Advanced Higher Order Finite Elements
From Theory to Application with Netgen/NGSolve

Christoph Lehrenfeld, Joachim Schöberl

Center for Computational Engineering Sciences (CCES)
RWTH Aachen University

Vancouver 2009, November 10, 2009

content

1. Netgen/NGSolve
   - The Mesh Generator Netgen
   - Finite Element Solver NGSolve

2. Hybrid DG Navier Stokes
   - Hybrid DG for scalar Laplace Operator
   - Hybrid DG Formulation for scalar convection
   - Hybrid DG Navier Stokes Formulation
   - Implementation aspects of Hybrid Navier Stokes FEM

3. other projects
   - NGSflow
   - Maxwell DG TD

Netgen/NGSolve

The Mesh Generator Netgen

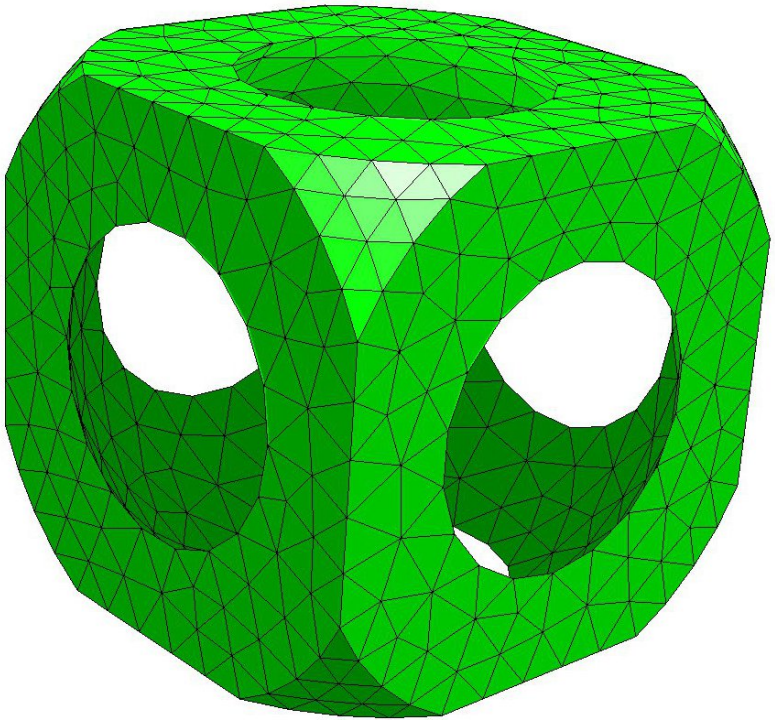
- Developed since 1994, developers: J.Schöberl, R. Gaisbauer (IGES/Step), J. Gerstmayr (STL), M. Wabro (customizing, SAT)
- 1998-2003: More than 400 non-commercial and 10 commercial licenses signed
- Since 2003: Open Source under LGPL, 300 downloads per month
- Included in several commercial as well as free packages
- Included in Linux distributions, Fink programme library (Apple)

The Mesh Generator Netgen

- Plane, surface, and volume mesh generator
- Input from Constructive Solid Geometry (CSG), Triangulated Surfaces (STL), Boundary representation (IGES/Step/SAT)
- Triangular and tetrahedral elements
- Delaunay and advancing front mesh generation algorithms
- Automatic local mesh-size control
- Anisotropic mesh generation (prisms and pyramids)
- Various mesh refinement algorithms (bisection, hp-refinement, etc.)
- Arbitrary order curved elements

Constructive Solid Geometry (CSG)

Complicated objects are described by Eulerian operations applied to (simple) primitives.

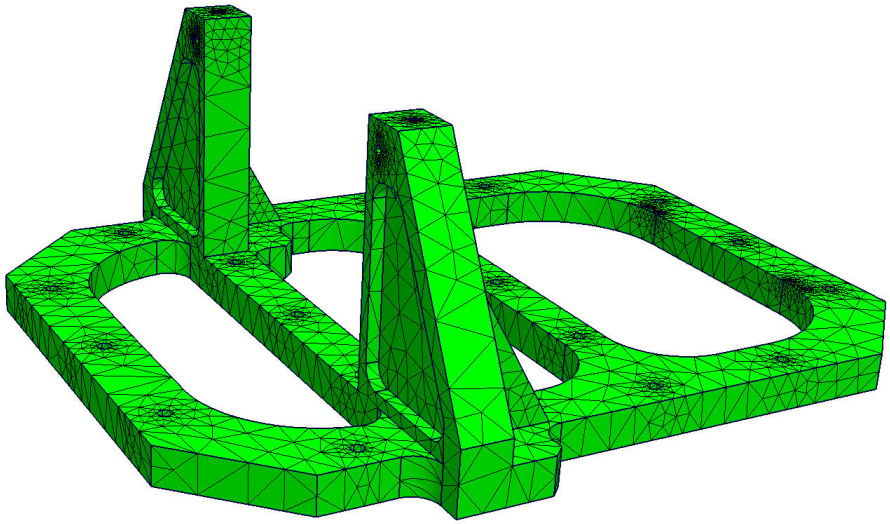


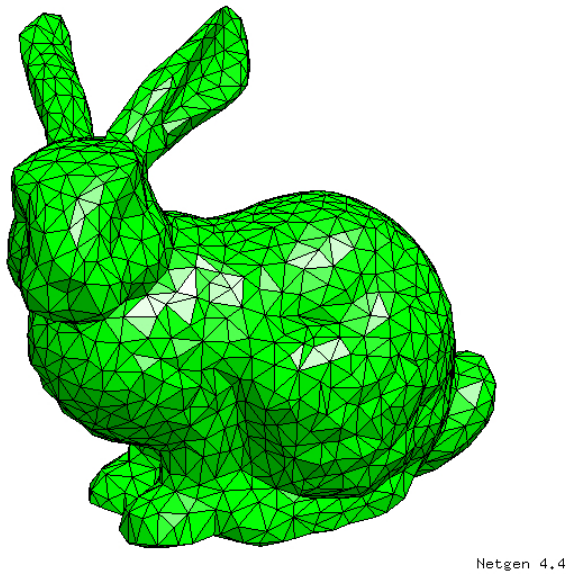
```
solid cube =
        plane (0, 0, 0; 0, 0, -1)
    and plane (0, 0, 0; 0, -1, 0)
    and plane (0, 0, 0; -1, 0, 0)
    and plane (100, 100, 100; 0, 0, 1)
    and plane (100, 100, 100; 0, 1, 0)
    and plane (100, 100, 100; 1, 0, 0);
solid main =
        cube
    and sphere (50, 50, 50; 75)
    and not sphere (50, 50, 50; 60);
```

