An Introduction to Netgen and NGSolve
Session 1
Using Netgen/NGSolve

Christoph Lehrenfeld

Center for Computational Engineering Sciences (CCES)
RWTH Aachen University

University of British Columbia, November 9, 2009
content

1 What is Netgen? What is NGSolve?
   - What is Netgen?
   - What is NGSolve?

2 How to use Netgen/NGSolve
   - (0) Defining Geometries
   - (1 − 2) 2D Geometries
   - (3) 3D Geometries
   - (4) Meshing
   - (5) Solving a scalar PDE - The Input File
   - (6a) Preconditioners
   - (6b) Drawing Fluxes, Evaluating Bilinearforms
   - (6c) Error Estimators, Marking Strategy
   - (6d) hp-Refinement
   - (7) Poisson in 3D
   - (8) Mechanics
   - (9) Constrains (Mechanics)
   - (10) Maxwell
   - (11) NGSflow
What is Netgen?

- Netgen is an automatic 2D and 3D tetrahedral mesh generator
- Plane, surface, and volume mesh generator
- Input can be provided by simple ASCII files (csg - files), or imported from CAD programs via IGES, Step, or STL files
- Netgen generates essentially unstructured triangular/tetrahedral meshes
- 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
Some Meshes generated by Netgen

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

We will define our own simple examples for CSG files later.
What is NGSolve?

- NGSolve is a finite element library which can be linked to Netgen
- contains arbitrary order finite elements of all standard element geometries, scalar, vector-valued, hybrid DG finite element spaces
- Integrators for basic equations (heat flow, elasticity, Maxwell, ...)
- Iterative solvers, a posteriori error estimates, ...
- There are extension modules for different application areas (ngs-mech, ngs-flow, ...)
- NGSolve is open source based on the LGPL license
Netgen knows the geometry, and maintains the mesh, does visualization
NGSolve does the finite elements and linear algebra
Connection is via the mesh interface (NGSolve: class MeshAccess)
Application classes can be loaded as shared libraries into NGSolve
Simulation of Mechanical Deformation and Stresses

Elastic beam:

Shell structures:
Simulation of Electromagnetic Fields

A simple coil:

Described by
Maxwell’s equations

A transformer built by Siemens - EBG, Linz:
Incompressible Flows (ngsflow)

Flow around a disk, $Re = 100$, $5^{th}$-order elements:

Flow around a cylinder, $Re = 100$
How to use Netgen/NGSolve
Define the geometry

Supported Geometry formats are

- 2D geometries (the .in2d files)
- 3D constructive solid geometries (the .csg files)
- surface triangulations (.stl files)
- IGES and Step files, (requires the optional OpenCascade geometry kernel)
(1 – 2) Defining 2D Geometries
### points (splinecurves2dv2)

splinecurves2dv2

2

points

<table>
<thead>
<tr>
<th>#</th>
<th>x-coordinate</th>
<th>y-coordinate</th>
<th>local mesh size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-3</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.2</td>
<td>-0.5</td>
<td>-maxh=0.1</td>
</tr>
<tr>
<td>6</td>
<td>0.3</td>
<td>-0.5</td>
<td>-maxh=0.1</td>
</tr>
<tr>
<td>7</td>
<td>0.3</td>
<td>0.5</td>
<td>-maxh=0.1</td>
</tr>
<tr>
<td>8</td>
<td>0.2</td>
<td>0.5</td>
<td>-maxh=0.1</td>
</tr>
<tr>
<td>9</td>
<td>-0.3</td>
<td>-0.5</td>
<td>-maxh=0.1</td>
</tr>
<tr>
<td>10</td>
<td>-0.2</td>
<td>-0.5</td>
<td>-maxh=0.1</td>
</tr>
<tr>
<td>11</td>
<td>-0.2</td>
<td>0.5</td>
<td>-maxh=0.1</td>
</tr>
<tr>
<td>12</td>
<td>-0.3</td>
<td>0.5</td>
<td>-maxh=0.1</td>
</tr>
</tbody>
</table>
segments and materials

