Gmsh, GetDP & ONELAB

C. Geuzaine

Université de Liège

March 23, 2022

Some background

An overview of Gmsh
- Introduction
- Geometry module
- Mesh module
- Solver module
- Post-processing module
- Recent developments

An overview of GetDP
- Introduction
- The 10 GetDP objects
- Recent developments

Tying it all together with ONELAB

Conclusions and perspectives

Some background

- I am a professor at the University of Liège in Belgium, where I lead a team of about 15 people in the Montefiore Institute (EECS Dept.), at the intersection of applied math, scientific computing and engineering physics
- Our research interests include modeling, analysis, algorithm development, and simulation for problems arising in various areas of engineering and science
- Current applications: low- and high-frequency electromagnetics, geophysics, biomedical problems
- We write quite a lot of codes, some released as open source software: https://gmsh.info, https://getdp.info, https://onelab.info

Gmsh, GetDP & ONELAB

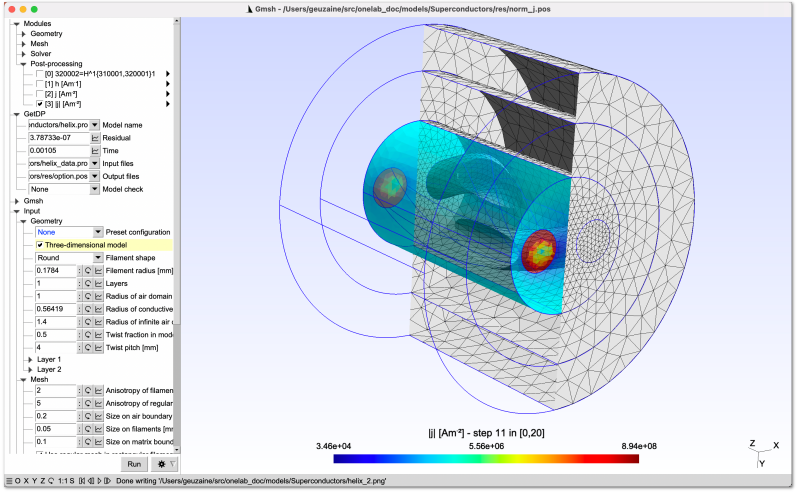
- Gmsh (https://gmsh.info) is a 3D finite element mesh generator with a built-in CAD engine and post-processor
  - Joint work with J.-F. Remacle at UCLouvain, with important contributions from J. Lambrechts, K. Hillewaert, M. Pellikka, A. Johnen, H. Si, A. Royer, C. Marot, I. Badia, T. Toulorge, M. Reberol, ...
- GetDP (https://getdp.info) is a general finite element solver using mixed finite elements
  - Joint work with P. Dular at ULiège, with important contributions from J. Gyselinck, R. Sabariego, M. Asam, B. Thierry, K. Jacques, F. Henrotte, G. Demésy, ...
- ONELAB (https://onelab.info) is an abstract interface for sharing information between codes

Some numbers

Today, Gmsh, GetDP and ONELAB represent about 500k lines of C++ code

- still only 3 core developers; about 100 with ≥ 1 commit
- about 1,300 registered users on the development site `https://gitlab.onelab.info`
- about 20,000 downloads per month (70% Windows)
- about 800 citations per year – the Gmsh paper is cited about 5000 times
- Gmsh has probably become one of the most popular (open source) finite element mesh generators?