Very useful for simple to moderate complexity

Some Meshes generated by Netgen from Step/STL

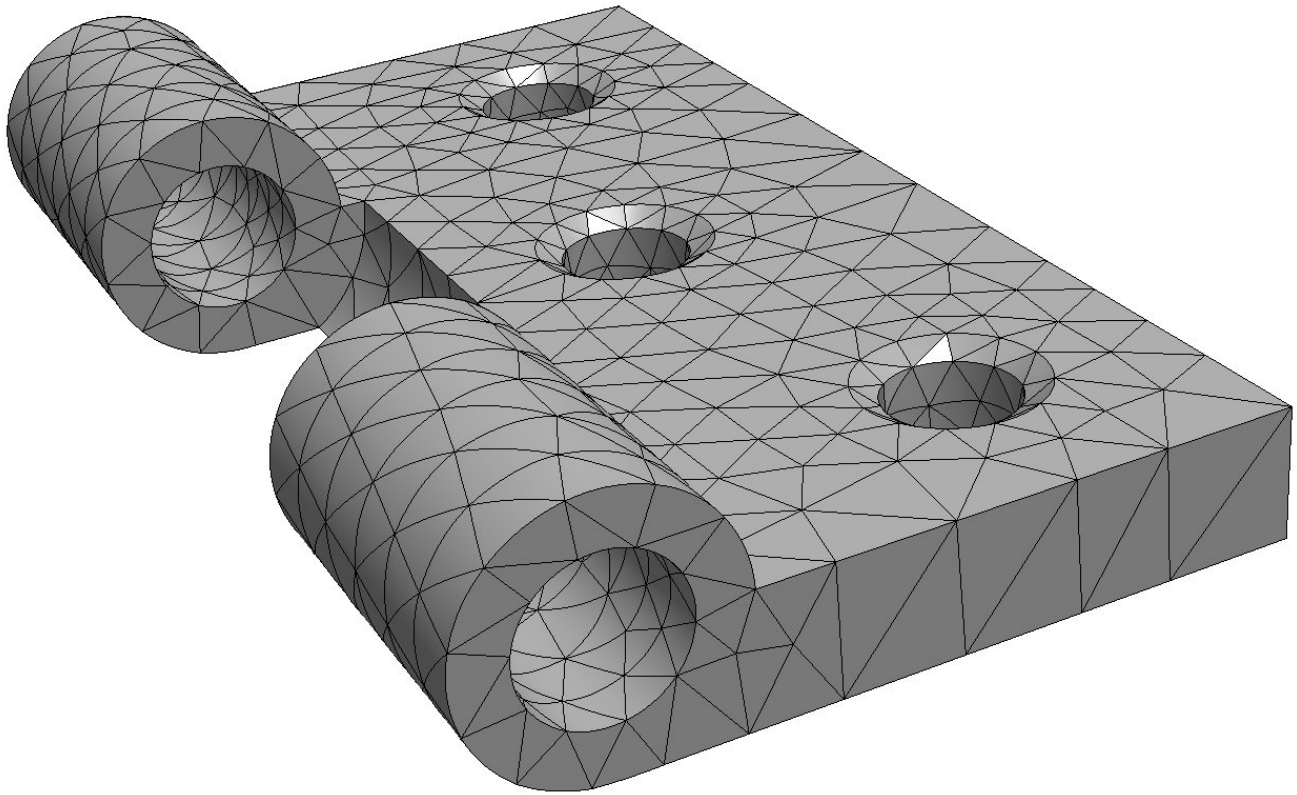
A machine frame imported via the Step - format, a bunny imported from STL





curved elements

Curved elements of arbitrary order by **projection based interpolation** [Demkowicz]:
Element deformation is described by hierarchical finite element basis functions.
Define edge-modes by H_0^1-projection onto edges, and face-modes by H_0^1-projections onto faces.



Finite Element Solver NGSolve

- Arbitrary order finite elements for 1D/2D/3D:
  segments, trigs, quads, tets, prisms, (pyramids), hexes
- Scalar elements, vector-valued elements for $H(\mathrm{curl})$ and $H(\mathrm{div})$, tensor-valued elements for elasticity.
- Integrators for Poisson, elasticity, Stokes, Maxwell, etc.
- Multigrid preconditioners with different block smoothers
- Solvers for boundary value problems, time-dependent problems, non-linear problems, eigenvalue problems, shape optimization problems, etc.
- hp-refinement
- Error estimators (ZZ, equilibrated residual, hierarchical)

- adaptive refinement strategies
- Can be connected to any mesh handler providing full topology (e.g. Netgen)
- Intensively object oriented C++ (Compile time polymorphism by templates)
- open source on LGPL

NGSolve script file for Poisson example

$$\text{Find } u \in H^1 \text{ s.t.} \qquad \int_\Omega \lambda \nabla u \cdot \nabla v \, dx + \int_{\partial\Omega} \alpha u v \, ds = \int_\Omega f v \, dx \qquad \forall\, v \in H^1$$

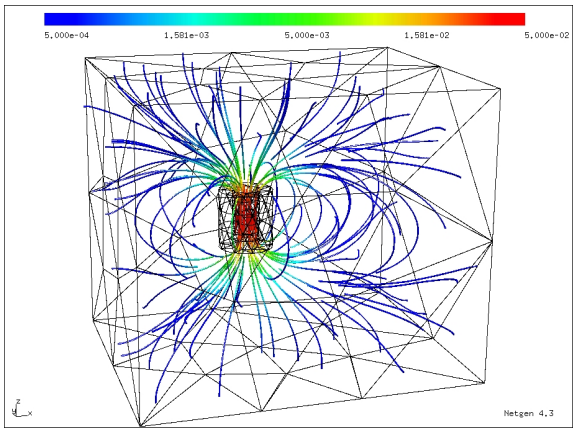
```
define coefficient lam      1,
define coefficient alpha    1e5, 1e5, 1e5, 0,
define coefficient cf       sin(x)*y,

define fespace v -h1 -order=4
define gridfunction u -fespace=v

define bilinearform a -fespace=v -symmetric
laplace lam
robin alpha

define linearform f -fespace=v
source cf

define preconditioner c -type=multigrid -bilinearform=a
   -smoothingsteps=1 -smoother=block

numproc bvp np1 -bilinearform=a -linearform=f -gridfunction=u
   -preconditioner=c -prec=1e-8
```

