GetDP: a General Finite-Element Solver for the de Rham Complex

C. Geuzaine, Université de Liège

July 18, 2007

Joint work with P. Dular

History

- Started at the end of 1996
- First feature-complete public release (binary-only): mid-1998
- Open-sourced under GNU GPL in 2004

Design:

- Small, fast, no GUI
- For (sophisticated) end-users: not yet another library
- Limit external dependencies to a minimum

Original Goal

Software environment open to various couplings

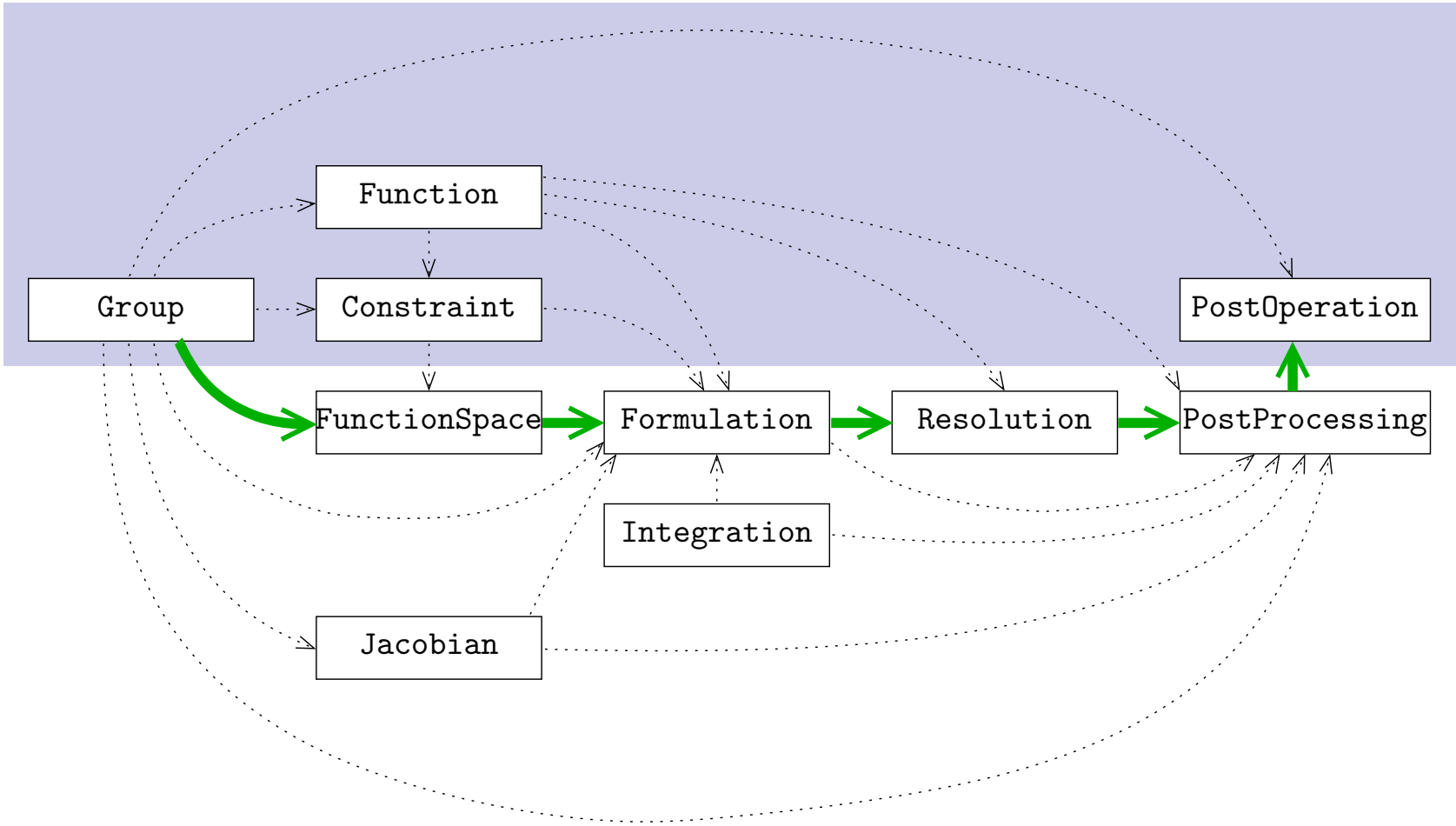
- *Physical* problems (electromagnetic, thermal, mechanical, ...)
- *Numerical* methods (finite element methods, integral methods, ...)
- *Geometries* (1D, 2D, 3D)
- *Time* states (static, harmonic, transient, eigen values)

How?