Short demo



Download the ONELAB software bundle from https://onelab.info

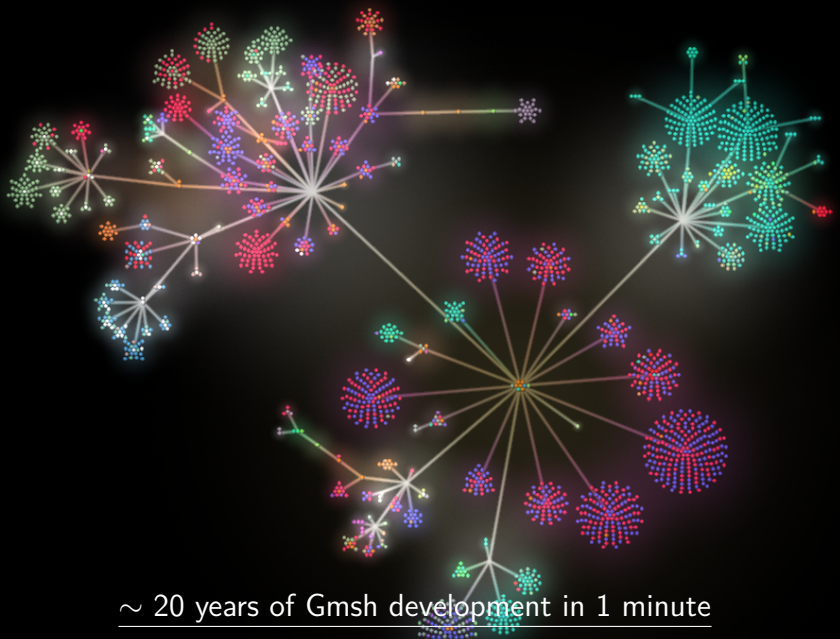
Open models/Superconductors/helix.pro with Gmsh

LIÈGE université

An overview of Gmsh

What is Gmsh?

- Gmsh (https://gmsh.info) is a 3D finite element mesh generator with a built-in CAD engine and post-processor
- Open source (GNU GPL v2+)
- Includes a graphical user interface (GUI)
- Can be used as a standalone software or as a library (C++, C, Python or Julia)
- Can drive any simulation code interactively through ONELAB



$\sim$ 20 years of Gmsh development in 1 minute

A warm thank you to all the contributors!

LIÈGE université

Gmsh - A little bit of history

- Gmsh was started in 1996, as a side project
- 1998: First public release
- 2003: Open sourced under GNU GPL
- 2006: OpenCASCADE integration (Gmsh 2)
- 2009: IJNME paper and switch to CMake
- 2012: Curvilinear meshing and quad meshing
- 2013: Homology and ONELAB solver interface
- 2015: Multi-threaded 1D and 2D meshing (coarse-grained)
- 2017: Boolean operations and switch to Git (Gmsh 3)
- 2018: C++, C, Python and Julia API (Gmsh 4)
- 2019: Multi-threaded 3D meshing (fine-grained), robust STL remeshing
- 2021: GmshFEM, Quasi-structured quad meshing

Gmsh - Strategic choices

- Design goals: fast, light and user-friendly
  - Written in simple C++
  - GUIs: FLTK (desktop), UIKit (iOS), Android
  - OpenGL graphics
  - Highly portable (OSes & compilers)
  - Easy to distribute & install: zero dependencies on installation
- Handling of numerous third party libraries
  - Build system based on CMake – everything is optional
  - Some libs integrated and redistributed directly in gmsh/contrib (HXT, BAMG, Metis, Concorde, ...)
- Funding
  - Hobby until 2006, then industry, Wallonia, Belgium & EU