Central NGSolve classes

- FiniteElement:
  Provides shape functions and derivatives on reference element
- ElementTransformation:
  Represents mapping to physical elements, computes Jacobian
- Integrator:
  Computes element matrices and vectors
- FESpace:
  Provides global dofs, multigrid-transfers and smoothing blocks
- BilinearForm/LinearForm:
  Maintains definition of forms, provides matrix and vectors
- PDE:
  Container to stores all components

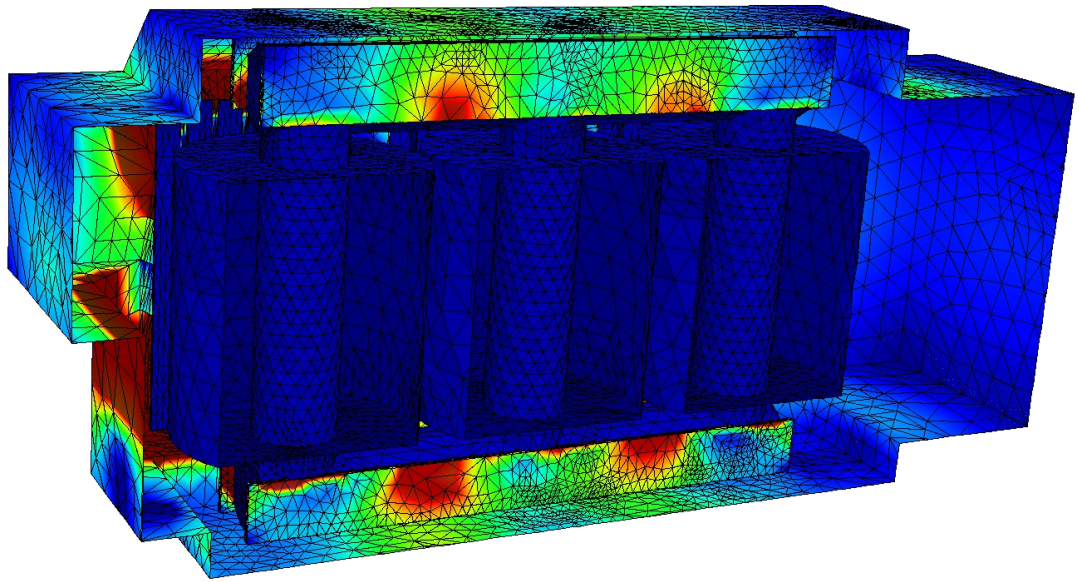
Simulation of Electromagnetic Fields

A simple coil:



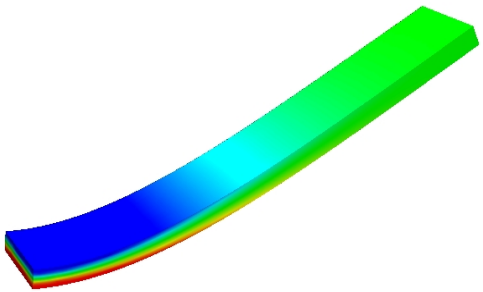
Described by
Maxwell's equations

A transformer built by Siemens - EBG, Linz:

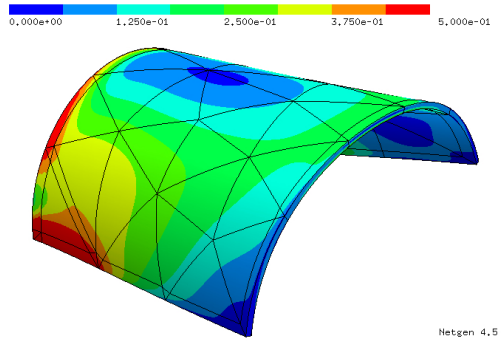


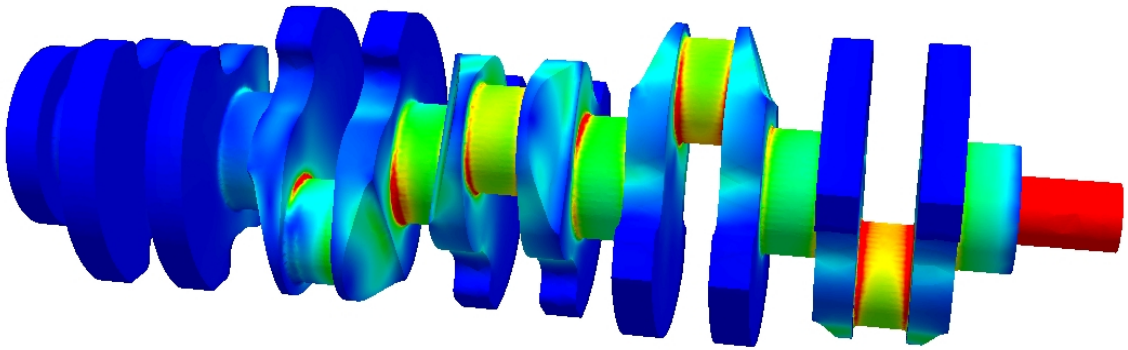
Simulation of Mechanical Deformation and Stresses

Elastic beam:



Shell structures:





incompressible Navier Stokes Equations

incompressible Navier Stokes Equations

$$\begin{aligned} \frac{\partial u}{\partial t} - \operatorname{div}(2\nu\ \varepsilon(u) - u \otimes u - pI) &= f \\ \operatorname{div}\ u &= 0 \\ + b.c. & \end{aligned}$$

variational formulation after simple time-stepping:
(The nonlinear convection term is treated explicitely)

Find $\hat{u} \in H^1, \hat{p} \in L^2$ such that

$$\frac{1}{\tau} M(\hat{u}, v) + A^{\nu}(\hat{u}, v) + B(\hat{p}, v) + B(q, \hat{u}) = \frac{1}{\tau} M(u, v) + f(v) - A^{c}(u; u, v)$$

for all $\hat{v} \in H^1, \hat{q} \in L^2$

where M, A^{ν}, B are according Bilinearforms and A^c is the convective Trilinearform.

$H(\mathrm{div})$-conforming elements for Navier Stokes

[Cockburn, Kanschat, Schötzau]:
DG methods for the incompressible Navier-Stokes equations cannot be both locally conservative as well as energy-stable unless the approximation to the convective velocity is exactly divergence-free.

