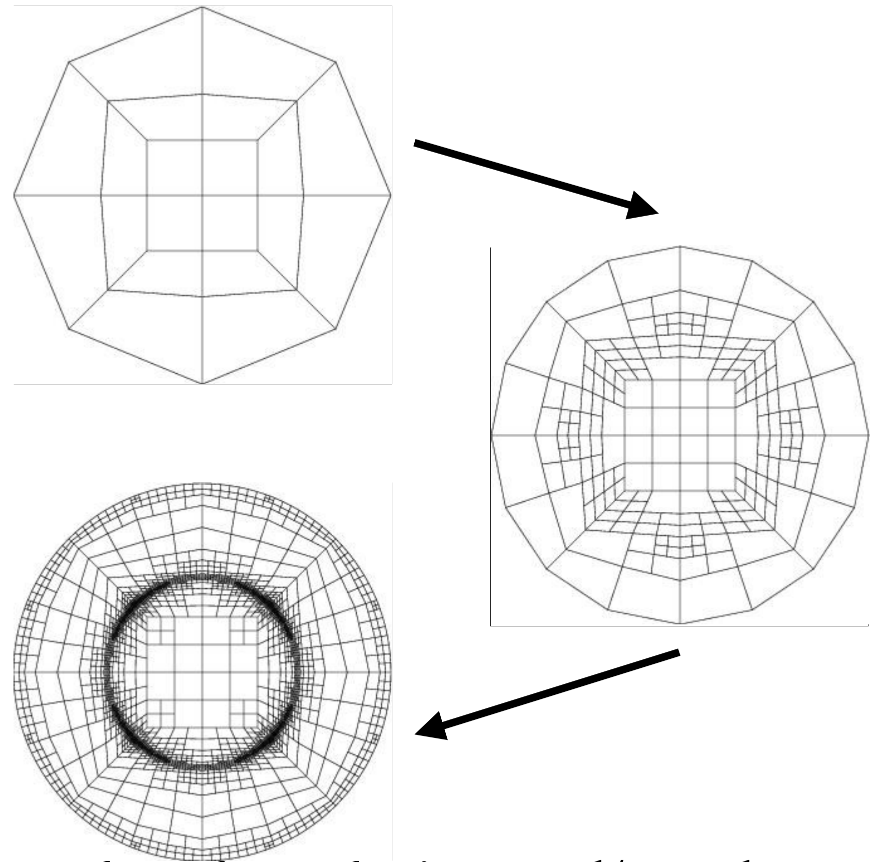
Department of Mathematics
Texas A&M University

Institute for Geophysics
University of Texas at Austin

The Adaptive Paradigm

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?

- **Parallelization and load balancing issues for adaptive meshes**

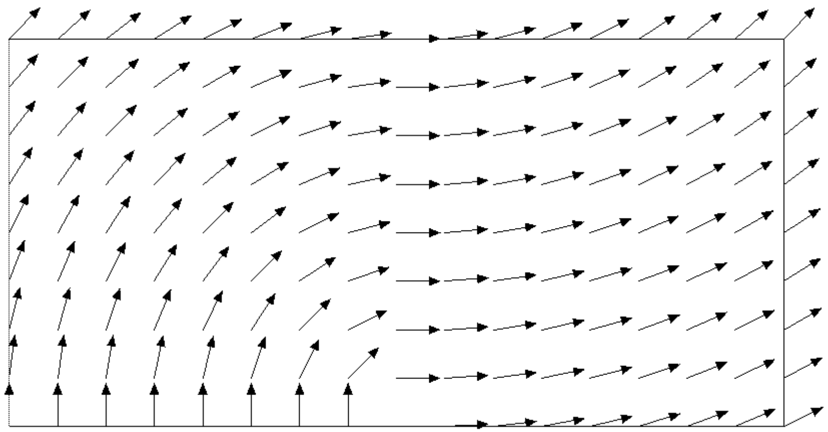
When do we gain by using Adaptivity?

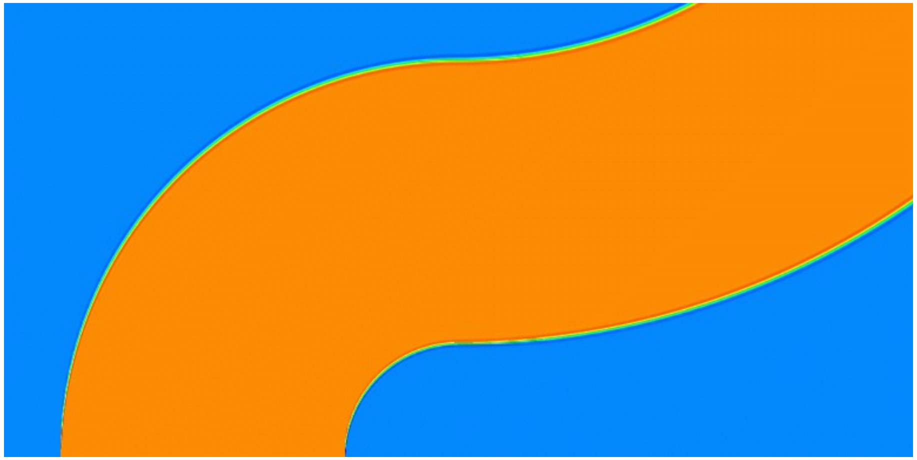
- Adaptive meshes can be beneficial because they promise to reach the same accuracy with less cells
- Can focus degrees of freedom where the solution shows significant variation
- Avoid generation of fine meshes (avoids scalability, bad shapes) by generating them as necessary from coarse meshes

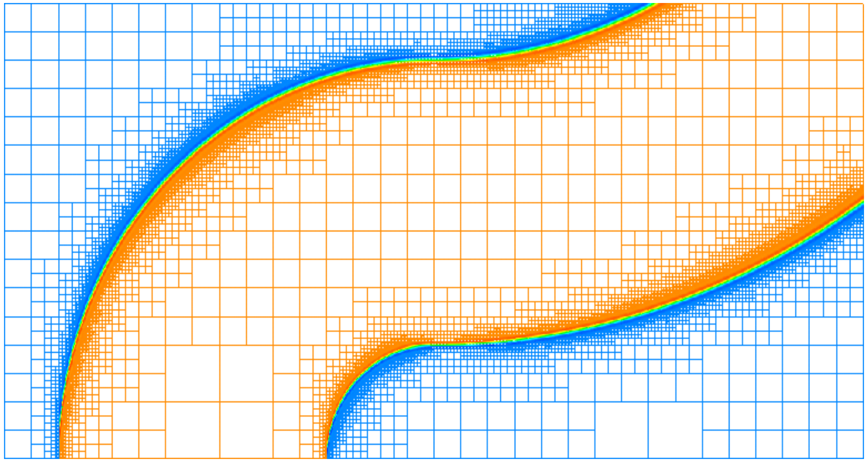
Adaptivity is good if and only if "the action is localized"

When do we gain by using Adaptivity?