Gmsh - Strategic choices

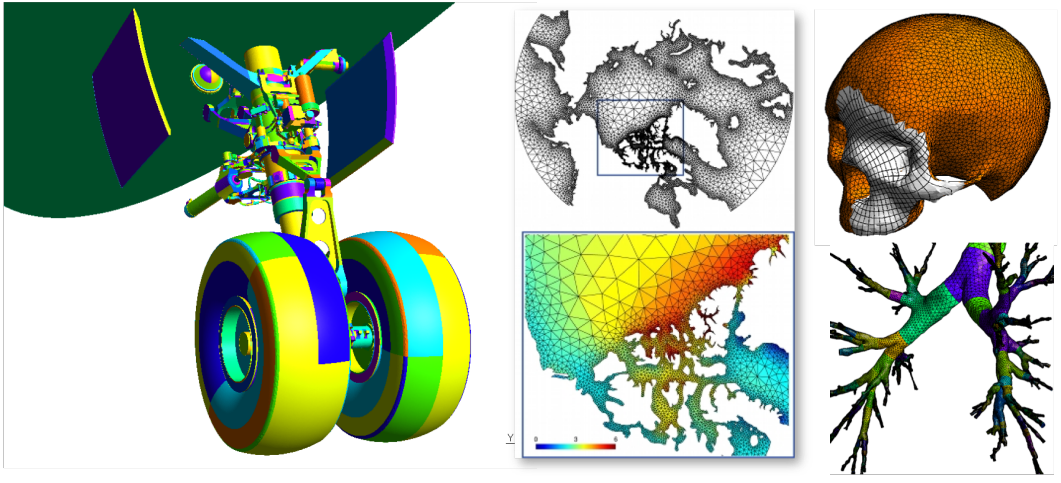
- Community infrastructure
  - Our own (using GitLab) to enable public/private parts (https://gitlab.onelab.info/gmsh/gmsh)
  - Continuous integration and delivery (CI/CD) of Gmsh app and Gmsh SDK on Windows, Linux and macOS
  - Web site (https://gmsh.info) with documentation, tutorials, etc.
  - Scientific aspects of algorithms detailed in journal papers
- Licensing
  - Gmsh is distributed under the GNU General Public License v2 or later, with exceptions to allow for easier linking with external libraries
  - We double-license to enable embedding in commercial codes

Gmsh - Basic concepts

- Gmsh is based around four modules: Geometry, Mesh, Solver and Post-processing
- Gmsh can be used at 3 levels
  - Through the GUI
  - Through the dedicated `.geo` language
  - Through the C++, C, Python and Julia API
- Main characteristics
  - All algorithms are written in terms of abstract model entities, using a Boundary REPresentation (BREP) approach
  - Gmsh never translates from one CAD format to another; it directly accesses each CAD kernel API (OpenCASCADE, Built-in, ...)

Gmsh - Basic concepts

The goal is to deal with very different underlying data representations in a transparent manner



Gmsh - Geometry module

Under the hood, 4 types of model entities are defined:

1. Model points G_i^0 that are topological entities of dimension 0
2. Model curves G_i^1 that are topological entities of dimension 1
3. Model surfaces G_i^2 that are topological entities of dimension 2
4. Model volumes G_i^3 that are topological entities of dimension 3

Gmsh - Geometry module

- Model entities are topological entities, i.e., they only deal with adjacencies in the model; a bi-directional data structure represents the graph of adjacencies

$$G_i^0 \rightleftharpoons G_i^1 \rightleftharpoons G_i^2 \rightleftharpoons G_i^3$$

- Any model is able to build its list of adjacencies of any dimension using local operations
- The BRep is extended with non-manifold features: adjacent entities, and *embedded* (internal) entities
- Model entities can be either CAD entities (e.g. from the built-in or OpenCASCADE kernel) or *discrete* entities (defined by a mesh, e.g. STL)

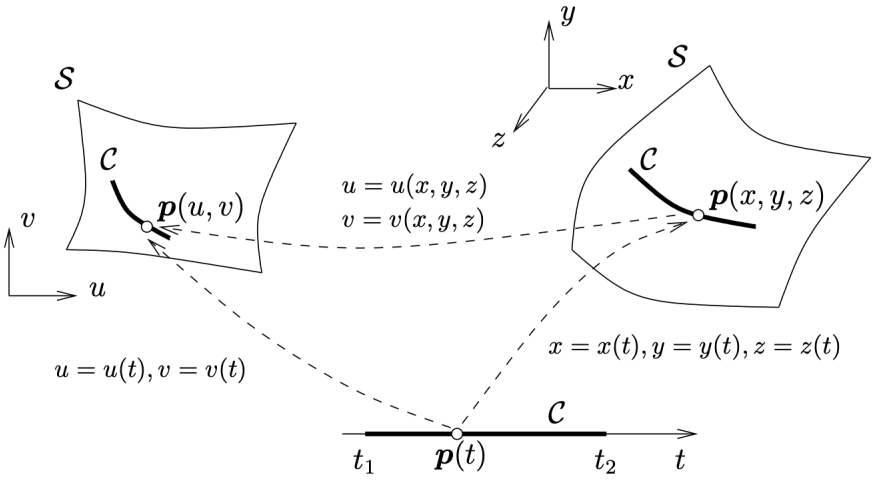
Gmsh - Geometry module

The geometry of a CAD model entity depends on the solid modeler kernel for its underlying representation. Solid modelers usually provide a parametrization of the shapes, i.e., a mapping:

$$\mathbf{p} \in R^d \mapsto \mathbf{x} \in R^3$$

1. The geometry of a model point G_i^0 is simply its 3-D location $\mathbf{x}_i = (x_i, y_i, z_i)$
2. The geometry of a model curve G_i^1 is its underlying curve $\mathcal{C}_i$ with its parametrization $\mathbf{p}(t) \in \mathcal{C}_i$, $t \in [t_1, t_2]$
3. The geometry of a model surface G_i^2 is its underlying surface $\mathcal{S}_i$ with its parametrization $\mathbf{p}(u, v) \in \mathcal{S}_i$
4. The geometry associated to a model volume is R^3

Gmsh - Geometry module

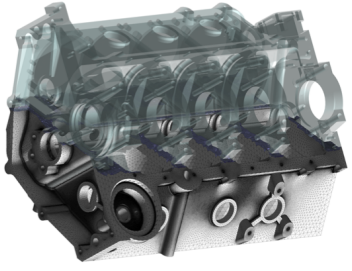


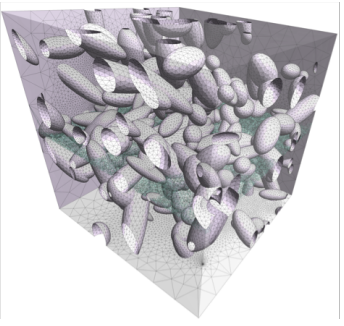
Point $\mathbf{p}$ located on a curve $\mathcal{C}$ that is itself embedded in a surface $\mathcal{S}$

Gmsh - Geometry module

Operations on CAD model entities are performed directly within their respective CAD kernels:

- There is no common internal geometrical representation
- Rather, Gmsh directly performs the operations (translation, rotation, intersection, union, fragments, ...) on the native geometrical representation using each CAD kernel's own API

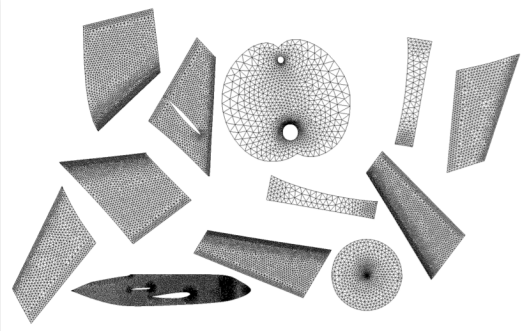


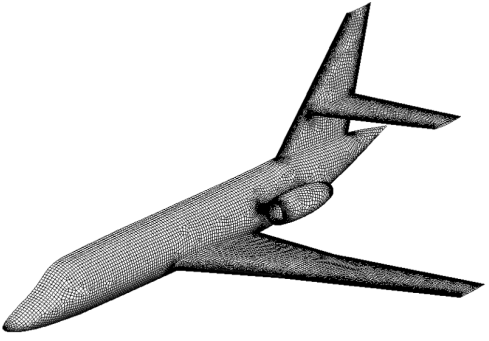


Gmsh - Geometry module

Discrete model entities are defined by a mesh (e.g. STL):

- They can be equipped with a geometry through a *reparametrization* procedure
- The parametrization is then used for meshing, in exactly the same way as for CAD entities