- $u_h \in V_h := BDM_k \subset H(\mathrm{div})$, $p_h \in Q_h := P_{k-1} \subset L_2$.
- u_h is exactly div-free ($\Rightarrow$ stability of discrete time-dependent equation)

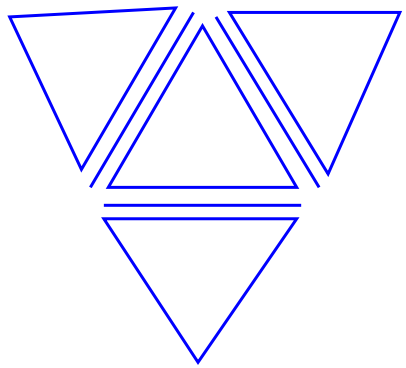
Challenge:
discretization of viscous Bilinearform A^ν and convective Bilinearform A^c

Discretization of the viscous term

As a scalar equivalent to the viscoues term we first take a look at the discretization of the scalar laplace bilinearform $A(u,v) = \int_\Omega \nabla u \nabla v \; dx$

We are using a hybrid versions of interior penalty methods [Cockburn+Gopalakrishnan+Lazarov]

- unknowns u in elements and unknowns $\lambda = \text{trace}\, u$ on facets.



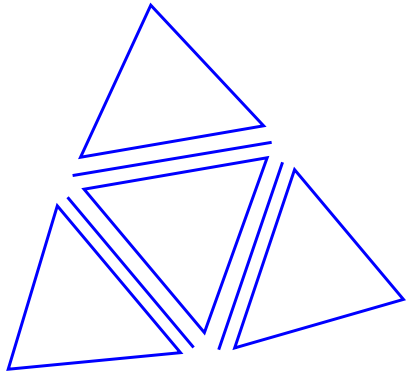
Find $u \in P^k(T)$ and $\lambda \in P^k(F)$ such that for all $v \in P^k(T)$ and $\mu \in P^k(F)$:

$$A(u,\lambda,v,\mu) = \sum_T \Big\{ \int_T \nabla u \cdot \nabla v - \int_{\partial T} \frac{\partial u}{\partial n}(v-\mu) - \int_{\partial T} \frac{\partial v}{\partial n}(u-\lambda) + \frac{\alpha p^2}{h} \int_{\partial T} (u-\lambda)(v-\mu)$$

consistency symmetry stability

Element unknowns now only couple with unknowns of the own element and unknowns of the surrounding facets

- more unknowns, but less matrix-entries
- fits into standard element-based assembling
- allows condensation of element unknowns



Discretization of the convection term

Again, we look at the scalar model problem:

$$\begin{aligned} -\varepsilon \Delta u + b \cdot \nabla u &= f && \text{in } \partial\Omega \\ u &= 0 && \text{on } \partial\Omega \end{aligned}$$

HDG Formulation:

$$A^{\nu}(u, \lambda, v, \mu) + A^{c}(u, \lambda, v, \mu) = \int fv$$

with diffusive term $A^{\nu}(.,.)$ from above and upwind-discretization for convective term

$$A^{c}(b; u, \lambda, v, \mu) = \sum_{T} \Big\{ - \int bu \cdot \nabla v + \int_{\partial_T} b_n u^{\text{up}} v \Big\}$$

with upwind choice

$$u^{\text{up}} = \begin{cases} \lambda & \text{if } b_n < 0, \text{ i.e. inflow edge} \\ u & \text{if } b_n > 0, \text{ i.e. outflow edge} \end{cases}$$

[H. Egger + J.Schöberl, submitted]

Back to Navier Stokes

Now:

- unknowns u_T in elements ($H(\mathrm{div})$-conforming)
- unknowns for the tangential component $u_F^t = \mathrm{trace}\, u^t$ on facets.

Viscosity:

$$\begin{aligned} A^\nu(u_T, u_F, v_T, v_F) &= 2\int_T \nu\varepsilon(u_T) : \varepsilon(v_T)\, dx \\ &- 2\int_{\partial T} \nu\varepsilon(u_T)\cdot\mathbf{n}\,(v_T - v_F^t)\, ds - 2\int_{\partial T} \nu\varepsilon(v)\cdot\mathbf{n}\,(u_T - u_F^t)\, ds \\ &+ \nu\beta\frac{p^2}{h}\int_{\partial T}(u_T^t - u_F^t)\cdot(v_T^t - v_F^t)\, ds \end{aligned}$$

Convection:

Here we only need to choose an upwind value for the tangential component as the normal component is continuous by $H(\mathrm{div})$-conformity.

$$-\int_\Omega \mathrm{div}(u\otimes u)\; v\; \mathrm{d}x \approx A^c(u;\ldots) = \int_\Omega u\otimes u : \nabla v\; dx - \int_{\partial\Omega} u^{up}\otimes u^{up}\cdot\mathbf{n}\; v\; ds$$

summary of ingredients and properties

So, we have a new Finite Element Method for Navier Stokes, with

- $H(\mathrm{div})$-conforming Finite Elements
- Hybrid Discontinuous Galerkin Method for viscous terms
- Upwind flux (in HDG-sence) for the convection term

leading to solutions, which are

- locally conservative
- energy-stable ($\frac{d}{dt}\|u\|_{L_2}^2 \leq \frac{C}{\nu}\|f\|_{L_2}^2$)
- exactly incompressible
- static condensation
- standard finite element assembly is possible
- less matrix entries than for std. DG approaches
- reduced basis possible (later)

saddlepoint problem $\longrightarrow$ ill-conditioned s.p.d. problem

Instead of looking at the saddlepoint problem, we impose the divergencefree-constraint by a penalty term formulation leading to:

Find $\hat{u} \in H^1$ such that

$$\frac{1}{\tau} M(\hat{u}, v) + A^{\nu}(\hat{u}, v) + P(\hat{u}, v) = \frac{1}{\tau} M(u, v) + f(v) - A^{c}(u; u, v)$$

with $P(\hat{u}, v) = \alpha \int_{\Omega} \operatorname{div}(\hat{u}) \operatorname{div}(v) dx$ for all $\hat{v} \in H^1$

This is a symmetric positive definite, but ill-conditioned system.

We can circumvent this problem with additive Schwarz preconditioning using kernel-preserving smoothing which is possible with our $H(\mathrm{div})$-conforming discretization.