- Clear *mathematical structure*
- Directly transcribed into *10 objects* in *text data file*

Definition of Discrete Problems

10 objects defined in *text data files* ("`.pro` files")



(Top: particular to a given problem. Bottom: particular to a method of resolution)

— INTERLUDE —

If you prefer to play with the code instead of listening to me:

http://geuz.org/gmsh
(3-D CAD, mesh and post-processing)

http://geuz.org/getdp
(the solver—THIS TALK)

`Group`: Topological Entities

- Regions
- "Functions" on regions (nodes, edges, edges of tree, dual faces, ...)

```
Group{
  Air = Region[1]; //elementary group (linked with the mesh)
  Core = Region[2];
  Gamma = Region[{3,4}];

  Omega = Region[{Air, Core}]; //combining elementary groups

  nodes = NodesOf[Omega]; //function group
  edgesOfSpanningTree = EdgesOfTreeIn[Omega, StartinOn Gamma];
}
```

Function: Functional Expressions

- Piecewise definitions
- Space-time dependent
- Physical characteristics, sources, constraints, ...

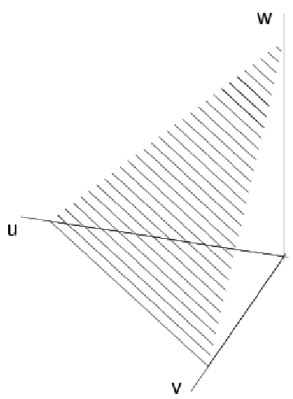
```
Function{
  f = 50; //constants
  mu0 = 4.e-7 * Pi;

  mu[Air] = mu0; //piecewise definition
  mu[Core] = mu0 + 1/(100 + 100 * ($1)^6); //argument ($1)

  TimeFct[] = Cos[2*Pi*f*$Time] * Exp[-$Time/0.01]; //current value
}
```

`FunctionSpace`: Discrete Function Spaces

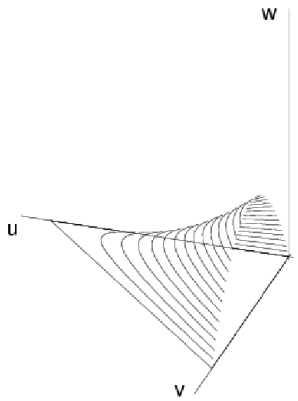
- Basis functions (associated with nodes, edges, faces, ...) of various orders
- Coupling of fields and potentials
- Definition of global quantities (fluxes, circulations, ...)
- Essential constraints (boundary and gauge conditions, ...)

```
FunctionSpace{
  { Name H1; Type Form0; //discrete function space for H1_h
    BasisFunction {
      { Name wi; NameOfCoef fi; Function BF_Node; //``P1 finite elements''
        Support Omega; Entity NodesOf[All]; }
    }
    Constraint {
      { NameOfCoef fi; EntityType NodesOf; NameOfConstraint Dirichlet; }
    }
  }
}
```