Positive example: Advective transport in a given wind field (Geophysical analogy: Transport of water and carbon by subducting slabs)



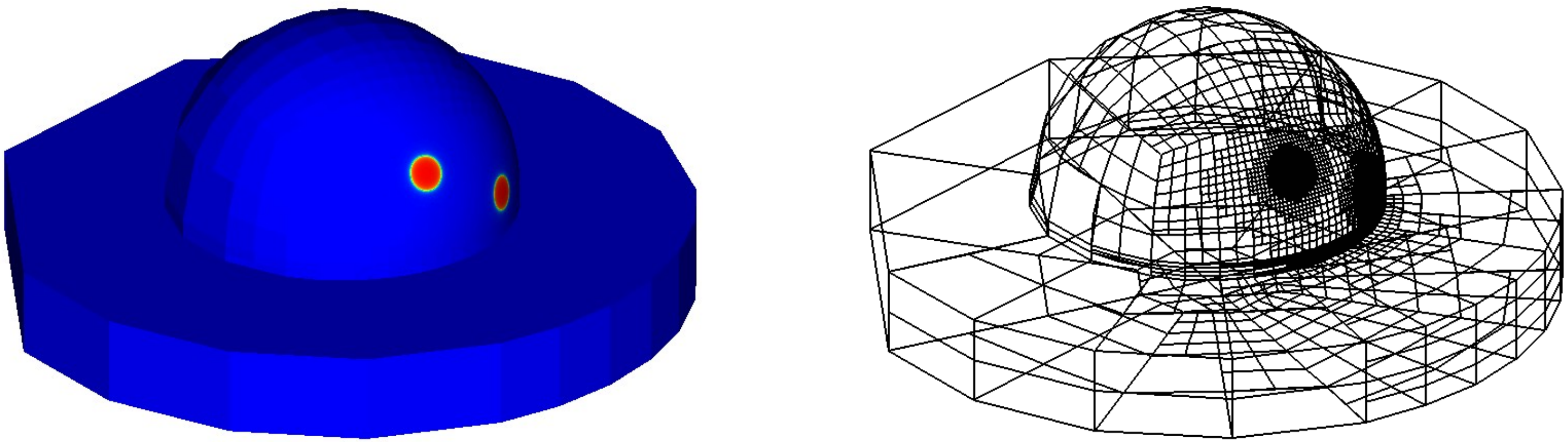




Savings from adaptive meshes are apparent. Note also the lack of numerical dispersion even without aligned meshes!

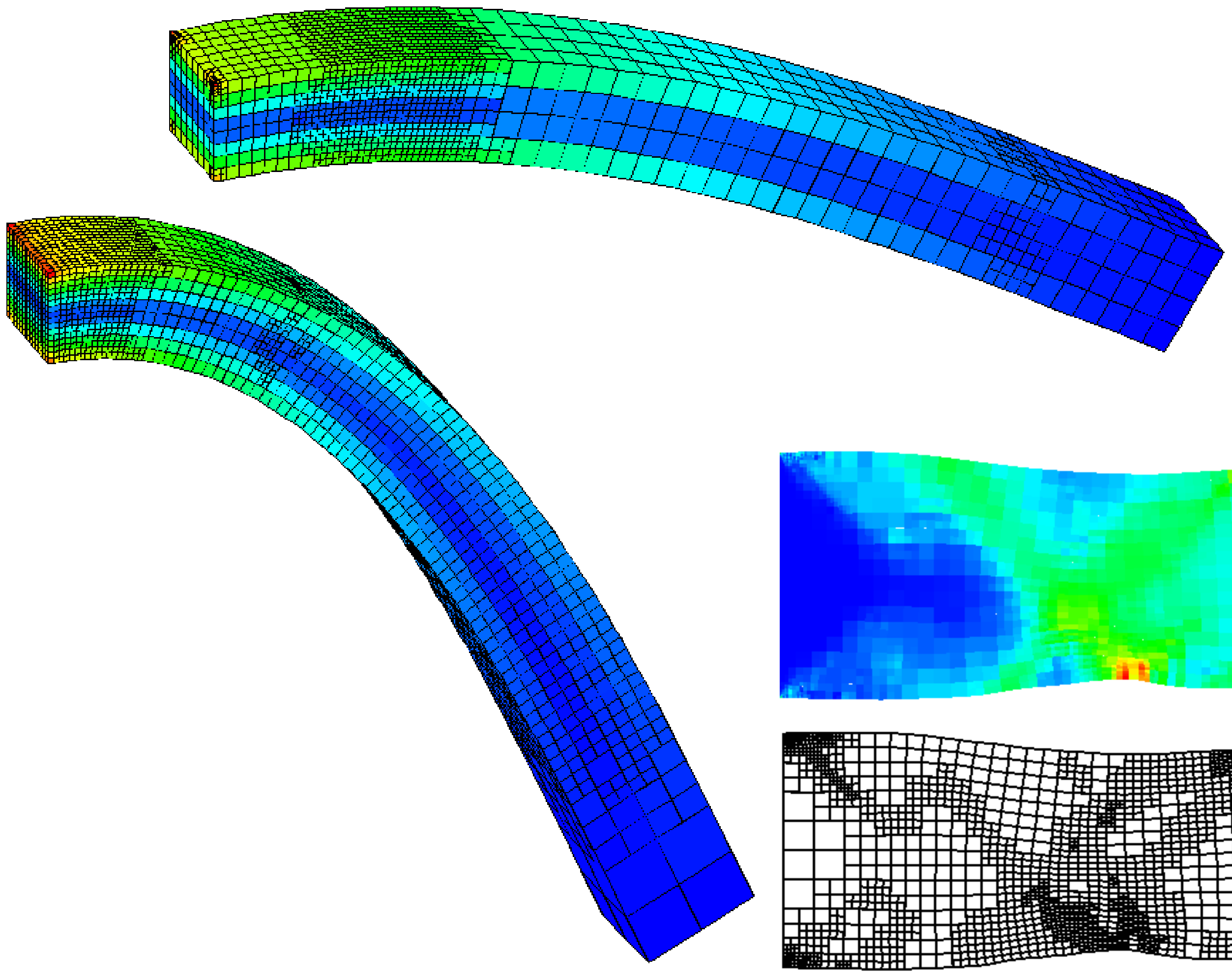
When do we gain by using Adaptivity?