Implementation

To implement this method with NGSolve we need:

- a new `FESpace`
- new `Bilinearformintegrators`
- new Application of $A^c(u; u, v)$
- a new `numproc` doing the time stepping

FESpace

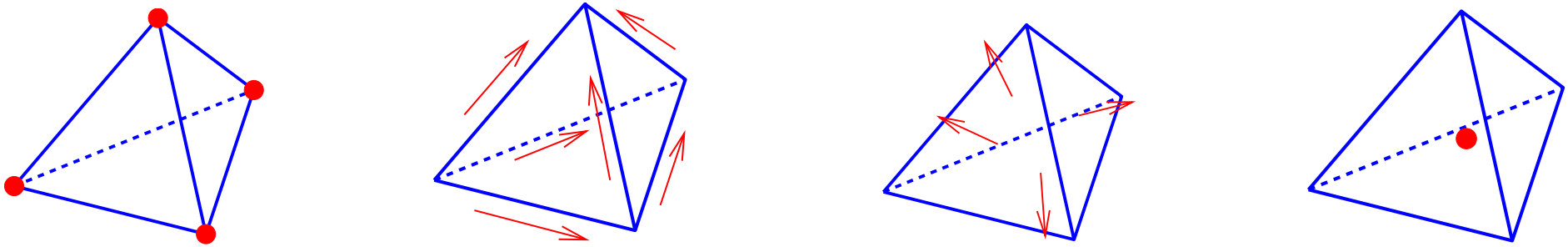
NGSolve comes with

- an $H(\mathrm{div})$ FESpace
- a vector-valued FacetFESpace
- a CompoundFESpace

```
class HybridStokesFESpace : public CompoundFESpace
...
//Constructor:
  {
    Array<const FESpace*> spaces(2);
    int order = int (flags.GetNumFlag ("order", 1));
    Flags uflags, lamflags;
    uflags.SetFlag ("order", order);
    uflags.SetFlag ("hodivfree");
    lamflags.SetFlag ("order", order);
    spaces[0] = new HDivHighOrderFESpace (ma, uflags);
    spaces[1] = new VectorFacetFESpace (ma, lamflags);
    Flags compflags;
    HybridStokesFESpace * fes = new CompoundFESpace (ma, spaces, compflags);
    return fes;
  }
```

The **de Rham** for continuous and discrete function spaces

$$
\begin{array}{ccccccc}
H^1 & \xrightarrow{\nabla} & H(\mathrm{curl}) & \xrightarrow{\mathrm{Curl}} & H(\mathrm{div}) & \xrightarrow{\mathrm{Div}} & L^2 \\
\cup & & \cup & & \cup & & \cup \\
W_h & \xrightarrow{\nabla} & V_h & \xrightarrow{\mathrm{Curl}} & Q_h & \xrightarrow{\mathrm{Div}} & S_h
\end{array}
$$

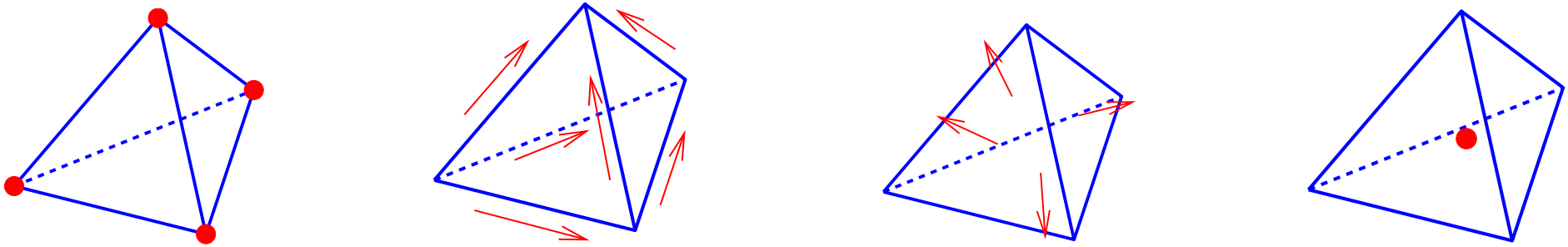


Used for constructing high order finite elements [JS+S. Zaglmayr, '05, Thesis Zaglmayr '06]

$$
\begin{aligned}
W_{hp} &= W_{\mathcal{L}_1} + \mathrm{span}\{\varphi^W_{h.o.}\} \\
V_{hp} &= V_{\mathcal{N}_0} + \mathrm{span}\{\nabla\varphi^W_{h.o.}\} + \mathrm{span}\{\varphi^V_{h.o.}\} \\
Q_{hp} &= Q_{\mathcal{R}T_0} + \mathrm{span}\{\mathrm{curl}\,\varphi^V_{h.o.}\} + \mathrm{span}\{\varphi^Q_{h.o.}\} \\
S_{hp} &= S_{\mathcal{P}_0} + \mathrm{span}\{\mathrm{div}\,\varphi^Q_{h.o.}\}
\end{aligned}
$$

The **de Rham** for continuous and discrete function spaces

$$\begin{array}{ccccccc} H^1 & \xrightarrow{\nabla} & H(\mathrm{curl}) & \xrightarrow{\mathrm{Curl}} & H(\mathrm{div}) & \xrightarrow{\mathrm{Div}} & L^2 \\ \cup & & \cup & & \cup & & \cup \\ W_h & \xrightarrow{\nabla} & V_h & \xrightarrow{\mathrm{Curl}} & Q_h & \xrightarrow{\mathrm{Div}} & S_h \end{array}$$

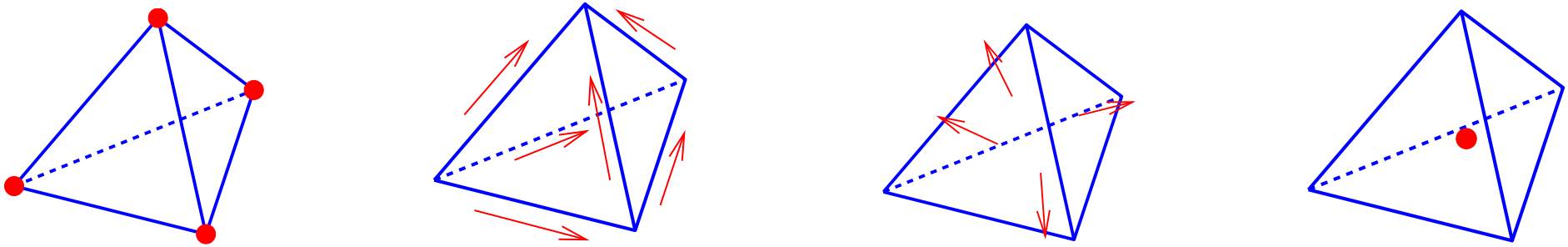


Used for constructing high order finite elements [JS+S. Zaglmayr, '05, Thesis Zaglmayr '06]

$$\begin{aligned} W_{hp} &= W_{\mathcal{L}_1} + \mathrm{span}\{\varphi^W_{h.o.}\} \\ V_{hp} &= V_{\mathcal{N}_0} + \mathrm{span}\{\nabla\varphi^W_{h.o.}\} + \mathrm{span}\{\varphi^V_{h.o.}\} \\ Q_{hp} &= Q_{\mathcal{R}T_0} + \mathrm{span}\{\mathrm{curl}\,\varphi^V_{h.o.}\} + \mathrm{span}\{\varphi^Q_{h.o.}\} \\ S_{hp} &= S_{\mathcal{P}_0} + \mathrm{span}\{\mathrm{div}\,\varphi^Q_{h.o.}\} \end{aligned}$$