segments
#domain left and right of the segment
# number of points the segment consists of
# points of the segment
# boundary number
# local mesh size
1 0 2 1 2 -bc=1
1 0 2 2 3 -bc=1
1 0 2 3 4 -bc=1
1 0 2 4 1 -bc=1
0 1 2 5 6 -bc=2 -maxh=0.1
0 1 2 6 7 -bc=2 -maxh=0.1
0 1 2 7 8 -bc=2 -maxh=0.1
0 1 2 8 5 -bc=2 -maxh=0.1
0 1 2 9 10 -bc=3 -maxh=0.1
0 1 2 10 11 -bc=3 -maxh=0.1
0 1 2 11 12 -bc=3 -maxh=0.1
0 1 2 12 9 -bc=3 -maxh=0.1

materials
1 inner -maxh=0.5
#0 outer
You specify the geometry in an ASCII file. Example:

```
algebraic3d
solid cube = orthobrick (0, 0, 0; 1, 1, 1);
solid sp = sphere (1, 1, 1; 0.5);
solid main = cube and not sp;
tlo main;
```

lots of geometry examples in share/netgen
(4) Meshing

You have the following **basic** meshing options:

- mesh grading
- element order (for curved elements)
- minimal and maximal global mesh size
- (3D: Set Boundary Numbers)

For mesh refinement you can choose between:

- uniform refinement
- regular refinement towards faces, edges, points (prescribing local maxh, minh,..)
- anisotropic, hp-refinement (has to be set in the geometry file already):
  - hpref(2D)
  - singular edge/point(3D)  [singular edge cube sphere]
Find \( u \in H^1 \) s.t. \[
\int_\Omega \varepsilon \nabla u \cdot \nabla v \, dx + \int_{\partial \Omega} \alpha uv \, ds = \int_{\partial \Omega} g v \, dx \quad \forall v \in H^1
\]

geometry = capacitor.in2d
define coefficient coef_eps 1,
define coefficient coef_alpha 0, 1e5, 1e5,
define coefficient coef_g 0, 1e5, -1e5,

define fespace v -order=5 -h1ho
define gridfunction u -fespace=v
define bilinearform a -fespace=v -symmetric
laplace coef_eps
robin coef_alpha

define linearform f -fespace=v
neumann coef_g

numproc bvp np1 -bilinearform=a -linearform=f -gridfunction=u -solver=direct
The PDE input file defines the variational problem. You specify the finite element space, and then, the bilinear-form (left hand side), and the linear form (right hand side) are build up from elementary blocks called integrators. The table below lists the names of the integrators needed in the equation above:

<table>
<thead>
<tr>
<th>Integrator Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>laplace coef_eps</td>
<td>$\int_{\Omega} \varepsilon \nabla u \cdot \nabla v , dx$</td>
</tr>
<tr>
<td>robin coef_alpha</td>
<td>$\int_{\Gamma_R} \alpha uv , ds$</td>
</tr>
<tr>
<td>neumann coef_g</td>
<td>$\int_{\Gamma_N} gv , ds$</td>
</tr>
</tbody>
</table>

Note that the last term is handled by the neumann - integrator with coefficient $\alpha u_0$. You might miss Dirichlet boundary conditions. Indeed, NGSolve always approximates them by Robin b.c. with large conductivity $\alpha$.

**Remark:**
There are first implementations for dirichlet boundary conditions, but not everywhere, not for everything...
Loading geometry and mesh

The first two lines load the prepared geometry file and the mesh file.

```python
gameometry = ngsolve/pde_tutorial/square.in2d
mesh = ngsolve/pde_tutorial/square.vol
```

Necessary is only the mesh - file.
But for flexibility (generating new mesh) and refinement (curved geometries), it's always advisable to describe the geometry-file as well.
Coefficient functions

The list of numbers correspond to the sub-domains, if the integral is taken over the domain, or, to parts of the boundary, if the integral is taken over the boundary, respectively. The boundary splits in our case into 3 boundary parts. We specify Robin boundary conditions (with conductivity $\alpha = 10^5$) on $\Gamma_2$ to $\Gamma_2$, and an homogenous Neumann boundary condition on $\Gamma_1$.

```
define coefficient coef_eps
1,
define coefficient coef_alpha
0, 1e5, 1e5,
define coefficient coef_g
0, 1e5, -1e5,
```