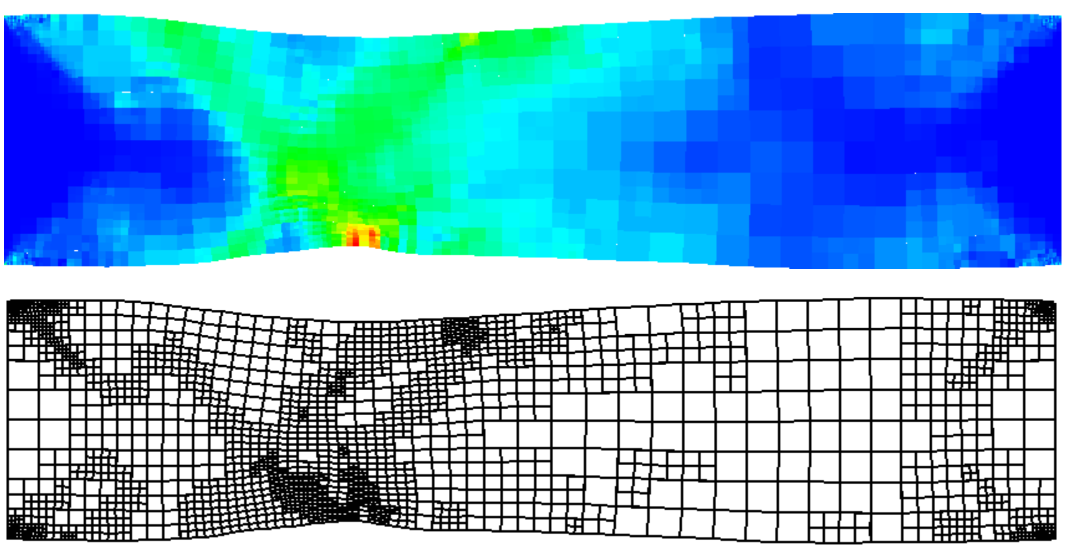
Positive example: Diffusion with localizes sources
(Geophysical analogy: Heat conduction around hot plumes?)



When do we gain by using Adaptivity?

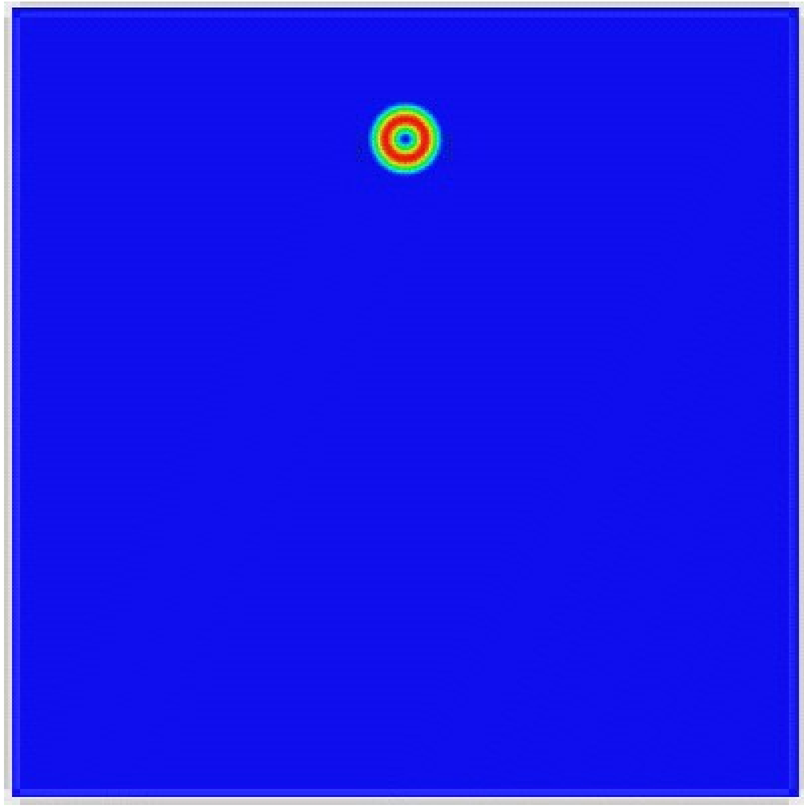
Positive example: Elastoplastic deformation
(Geophysical analogy: Long-term continental deformation, subduction bending of continental plates)