The **de Rham** for continuous and discrete function spaces

$$\begin{array}{ccccccc} H^1 & \xrightarrow{\nabla} & H(\mathrm{curl}) & \xrightarrow{\mathrm{Curl}} & H(\mathrm{div}) & \xrightarrow{\mathrm{Div}} & L^2 \\ \cup & & \cup & & \cup & & \cup \\ W_h & \xrightarrow{\nabla} & V_h & \xrightarrow{\mathrm{Curl}} & Q_h & \xrightarrow{\mathrm{Div}} & S_h \end{array}$$



Used for constructing high order finite elements [JS+S. Zaglmayr, '05, Thesis Zaglmayr '06]

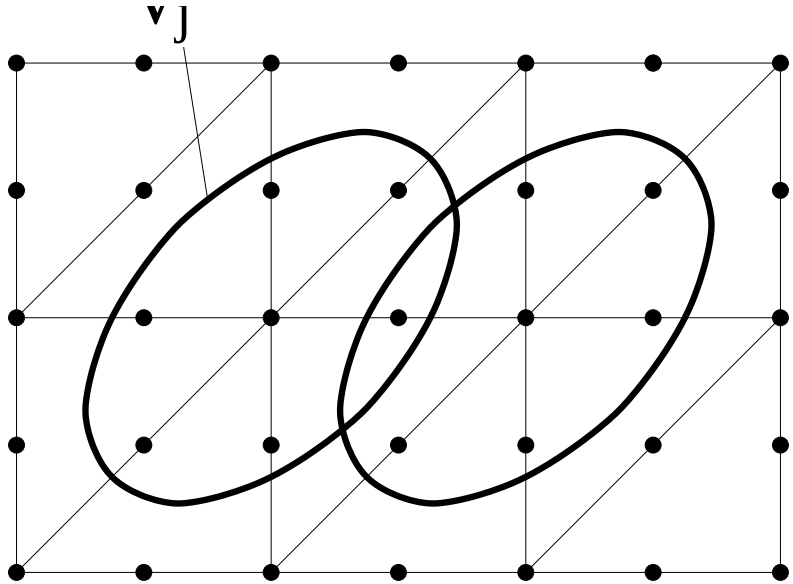
$$\begin{aligned} W_{hp} &= W_{\mathcal{L}_1} + \mathrm{span}\{\varphi^W_{h.o.}\} \\ V_{hp} &= V_{\mathcal{N}_0} + \mathrm{span}\{\nabla\varphi^W_{h.o.}\} + \mathrm{span}\{\varphi^V_{h.o.}\} \\ Q_{hp} &= Q_{\mathcal{R}T_0} + \mathrm{span}\{\mathrm{curl}\,\varphi^V_{h.o.}\} + \cancel{\mathrm{span}\{\varphi^Q_{h.o.}\}} \\ S_{hp} &= S_{\mathcal{P}_0} + \mathrm{span}\{\mathrm{div}\,\varphi^Q_{h.o.}\} \end{aligned}$$

FESpace

The FESpace is also responsible for the smoothingblocks.
You just have to implement:

```
Table<int> * CreateSmoothingBlocks (int type) const
```

The table has as muchs rows as blocks you want and each row has all degrees of freedom to the according block.
To use kernel-preserving smoothers, we use as much blocks as we have vertices and every vertex gets the degrees of freedom of all surrounding edges and elements (in 2D).



BilinearFormIntegrators

For the Massmatrix and the Penalty-Matrix there are `BilinearForms` for $H(\text{div})$-elements. With the help of so called `CompoundBilinearFormIntegrators` they can be easily implemented:

```
bfi = new DivDivHDivIntegrator<D>
    (new ConstantCoefficientFunction(1e6));
bfi = new CompoundBilinearFormIntegrator (*bfi, 0);
```

BilinearFormIntegrators

For A^ν we have to implement a new `BilinearFormIntegrator`

```
template<int D>  class HybridStokesIntegrator
    : public BilinearFormIntegrator
```

which essentially has to provide the function

```
virtual void ComputeElementMatrix (const FiniteElement & fel,
                                   const ElementTransformation & eltrans,
                                   FlatMatrix<double> & elmat,
                                   LocalHeap & lh) const
{
  const CompoundFiniteElement & cfel =
    dynamic_cast<const CompoundFiniteElement&> (fel);
  const HDivFiniteElement<D> & fel_u =
    dynamic_cast<const HDivFiniteElement<D> &> (cfel[0]);
  const VectorFacetVolumeFiniteElement & fel_facet =
    dynamic_cast<const VectorFacetVolumeFiniteElement &> (cfel[1]);
...
```

Convection-Linearform

As we treat the Convection explicitely and so in every time step we have to compute a new r.h.s.. This is implemented by

```
void CalcConvection (const VVector<double> & vecu,
                     VVector<double> & conv, LocalHeap & clh)
```

When using direct solvers, this is the most time consuming part of the time stepping.
Parallelized with OpenMP.

The timestepping numproc

Now we have all the ingredients that where missing to solve the unsteady Navier Stokes equations. We just have to put it all together:

```
template <int D>
class NumProcNavierStokes : public NumProc
...
NumProcNavierStokes (PDE & apde, const Flags & flags) : NumProc (apde)
{
  //Gathering all coefficients and Integrators
  //for BilinearForms, LinearForm, etc...
  //Add Boundary conditions
  ...
}
...
```