**Attention:**
coefficients are not filled up with zeros automatically!

#boundaries / #domains $\leq$ #coefficient entries
Definition of the PDE

The next lines define the mathematical objects finite-element space, grid-function, bilinear-form and a linear-form.

```
define fespace v -order=5 -h1ho

define gridfunction u -fespace=v

#define bilinearform a -fespace=v -symmetric
laplace coef_eps
robin coef_alpha

#define linearform f -fespace=v
neumann coef_g
```

The last line (numproc) calls the solver for boundary value problems (bvp). Here, a direct solver is used to solve the linear system (SparseCholesky,Pardiso,Mumps).

```
numproc bvp np1 -bilinearform=a -linearform=f -gridfunction=u
-solver=direct
```
Finite Element Spaces

The definition

\texttt{define fespace <name> <flaglist>}

defines the finite element space \texttt{<name>}.  

Example:

\texttt{define fespace \textit{v} -order=2 -dim=3}

There are various classes of finite element spaces. Default are continuous, nodal-valued finite element spaces. The following define flags select the type of spaces:

<table>
<thead>
<tr>
<th>no flag</th>
<th>continuous nodal finite element space</th>
</tr>
</thead>
<tbody>
<tr>
<td>-hcurl</td>
<td>\textit{H(curl)} finite elements (Nedelec-type, edge elements)</td>
</tr>
<tr>
<td>-hdiv</td>
<td>\textit{H(div)} finite elements (Raviart-Thomas, face elements)</td>
</tr>
<tr>
<td>-12</td>
<td>non-continuous elements, element by element</td>
</tr>
<tr>
<td>-l2surf</td>
<td>element by element on surface</td>
</tr>
<tr>
<td>-h1ho</td>
<td>Arbitrary order continuous elements</td>
</tr>
<tr>
<td>-hcurlho</td>
<td>Arbitrary order \textit{H(curl)} elements</td>
</tr>
<tr>
<td>-hdivho</td>
<td>Arbitrary order \textit{H(div)} elements</td>
</tr>
<tr>
<td>-12ho</td>
<td>Arbitrary order non-continuous elements</td>
</tr>
</tbody>
</table>
The following flags specify the finite element spaces

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-order=&lt;num&gt;</td>
<td>Order of finite elements</td>
</tr>
<tr>
<td>-dim=&lt;num&gt;</td>
<td>Number of fields (copies of fe), 2 for 2D elasticity</td>
</tr>
<tr>
<td>-vec</td>
<td>-dim=spacedim</td>
</tr>
<tr>
<td>-tensor</td>
<td>-dim=spacedim*spacedim</td>
</tr>
<tr>
<td>-symtensor</td>
<td>-dim=spacedim*(spacedim+1) / 2, (sym. stress tensor)</td>
</tr>
<tr>
<td>-complex</td>
<td>complex valued fe-space</td>
</tr>
</tbody>
</table>

A compound fe-space combines several fe-spaces to a new one. Useful, e.g., for Reissner-Mindlin plate models containing the deflection w and two rotations beta:

```
fespace vw -order=2 fespace vbeta -order=1 fespace v -compound
-spaces=[vw,vbeta,vbeta]
```

The fespace maintains the degrees of freedom. On mesh refinement, the space provides the grid transfer operator (prolongation). High order fe spaces maintain a lowest-order fespace of the same type for preconditioning.
Enhancements (preconditioners)

There are different std. preconditioners available

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

multigrid -cycle=
- 0 Gauss-Seidel
- 1 V-Cycle Multigrid
- 2 W-Cycle Multigrid

numproc bvp np1 -bilinearform=a -linearform=f -gridfunction=u
  -preconditioner=c -maxsteps=1000

Add -test for eigenvalues of $C^{-1}A$
You may as well want to do static condensation:

\begin{verbatim}
define fespace v -order=5 -h1ho -eliminate_internal
define bilinearform a -fespace=v -symmetric
        -eliminate_internal -linearform=f
\end{verbatim}

But then you have to define the linearform before you define the bilinearform.
(6b) Enhancements (fluxes, evaluations)

- fluxes:

  numproc drawflux np2 -bilinearform=a -solution=u -label=flux

- evaluating bilinearforms of gridfunctions:

  define bilinearform aeval -fespace=v -symmetric -nonassemble
  laplace coef_eps
  numproc evaluate npeval -bilinearform=aeval -gridfunction=u
  -gridfunction2=u -filename=evalp.out
(6c) Enhancements (estimators, element marking)