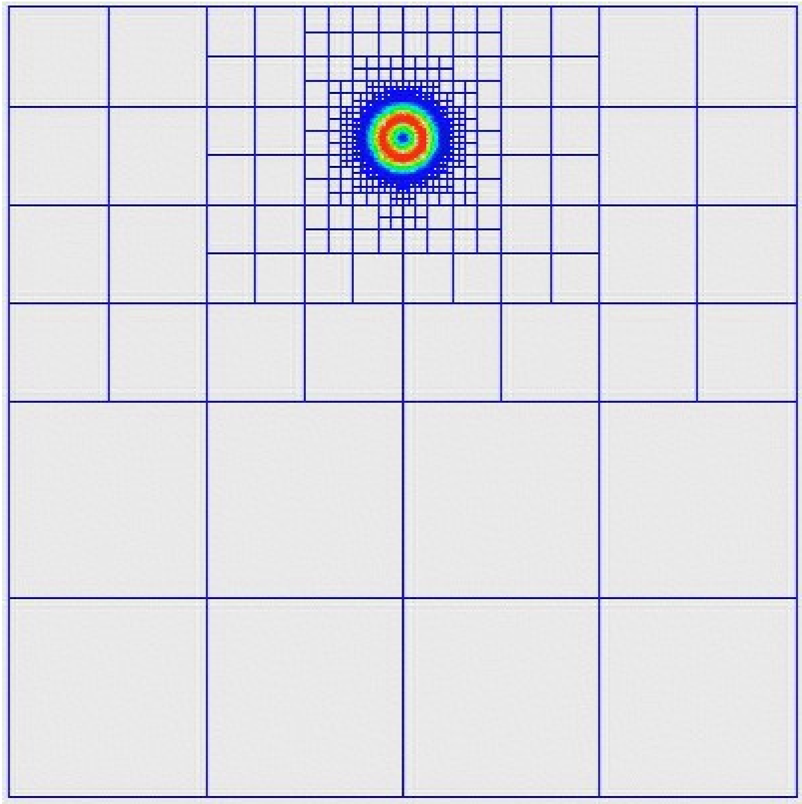




When do we gain by using Adaptivity?

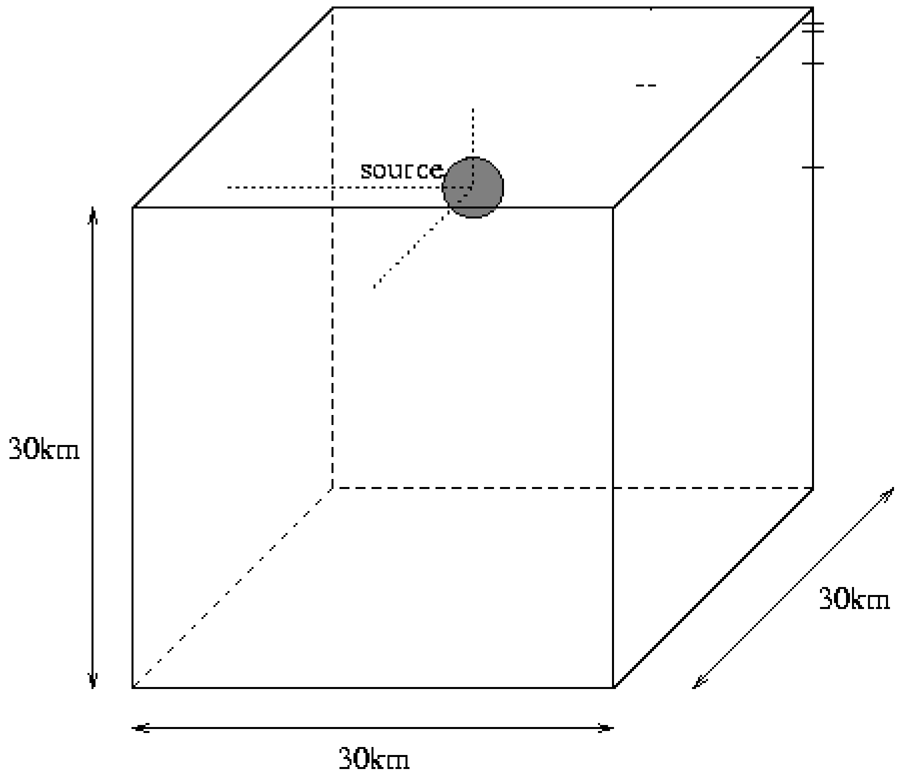
Counterexample: Wave equation in heterogeneous media often does not yield to adaptivity because the domain “is full of waves” after some time

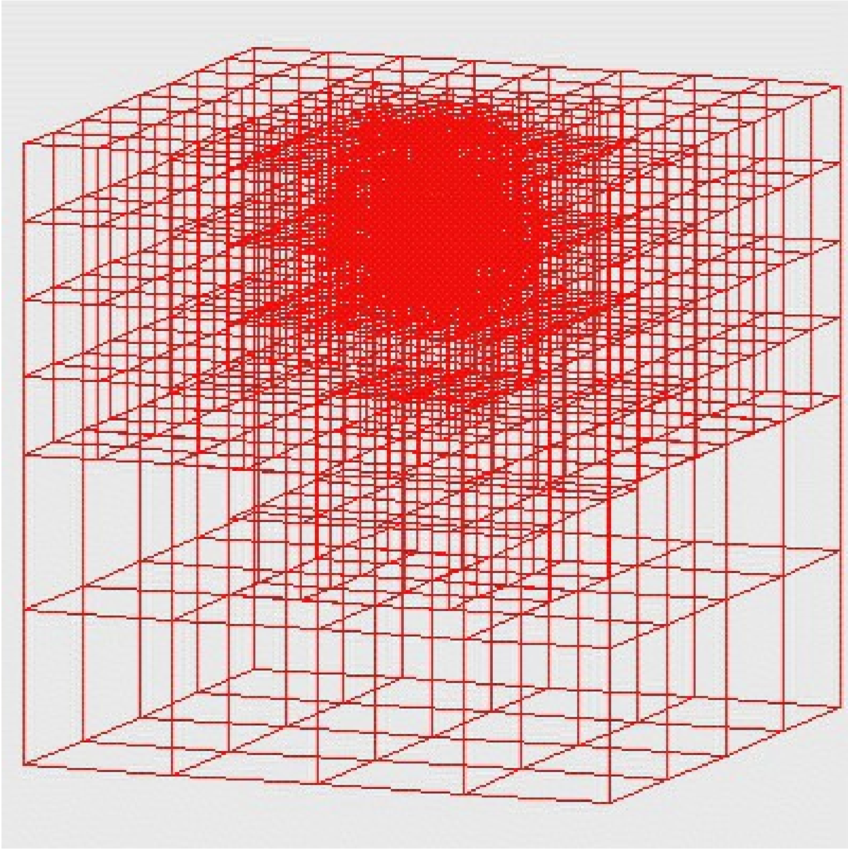




When do we gain by using Adaptivity?