The timestepping numproc

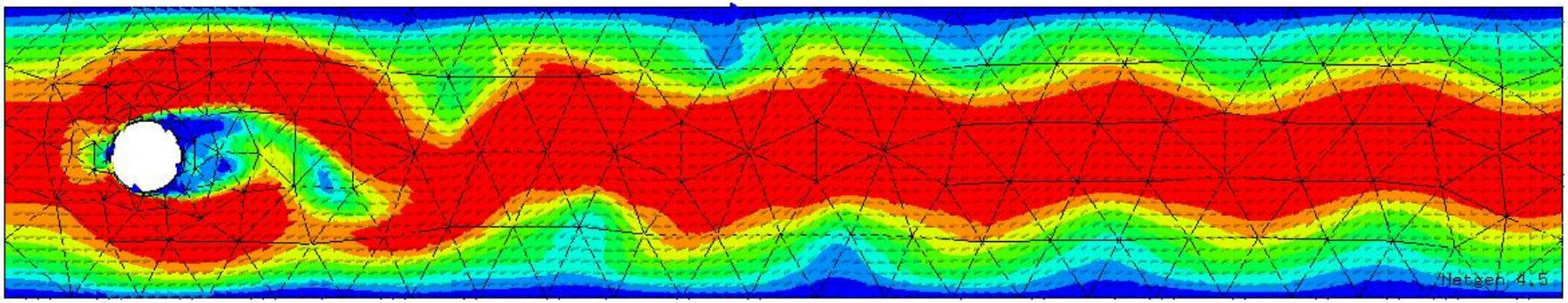
```
...
virtual void Do (LocalHeap & lh)
{
  //Assemble time independent BilinearForms, LinearForm, etc...
  //set startvalues
  bfsum -> GetMatrix().AsVector() = 1.0/tau * bfmass -> GetMatrix().AsVector()
                                    + bfvisc -> GetMatrix().AsVector();
  BaseMatrix & invmat = bfsum -> GetMatrix() . InverseMatrix();
  for (double t = 0; t < tend; t+=tau)
  {
    CalcConvection (vecu, conv, lh);
    res = vecf + conv + 1.0/tau * bfmass -> GetMatrix() * vecu;
    vecu = invmat * res;
    Ng_Redraw();
  }//end of time loop
}
```

The input file for a Navier Stokes Example

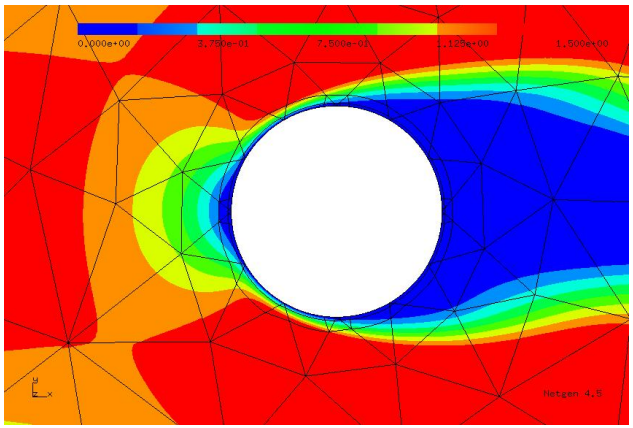
```
# boundaries:
# 1 .. inflow
# 2 .. outflow
# 3 .. wall (no slip)
# 4 .. cylinder  (no slip)

geometry = cylinder.in2d
mesh = cylinder.vol

shared = libnavierstokes

define constant hpref = 2
define constant hpref_geom_factor = 0.1
define constant geometryorder = 4

define coefficient inflow
(4*y*(0.41-y)/(0.41*0.41)), 0, 0, 0, 0, 0,

numproc navierstokeshdiv np1
  -order=5 -tau=6e-4 -tend=100 -alpha=30 -nu=2e-3  -umax=3 -wallbnds=[1,3,4]
```

Flow around a disk in 2D

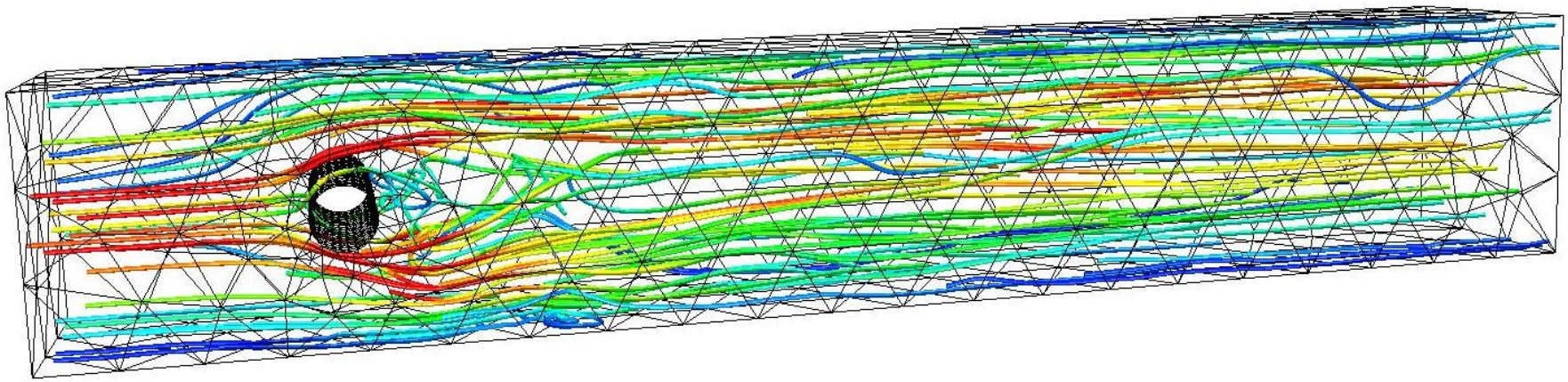
Re = 100, 5^{th}-order elements



Boundary layer mesh around cylinder:



Flow around a cylinder in 3D



Software

Netgen

- Downloads: http://sourceforge.net/projects/netgen-mesher/
- Help & Instructions: http://netgen-mesher.wiki.sourceforge.net

NGSolve

- Downloads: http://sourceforge.net/projects/ngsolve/
- Help & Instructions: http://sourceforge.net/apps/mediawiki/ngsolve

NGSflow (including the presented Navier Stokes Solver)

- Downloads: http://sourceforge.net/projects/ngsflow/
- Help & Instructions: http://sourceforge.net/apps/mediawiki/ngsflow

NGSflow

NGSflow is the flow solver add-on to NGSolve.
It includes:

- The presented Navier Stokes Solver
- A package solving for heat driven flow
- (coming soon:) 2D incompressible Navier Stokes with turbulence model

heatdrivenflow

changes in density are small:

- incompressibility model is still acceptable
- changes in density just cause some buoyancy forces

Navier Stokes is just modified at the force term:

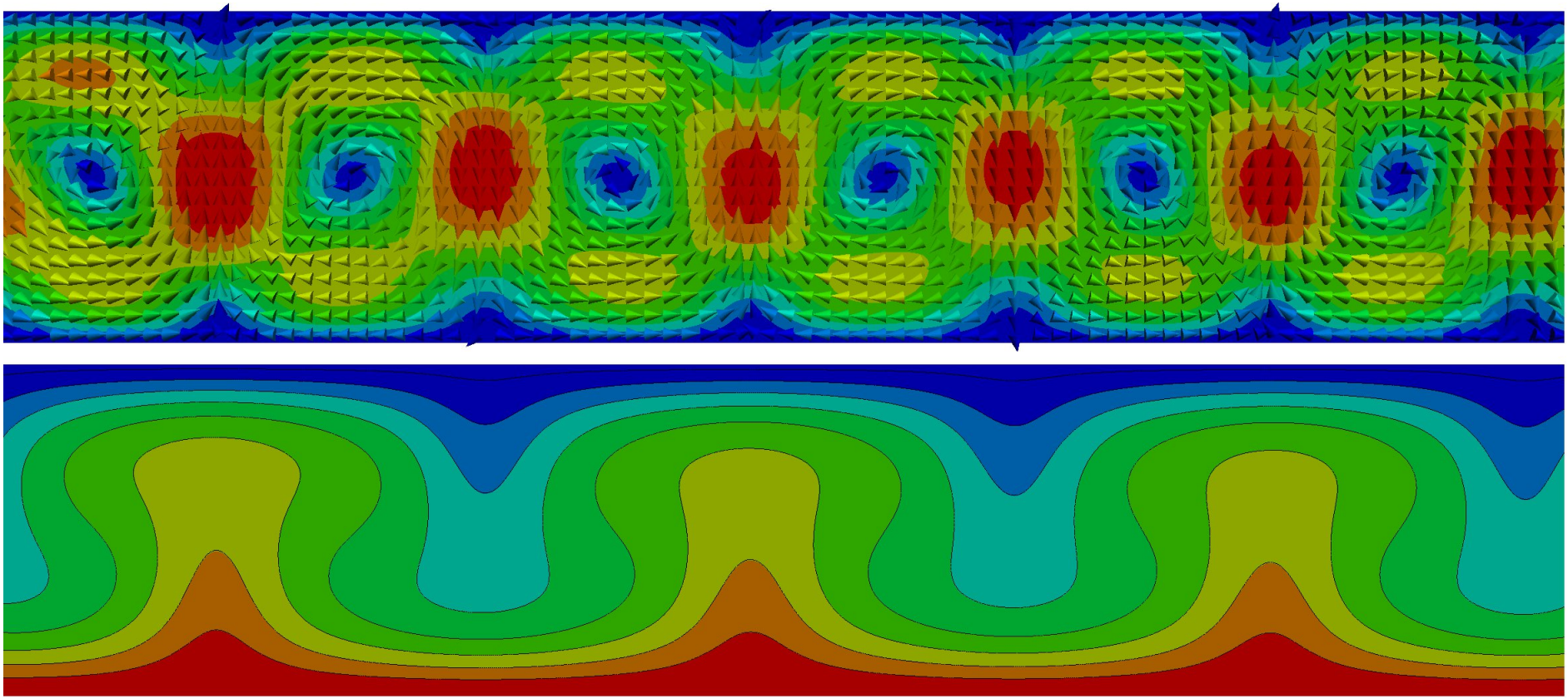
$$f = g \quad \rightarrow \quad (1 - \beta(T - T_0))g \qquad \beta: \text{ heat expansion coefficient}$$

Convection-Diffusion-Equation for the temperature

$$\frac{\partial T}{\partial t} + \operatorname{div}(-\lambda \nabla T + u \cdot T) = q$$

- Discretization of Convection Diffusion Equation with HDG method
- Weak coupling of unsteady Navier Stokes Equation and unsteady Convection Diffusion Equation

Benard-Rayleigh example:
Bottom temperature: constant $20°C$
Top temperature: constant $20.5°C$



students project of the lecture "Advanced Finite Elements Methods"

DG Solver for Maxwell TD

DG Solver for Time Domain Maxwells Equation in an isotropic, non-conducting medium

$$\begin{aligned} \varepsilon \frac{\partial E}{\partial t} &= \mathrm{curl} H \\ \nu \frac{\partial H}{\partial t} &= -\mathrm{curl} E \\ &+ \text{ boundary conditions} \end{aligned}$$

with a stabilized non-dissipative flux [Hesthaven, Warburton].

computational aspects

- full polynomials of degree $k+1$ for E_h
- full polynomials of degree k and additional facet unknowns for H_h
- **block-diagonal mass matrices**

For further optimization we

- don't assemble global matrices G_h or $-G_h^T$
- but implement the operations $M_\nu^{-1} G_h(E_h^k)$ and $-M_\varepsilon^{-1} G_h^T(H_h^k)$

The complexity of elementwise operations is dominated by

- evaluating gradients
- evaluating trace terms

For both we use non-standard techniques on **tetrahedra** elements leading to fast computations.

- We use the covariant transformation ($u(x) = F^{-T}\hat{u}(\hat{x})$) (*G. Cohen et al.* 2005) for the transformation from the reference element leading to geometry independent element matrices
- On the reference element we use an orthogonal basis which is computed in a tensor product form with Jacobi polynomials

This leads to several advantages

- diagonal mass matrices for uncurved elements (orthogonal polynomials + covariant transformation)
- gradients can be evaluated by recursive relations (Jacobi polynomials)
- traces can be computed efficiently (tensor product form)

Time stepping:

- energy-conserving 2nd order accurate local time stepping scheme
- Higher Order methods in time for conductive and lossy materials (also for PML) with Composition Methods

(both can be combined for higher order time stepping with local time steps)

- We use the covariant transformation ($u(x) = F^{-T}\hat{u}(\hat{x})$) (*G. Cohen et al.* 2005) for the transformation from the reference element leading to geometry independent element matrices
- On the reference element we use an orthogonal basis which is computed in a tensor product form with Jacobi polynomials

This leads to several advantages

- diagonal mass matrices for uncurved elements (orthogonal polynomials + covariant transformation)
- gradients can be evaluated by recursive relations (Jacobi polynomials)
- traces can be computed efficiently (tensor product form)

Time stepping:

- energy-conserving 2nd order accurate local time stepping scheme
- Higher Order methods in time for conductive and lossy materials (also for PML) with Composition Methods

(both can be combined for higher order time stepping with local time steps)

conclusions and ongoing work

Conclusion:
Netgen/NGSolve provides modern solvers for a lot of different applications
ongoing work:

- Netgen/NGSolve: ...
- Navier Stokes: weakly compressible flows
- Navier Stokes: turbulence models
- Maxwell DG TD: stability of local time stepping

Thank you for your attention!