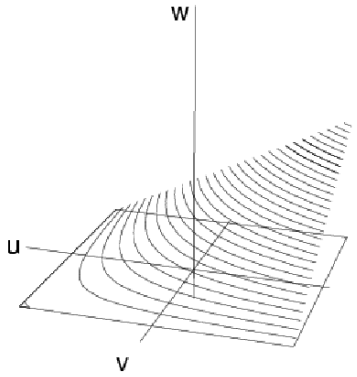
```
//higher-order version
FunctionSpace{
  { Name H1; Type Form0;
    BasisFunction {
      { Name wi; NameOfCoef fi; Function BF_Node; //order 1
        Support Omega; Entity NodesOf[All]; }
      { Name wi2; NameOfCoef fi2; Function BF_Node_2E; //order 2
        Support Omega; Entity EdgesOf[All]; }
    }
    Constraint {
      { NameOfCoef fi; EntityType NodesOf; NameOfConstraint Dirichlet; }
      { NameOfCoef fi2; EntityType EdgesOf; NameOfConstraint Dirichlet2; }
    }
  }
}
```



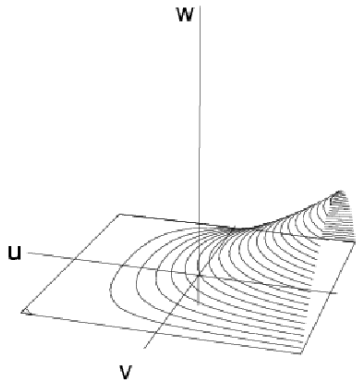
a. $p0$



d. $p3$



a. $p0$



e. $p4$

de Rham Complex

$$H_h^1(\Omega) \overset{\mathbf{grad}_h}{\longrightarrow} \boldsymbol{H}_h(\mathbf{curl};\Omega) \overset{\mathbf{curl}_h}{\longrightarrow} \boldsymbol{H}_h(\mathrm{div};\Omega) \overset{\mathrm{div}_h}{\longrightarrow} L^2(\Omega)$$

Exact sequence preserved at the discrete level using Whitney elements.

Example for $\boldsymbol{H}_h(\mathbf{curl};\Omega)$:

```
FunctionSpace {
  { Name Hcurl_h; Type Form1; //discrete Hcurl_h
    BasisFunction {
      { Name se; NameOfCoef he; Function BF_Edge;
        Support Omega; Entity EdgesOf[All]; }
    }
    Constraint {
      { NameOfCoef he; EntityType EdgesOf; NameOfConstraint Dirichlet; }
    }
  }
}
```

$$\mathbf{h} = \sum_{e \in \mathcal{E}(\Omega)} h_e \mathbf{s}_e \quad \mathbf{h} \in W^1(\Omega)$$

Coupled Field-Potential

$$\mathbf{h} = \sum_{e \in \mathcal{E}(\Omega_c)} h_e \mathbf{s}_e + \sum_{n \in \mathcal{N}(\Omega - \Omega_c)} \phi_n \mathbf{v}_n \quad \mathbf{h} \in W^1(\Omega)$$

```
FunctionSpace {
  { Name Hcurl_hphi; Type Form1;
    BasisFunction {
      { Name se; NameOfCoef he; Function BF_Edge;
        Support OmegaC; Entity EdgesOf[All, Not SkinOmegaC]; }
      { Name vn; NameOfCoef phin; Function BF_GradNode;
        Support OmegaCC; Entity NodesOf[All]; }
      { Name vn; NameOfCoef phic; Function BF_GroupOfEdges;
        Support OmegaC; Entity GroupsOfEdgesOnNodesOf[SkinOmegaC];}
    }
    Constraint {
      { NameOfCoef he; EntityType EdgesOf; NameOfConstraint h; }
      { NameOfCoef phin; EntityType NodesOf; NameOfConstraint phi; }
      { NameOfCoef phic; EntityType NodesOf; NameOfConstraint phi; }
    }
  }
}
```