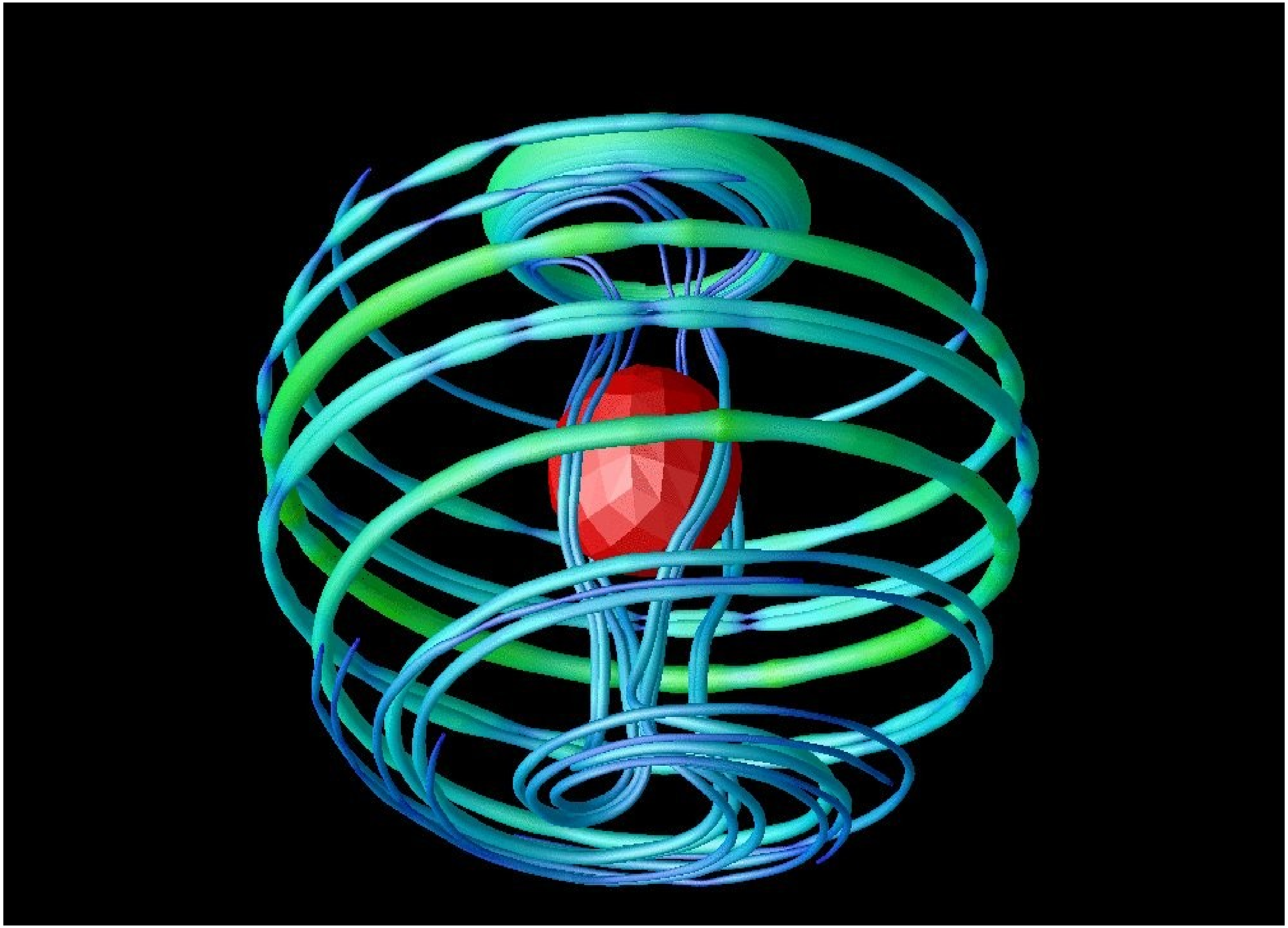
Counterexample: Wave equation in heterogeneous media often does not yield to adaptivity because the domain "is full of waves" after some time





When do we gain by using Adaptivity?

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



How to generate adaptive meshes

We need some sort of refinement criterion. For example:

- A mathematically well-founded error estimate, possibly taking into account what exactly we are interested in

- A heuristic indicator that tells us where a function is smooth and where it is not:
  - may not get the blessing of your mathematician
  - but is independent of progress in construction of estimators
  - turns out to be a really successful strategy and appears to be pretty universally applicable!

Most such indicators look at derivatives of (components of the) solution, for example:

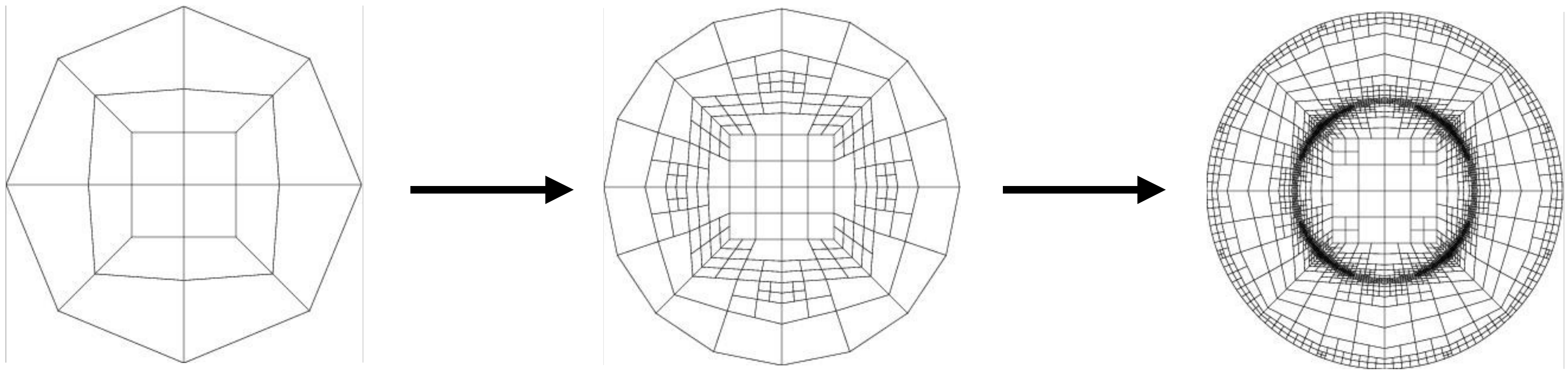
$$\eta_K = h_K^{1/2} \| [\partial u_h / \partial n] \| \quad \text{or} \quad \eta_K = h_K^2 \| \nabla_h^2 u_h \|$$