- **error estimators:**
  
  ```
  define fespace verr -l2 -order=0
  define gridfunction err -fespace=verr
  numproc zzerrorestimator np3 -bilinearform=a -linearform=f
  -solution=u -error=err -minlevel=1 -filename=errp.out
  ```

- **refinement marker:**
  
  ```
  numproc markelements np5 -error=err -minlevel=1 -factor=0.9
  numproc drawflux df1 -marked
  ```

- **enable prolongation from coarse gridfunction to finer gridfunction:**
  
  ```
  define gridfunction u -fespace=v -nested
  ```

Mesh refinement in Netgen is done by the *marked edge bisection*
(6d) hp-Refinement

- mark singular points segments, etc. for hp-refinement
- set NGSolve constants
  
  ```
  define constant hpref = 2
  define constant hpref_geom_factor = 0.1
  ```

- adapt fespace for variableorder
  
  ```
  define fespace v -order=5 -variableorder -h1ho
  -eliminate_internal
  ```

- take a look at the order-distribution:
  
  ```
  numproc drawflux df2 -order
  ```
How to find out about details of NGSolve-Objects:

“Print Equations” lists all registered NGSolve-Objects such as:
integrators, FESpaces, preconditioners and numprocs
(7) 3D poisson

We can do the same also in 3D (take a look at the 3D example)
(8) Mechanics - Linear Elasticity (d5 Beam.pde)

\[
\begin{align*}
\text{div} (\sigma) &= f \\
\sigma &= D\varepsilon(u) \\
\sigma \cdot n &= g \text{ on } \Gamma_N
\end{align*}
\Rightarrow \int_\Omega D\varepsilon(u) : \varepsilon(v) \, dx + \alpha \int_{\Gamma_P} uv \, ds = \int_\Omega fv \, dx + \int_{\Gamma_N} gv \, ds
\]

define coefficient E 2.1E11, #elasticity module
define coefficient nu 0.2, #poisson ratio
define coefficient penalty 1e20, 0,
define coefficient coef_force_z 7e4,
define coefficient coef_surface_force_z 0, 1e5,
define fespace v -dim=3 -order=4 -eliminate_internal
define fespace vp -dim=6 -order=3
define gridfunction u -fespace=v
define gridfunction stress -fespace=vp

define linearform f -fespace=v
neumann coef_surface_force_z -comp=3

define bilinearform a -fespace=v -symmetric
-eliminate_internal -linearform=f
elasticity E nu
robin penalty -comp=1
robin penalty -comp=2
robin penalty -comp=3

define preconditioner c -type=multigrid -bilinearform=a

numproc bvp np1 -bilinearform=a -linearform=f -gridfunction=u -preconditioner=c -maxsteps=200

numproc calcflux np2 -bilinearform=a -solution=u -flux=stress -applyd
(9) constrainedbvp mechanics
So far we looked at numproc bvp, but you can as well add scalar constraints. 
these constrains are imposed by linearforms. 
For example restrict averaged movement of certain regions: \( \int_{\Gamma} u_j \, ds = 0 \)

```
define coefficient bearingbc1 1, 0, 0, 0, 0, 0, 0, 0,
define coefficient bearingbc2 0, 1, 0, 0, 0, 0, 0, 0,
define coefficient bearingbc3 0, 0, 1, 0, 0, 0, 0, 0,
```

```
define linearform cnsty1 -fespace=v
neumann bearingbc1 -comp=2
define linearform cnstz1 -fespace=v
neumann bearingbc1 -comp=3
define linearform cnsty2 -fespace=v
neumann bearingbc2 -comp=2
define linearform cnstz2 -fespace=v
neumann bearingbc2 -comp=3
define linearform cnsty3 -fespace=v
neumann bearingbc3 -comp=2
define linearform cnstz3 -fespace=v
neumann bearingbc3 -comp=3
```

```
umproc constrainedbvp np1 -bilinearform=a -linearform=f
-gridfunction=u -preconditioner=c -maxsteps=50
-constraints=[cnsty1,cnstz1,cnsty2,cnstz2,cnsty3,cnstz3]
```
(10) Maxwell (Example d7_coil.pde)