Topologically Non-Trivial Domains

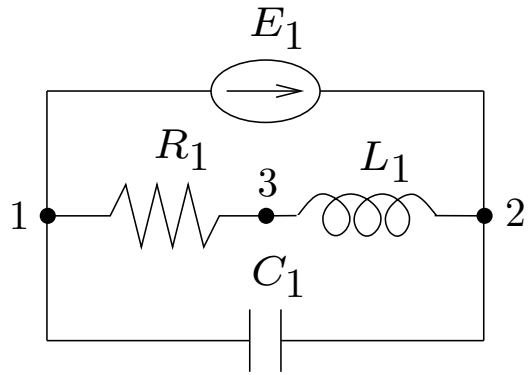
```
FunctionSpace {
  { Name Hcurl_hphi; Type Form1;
    BasisFunction {
      ...//same as above
      { Name sc; NameOfCoef Ic; Function BF_GradGroupOfNodes;
        Support ElementsOf[DomainCC, OnOneSideOf SurfaceCut];
        Entity GroupsOfNodesOf[SurfaceCut]; }
      { Name sc; NameOfCoef Icc; Function BF_GroupOfEdges;
        Support DomainC; Entity GroupsOfEdgesOf[SurfaceCut,
           InSupport ElementsOf[SkinDomainC, OnOneSideOf SurfaceCut]]; }
    }
    GlobalQuantity {
      { Name I; Type AliasOf       ; NameOfCoef Ic; }
      { Name U; Type AssociatedWith; NameOfCoef Ic; }
    }
    Constraint {
      ...//same as above
      { NameOfCoef Ic; EntityType GroupsOfNodesOf; NameOfConstraint I; }
      { NameOfCoef Icc; EntityType GroupsOfNodesOf; NameOfConstraint I; }
      { NameOfCoef U; EntityType GroupsOfNodesOf; NameOfConstraint V; }
    }
  }
```

`Constraint`: Constraints on Function Spaces

- Boundary conditions (classical, periodic, etc.)
- Initial conditions
- Topology of circuits with lumped elements
- Other constraints (on local and global quantities)

```
Constraint{
  { Name Dirichlet; Type Assign; //boundary conditions
    Case {
      { Region Surface0; Value 0; }
      { Region Surface1; Value 1; }
    }
  }
}
```

```
Constraint{
  //time-dependent or harmonic constraints
  { Name Current; Type Assign;
    Case {
      { Region CurrentLoop; Value 1000; TimeFunction TimeFct[]; }
    }
  }

  //network relations between global quantities
  { Name ElectricalCircuit; Type Network;
    Case Circuit {
      { Region E1; Branch {1,2}; }
      { Region R1; Branch {1,3}; }
      { Region L1; Branch {3,2}; }
      { Region C1; Branch {1,2}; }
    }
  }
}
```



`Formulation`: Equation builder

- Various formulation types: Galerkin finite elements, collocation, boundary elements, ...
- Symbolic expression of equations
- Involves local, global and integral quantities based on function spaces

```
Formulation{
  { Name Maxwell_e; Type FemEquation;
    Quantity {
      { Name e; Type Local; NameOfSpace Hcurl_h; }
    }
    Equation {
      Galerkin { [ 1/mu[] * Dof{Curl e} , {Curl e} ];
                 In Omega; Jacobian Jac1; Integration Int1; }
      Galerkin { DtDt [ epsilon[] * Dof{e} , {e} ];
                 In Omega; Jacobian Jac1; Integration Int1; }
    }
  }
}
```

"Find $\boldsymbol{e} \in \boldsymbol{H}_h(\mathbf{curl}; \Omega)$ such that
$(\mu^{-1}\mathbf{curl}\,\boldsymbol{e}, \mathbf{curl}\,\boldsymbol{e}') + \partial_t^2(\epsilon\boldsymbol{e}, \boldsymbol{e}') = 0, \quad \forall \boldsymbol{e}' \in \boldsymbol{H}_h(\mathbf{curl}; \Omega)$"

```
Formulation { //handle complexity with loops, etc.
  { Name OSRC; Type FemEquation;
    Quantity {
      { Name psi; Type Local; NameOfSpace Hdiv_psi; }
      { Name w; Type Local; NameOfSpace Hdiv_w; }
      For j In{1:N}
      { Name phi~{j}; Type Local; NameOfSpace Hdiv_phi~{j}; }
      EndFor
      { Name nxh; Type Local; NameOfSpace Hdiv_nxh; }
    }
    Equation {
      Galerkin { [ Z0 * OSRC_C0[]{N,theta_branch} * Dof{nxh} , {nxh} ];
                 In Gama; Jacobian JSur; Integration I1; }
      Galerkin { [ -{psi} , {nxh} ];
                 In Gama; Jacobian JSur; Integration I1; }
      For j In{1:N}
        Galerkin { [ Z0 * OSRC_Aj[]{j,N,theta_branch} * Dof{phi~{j}} , {nxh} ];
                   In Gama; Jacobian JSur; Integration I1; }
        Galerkin { [ Dof{phi~{j}} , {phi~{j}} ];
                   In Gama; Jacobian JSur; Integration I1; }
        Galerkin { [ -OSRC_Bj[]{j,N,theta_branch} / keps[]^2 * Dof{d phi~{j}} , {d phi~{j}} ];
                   In Gama; Jacobian JSur; Integration I1; }
        Galerkin { [ 1 / keps[]^2 * Dof{d nxh} , {d phi~{j}} ];
                   In Gama; Jacobian JSur; Integration I1; }
      EndFor
    }
  }
}
```