How to generate adaptive meshes

We need to use the refinement criterion for mesh refinement:

- For existing codes, one can use refinement criteria as weight function for the creation of a new mesh

- Better but more invasive: Allow codes to store meshes as objects that can be dynamically refined or coarsened



How to use adaptive meshes

What we need for existing codes:

- It is considered hard to convert existing codes to use adaptive meshes because the changes in data structures and algorithms are so pervasive. These changes involve:
  - mesh data structures
  - finite elements/finite difference stencils
  - handling of hanging node constraints
  - linear solvers/preconditioners
  - top-level logic

- People generally assume that it is simpler to write a new code from scratch (but there appears to be little evidence in this area)

- Rewrite may be less painful than thought because of experience gained from previous codes (e.g.: what discretization, which solvers work, and which don't) and using libraries that support adaptive finite element codes.

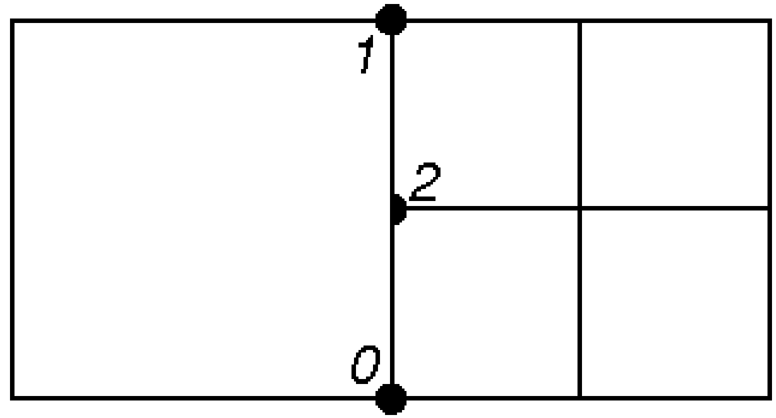
How to use adaptive meshes

What we need for new codes:

- Top-level code that loops over successively finer meshes
- Refinement criteria
- Code that transfers the solution from one mesh to another
- Solvers that are robust against widely varying mesh sizes
- Code that can deal with “hanging nodes”

Hanging nodes

Consider a mesh like this, with hanging node 2:



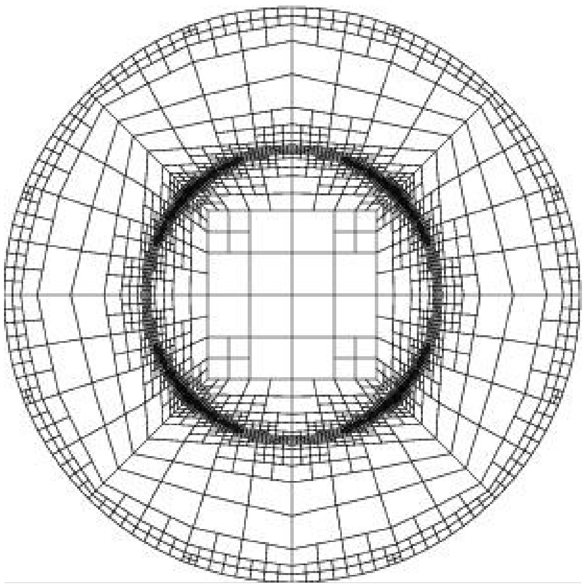
To make sure that finite element solution is continuous, we require

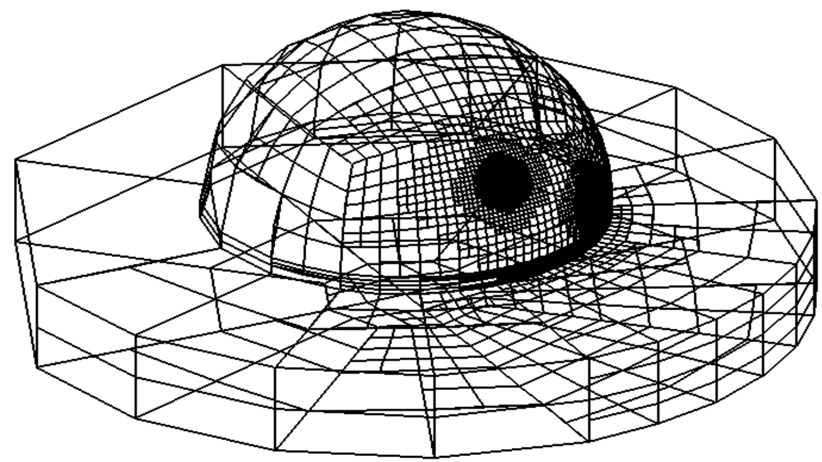
$$u_2 = \frac{1}{2} u_0 + \frac{1}{2} u_1$$

i.e. u_2 is not a “real” degree of freedom. It therefore needs to be eliminated from the linear system, and we set it to the correct value after solving.

Hanging nodes

For meshes with many hanging nodes:





We get a whole set of constraints,

$$u_i^{\text{constrained}} = C_{ij} u_j$$

and the linear system

$$Au = b$$

needs to be transformed to

$$\tilde{A} u^{\text{unconstrained}} = \tilde{b}^{\text{unconstrained}}, \qquad u_i^{\text{constrained}} = C_{ij} u_j$$

How to use adaptive meshes

What we need for new codes:

- Top-level code that loops over successively finer meshes
- Refinement criteria
- Code that transfers the solution from one mesh to another
- Solvers that are robust against widely varying mesh sizes
- Code that can deal with "hanging nodes"

Except for the first one (which is in application code), all these are available as building blocks in libraries such as deal.II !