Calculation of a stationary magnet field

\[ \text{curl} \frac{1}{\mu} \text{curl} A + \sigma A = j \]

(where \( \sigma \) can be a small number (as regularization term)) boundary condition:

- (penalty-approximated) dirichlet b.c. on boundary 3 (for tangential component)
- neumann b.c. on all other boundaries

The PDE-file (Part 1:)

define constant geometryorder = 4
define coefficient nu 1, 1,
define coefficient sigma 1e-6, 1e-6,
define coefficient cs1 ( y ), 0,
define coefficient cs2 ( -x ), 0,
define coefficient cs3 0, 0,
define coefficient penalty 0, 0, 1e8, 0, 0, 0
weak form: \[ \int_{\Omega} \frac{1}{\mu} \text{curl} A \text{curl} \; v \; dx + \int_{\Omega} \sigma A v \; dx + \int_{\Gamma_3} \text{pen} A v \; ds = \int_{\Omega} f v \; dx \]

The PDE-file (Part 2:)

define fespace v -hcurlho -order=4 -eliminate_internal -nograds

define gridfunction u -fespace=v -novisual

define linearform f -fespace=v
  sourceedge cs1 cs2 cs3 -definedon=1

define bilinearform a -fespace=v -symmetric -linearform=f -eliminate_internal
  curlcurledge nu
  massedge sigma -order=2
  robinedge penalty

define preconditioner c -type=multigrid -bilinearform=a -cylce=1 -smoother=block
  -blocktype=1 -coarsetype=direct -coarsesmoothingsteps=5 -notest

numproc bvp np1 -bilinearform=a -linearform=f -gridfunction=u
  -preconditioner=c -prec=1.e-9
Magnetic field visualization:

New options in this pde-file:

- "geometrieorder"-flag: curve elements up to given order
- "blocktype": there are different possibilities for the preconditioners you may want to use
- "nograds": see next slide
“nograds”-flag - The de Rham diagram for continuous and discrete function spaces

\[
\begin{align*}
H^1 & \xrightarrow{\nabla} H(\text{curl}) \xrightarrow{\text{Curl}} H(\text{div}) \xrightarrow{\text{Div}} L^2 \\
U & \xrightarrow{} U \\
W_h & \xrightarrow{\nabla} V_h \xrightarrow{\text{Curl}} Q_h \xrightarrow{\text{Div}} S_h
\end{align*}
\]

Used for constructing high order finite elements [J. Schöberl + S. Zaglmayr, ’05, Thesis Zaglmayr ’06]

\[
\begin{align*}
W_{hp} & = W_{L^1} + \text{span}\{\varphi_{h.o.}^W\} \\
V_{hp} & = V_{N_0} + \text{span}\{\nabla \varphi_{h.o.}^W\} + \text{span}\{\varphi_{h.o.}^V\} \quad \text{“nograds” kicks red part out} \\
Q_{hp} & = Q_{RT_0} + \text{span}\{\text{curl} \varphi_{h.o.}^V\} + \text{span}\{\varphi_{h.o.}^Q\} \\
S_{hp} & = S_{P_0} + \text{span}\{\text{div} \varphi_{h.o.}^Q\}
\end{align*}
\]
Within the NGSflow-package we have got numprocs which care for the FESpace, Bilinearforms, etc. on their own. So you just have to prescribe

- geometry
- mesh
- boundary conditions
- initial conditions (or stokes)
- time step
- time interval
define constant heapsize = 1000000

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

game geometry = collision.in2d
mesh = collision.vol

shared = libnavierstokes_hdivred

define constant geometryorder = 4

define coefficient inflow
((x>-1.01)*(x<1.01)*(1+x)*(1-x)+(y>-1.01)*(y<1.01)*(1+y)*(1-y)),0,0,0,0,0,

numproc navierstokeshdiv np1 -order=4 -tau=1e-2 -tend=20 -alpha=30 -nu=2e-3
-umax=1 -wallbnds=[1,3] -slipbnds=[]

numproc visualization vis -scalarfunction=u.0 -subdivision=3 -maxval=1.5
Thank you for your attention!