Jacobian: Mappings

- Mapping from reference to real space
- Geometrical transformations (axisymmetry, infinite domains, PML, ...)

```
Jacobian{
  { Name Jac1;
    Case { //piecewise defined on groups
      { Region OmegaInf; Jacobian VolSphShell{Rint, Rext}; }
      { Region OmegaAxi; Jacobian VolAxi; }
      { Region All; Jacobian Vol; }
    }
  }
}
```

Integration: Integration Methods

- Various numeric and analytic integration methods
- Criterion-based selection

```
Integration {
  { Name Int1; Criterion Test[];
    Case {
      { Type Gauss;
        Case {
          { GeoElement Triangle; NumberOfPoints 3; }
          { GeoElement Tetrahedron; NumberOfPoints 3; }
        }
      }
      { Type Analytic; }
    }
  }
}
```

`Resolution`: Solver

- Description of a sequence of operations
- Time stepping, nonlinear iterations, ...
- Coupled problems (e.g. magneto-thermal coupling)
- Link various resolution steps (e.g. pre-computation of source fields)

```
Resolution{
  { Name Parabolic;
     System {
       { Name A; NameOfFormulation Parabolic; }
     }
     Operation{
       InitSolution[A];
       TimeLoopTheta[tmin,tmax,dt,1]{
         Generate[A]; Solve[A]; If[Save[]]{ SaveSolution[A]; }
       }
     }
  }
```