Hanging nodes

Code example: Assembling matrix for the Laplace equation

```
active_cell_iterator cell = dof_handler.begin_active(),
                     endc = dof_handler.end();
for (; cell!=endc; ++cell) {
  cell_matrix = 0;
  cell_rhs = 0;
  fe_values.reinit (cell);
  for (q_point=0; q_point<n_q_points; ++q_point)
    for (i=0; i<dofs_per_cell; ++i)
      for (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));

   cell->distribute_local_to_global (cell_matrix,
                                     global_matrix)
}
```

Hanging nodes

Code example: Eliminating hanging node constraints

```
ConstraintMatrix hanging_node_constraints;
DoFTools::make_hanging_node_constraints
                        (dof_handler,  hanging_node_constraints);

hanging_node_constraints.condense (system_matrix);
hanging_node_constraints.condense (system_rhs);
```

The deal.II library

deal.II is a finite element software library:

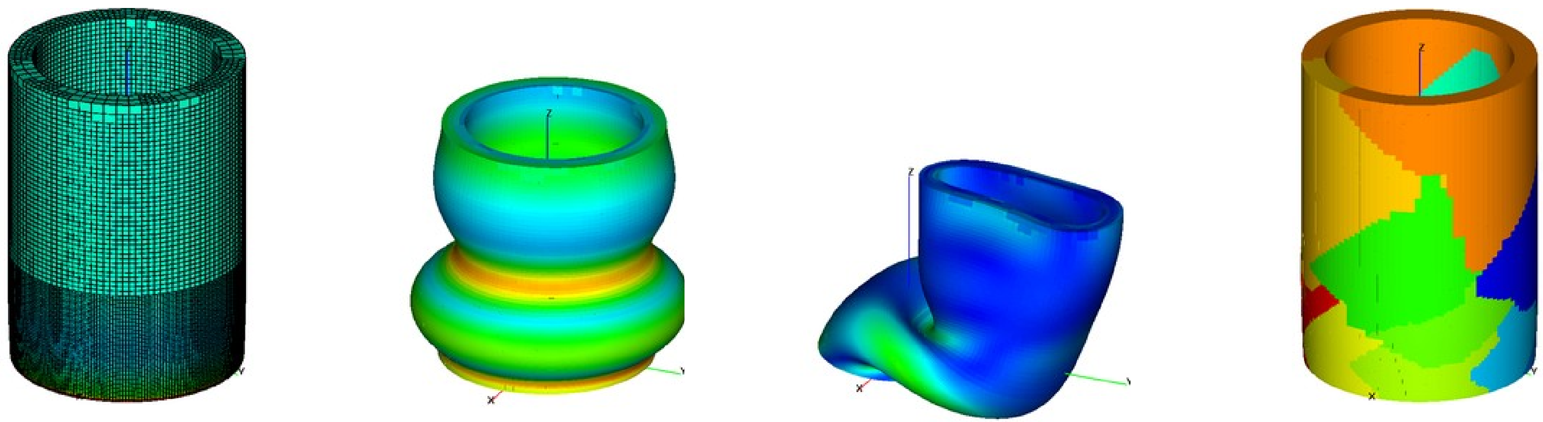
- Provides support for adaptive meshes in 1d, 2d, and 3d through a unified interface
- Has standard refinement indicators built in
- Provides a variety of different finite element types (continuous, discontinuous, mixed, Raviart-Thomas, ...)
- Low and high order elements available
- Full support for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

The deal.II library

- Interfaces to all major graphics programs
- Fairly widely distributed in the finite element/adaptivity community:
  - 200 downloads per month
  - >1000 hits on homepage
  - >10 publications per year based on deal.II
- Supports a wide variety of applications in all sciences
- Presently over 350,000 lines of C++ code
- More than 4000 pages of documentation
- ~25 tutorial programs that explain the use of the library in detail, starting from very simple to parallel quasistatic elasticity/multiphase flow/neutron transport/... applications
- Open Source, active development

Challenges in adaptivity

- **Parallelization, partitioning and load balancing**

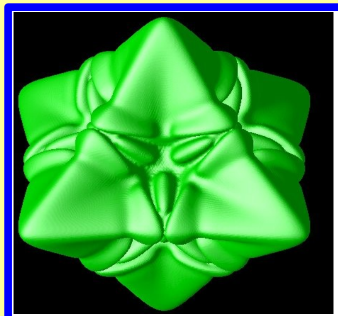
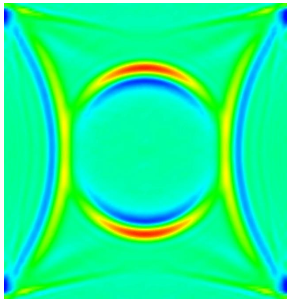
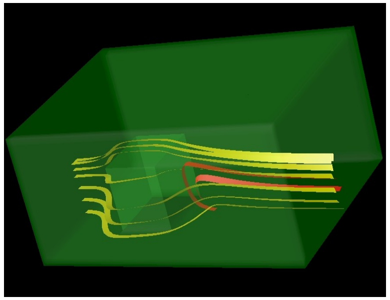
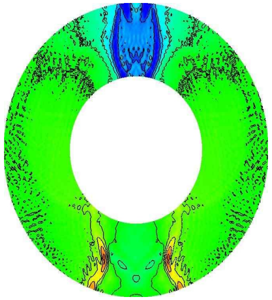
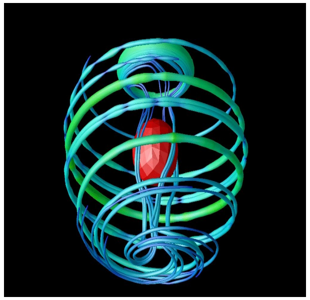
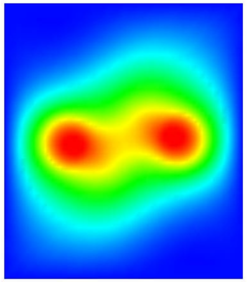


After computing the solution, the refinement indicator tells me which cells to refine