PostProcessing: Quantities of Interest

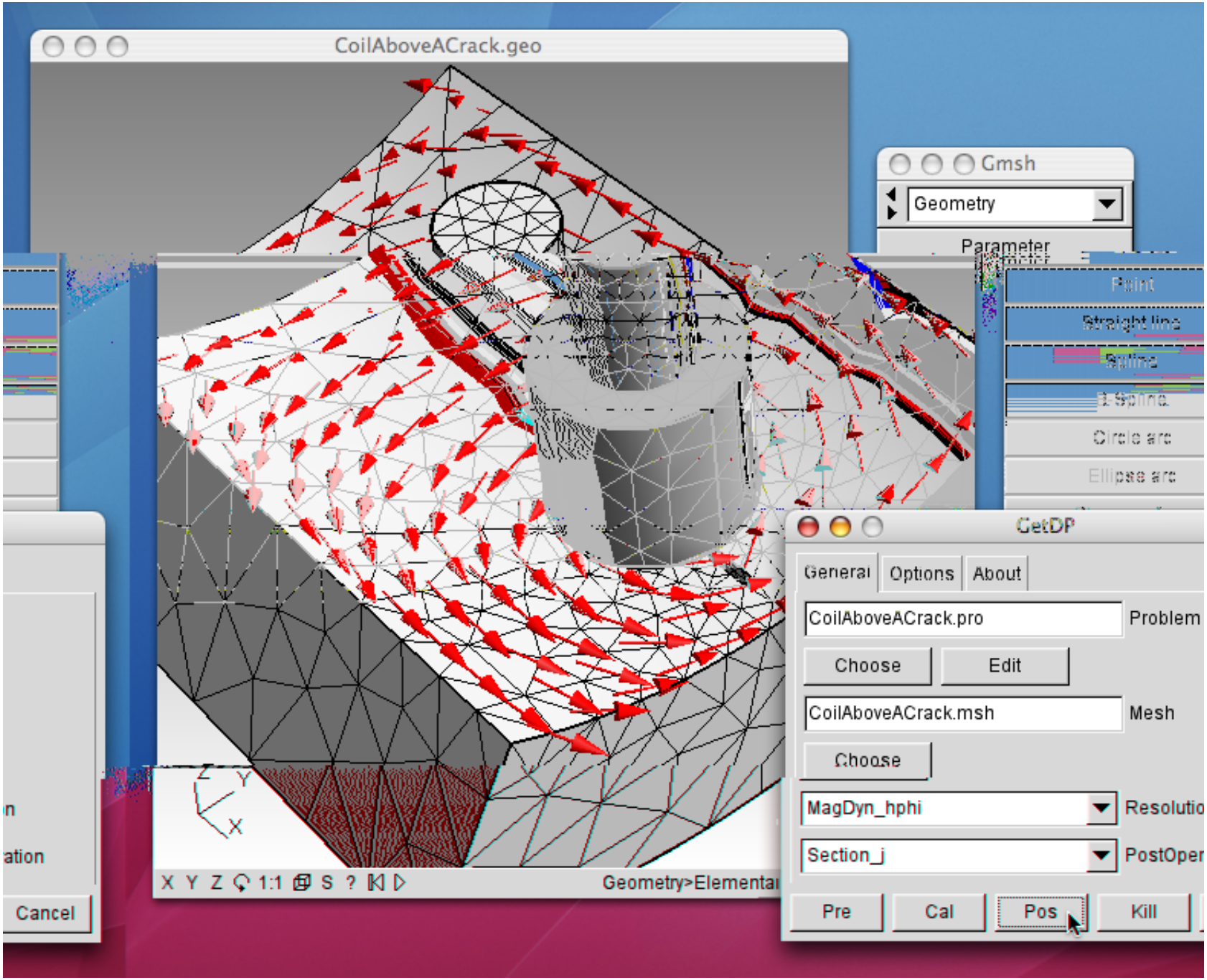
- "Front-end" to computational data
- Piecewise definition of any quantity of interest
- Local or integral evaluation

```
PostProcessing {
  { Name magfields; NameOfFormulation Dynamic;
    Quantity {
      { Name b;
         Value {
           Local { [ -mu[] * {Grad phi} ]; In OmegaCC; }
           Local { [ mu[] * h ]; In OmegaC; }
         }
      }
   }
}
```

PostOperation: Export

- Evaluation of post-processing quantities (e.g. maps, sections, local or global evaluation, ...)
- Operations on post-processing quantities (sorting, smoothing, adaptation, ...)
- Various output formats (e.g. space or time oriented, text, binary, ...)

```
PostOperation {
  { Name Map_b; NameOfPostProcessing magfields;
     Operation {
       Print[ b, OnElementsOf Omega, File "b.pos", Format Gmsh ];
       Print[ b, OnLine {{0,0,0}{1,0,0}} {100}, File "b.txt" ];
     }
  }
}
```


CoilAboveACrack.geo
Gmsh
Geometry
Parameter
Point
Straight line
Spline
Circle arc
Ellipse arc
GetDP
General
Options
About
CoilAboveACrack.pro
Problem
Choose
Edit
CoilAboveACrack.msh
Mesh
Choose
MagDyn_hphi
Resolutio
Section_j
PostOper
Pre
Cal
Pos
Kill
Cancel
X Y Z 1:1 S ?
Geometry>Elementar

Technical Details

- Written in C
- Language parser using Lex/Yacc
- Linear algebra: Sparskit2 or PETSc
- (GPL version depends on GSL)

Performance?

IMHO, the only limitation on interesting problems is the solver.

More info: http://www.montefiore.ulg.ac.be/~geuzaine

(or simply http://geuz.org)