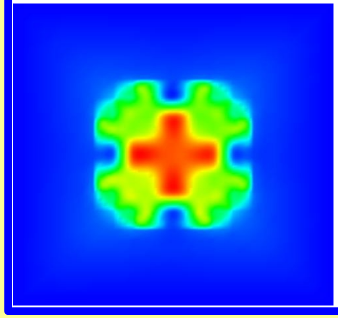
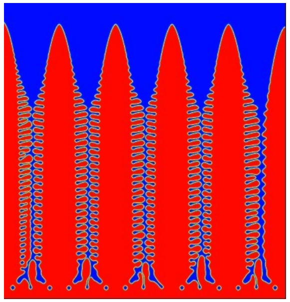
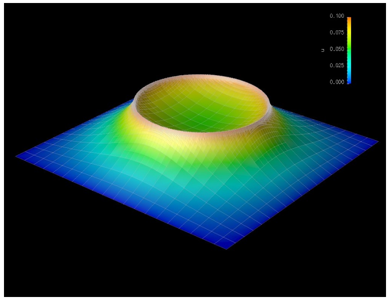
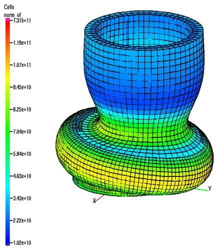
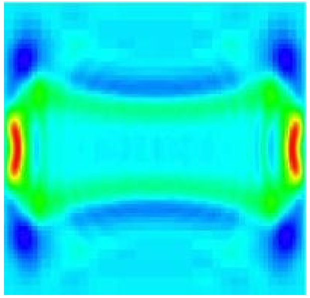
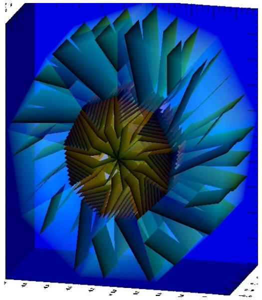
Problem: The individual blocks are now no longer load balanced!
Solution: We need to partition our domain again after refinement.

Problem: Requires access to the entire mesh to be efficient!
Solution: ???

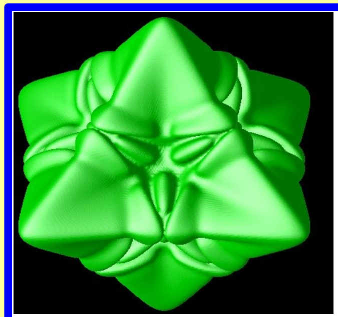
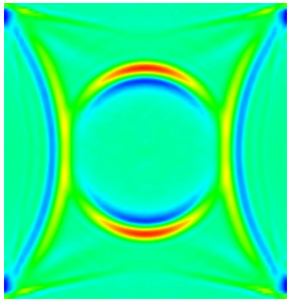
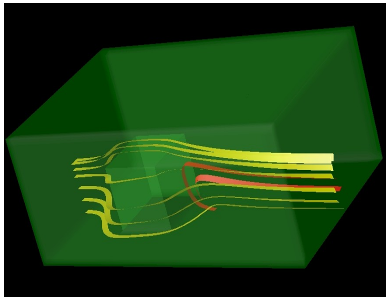
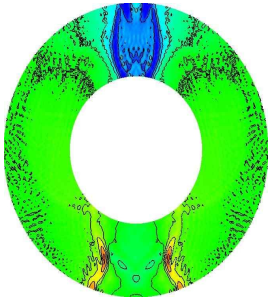
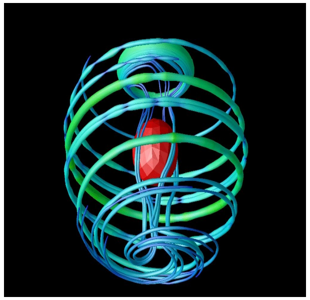
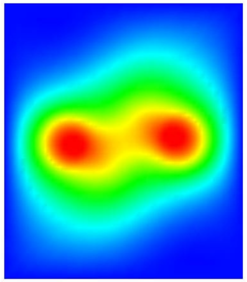
Conclusions

- Adaptivity promises better resolution with less work
- Requires substantial changes to codes
- It may be simpler to re-write a code
- **But:** New programs can draw from very large libraries of building blocks!

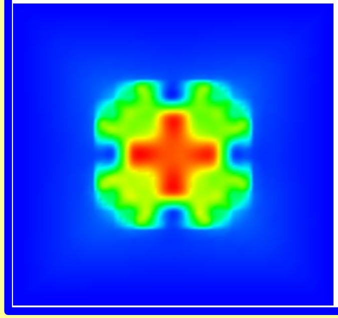
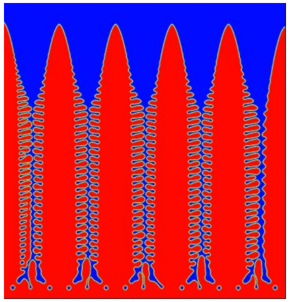
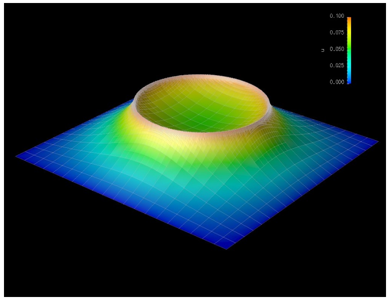
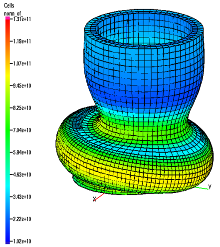
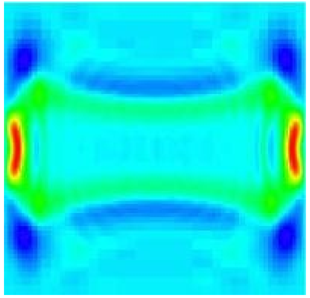
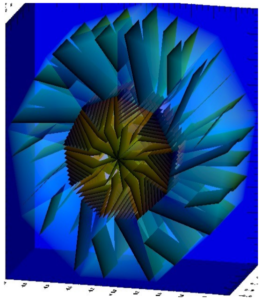







The deal.II library

Visit the deal.II library:

http://www.dealii.org

www.dealii.org