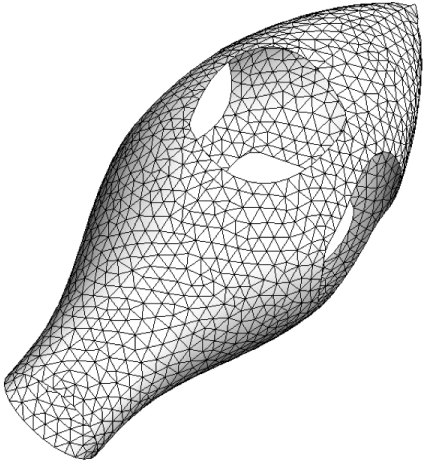


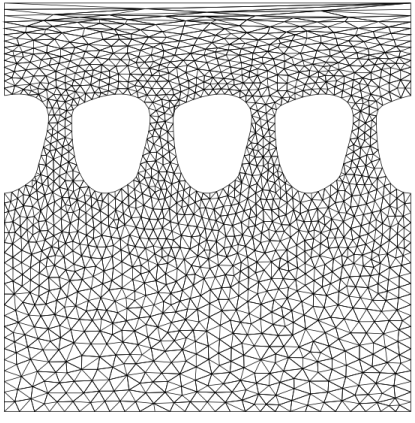


Gmsh - Mesh module

- Gmsh implements several meshing algorithms with specific characteristics
  - 1D, 2D and 3D
  - Structured, unstructured and hybrid
  - Isotropic and anisotropic
  - Straight-sided and curved
  - From standard CAD data or from STL through reparametrization
- Built-in interfaces to external mesh generators (BAMG, MMG3D, Netgen)

Gmsh - Mesh module





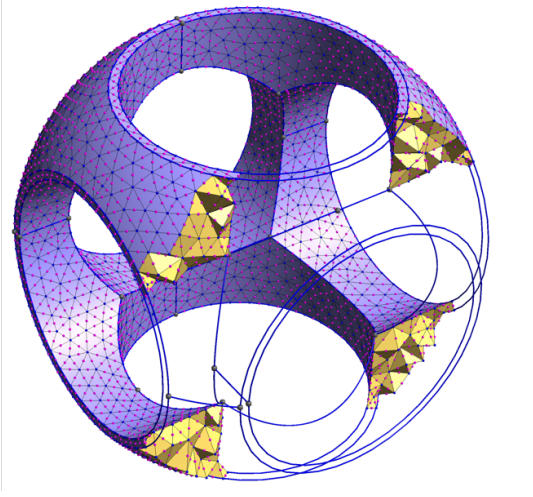
Typical CAD kernel idiosyncrasies: seam edges and degenerated edges

Gmsh - Mesh module

- Mesh data is made of *elements* (points, lines, triangles, quadrangles, tetrahedra, hexahedra, ...) defined by an ordered list of their *nodes*
- Elements and nodes are stored (*classified*) in the model entity they discretize:
  - A model point will thus contain a mesh element of type point, as well as a mesh node
  - A model curve will contain line elements as well as its interior nodes, while its boundary nodes will be stored in the bounding model points
  - A model surface will contain triangular and/or quadrangular elements and all the nodes not classified on its boundary or on its embedded entities (curves and points)
  - A model volume will contain tetrahedra, hexahedra, etc. and all the nodes not classified on its boundary or on its embedded entities (surfaces, curves and points)

Gmsh - Mesh module

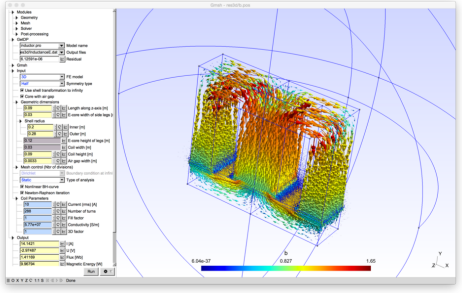
This mesh data structure allows to easily and efficiently handle the creation, modification and destruction of conformal finite element meshes



Gmsh - Solver module

- Gmsh implements a ONELAB (https://onelab.info) server to pilot external solvers, called "clients"
- Example client: GetDP finite element solver (https://getdp.info)

  - The ONELAB interface allows to call such clients and have them share parameters and modeling information
  - Parameters are directly controllable from the GUI



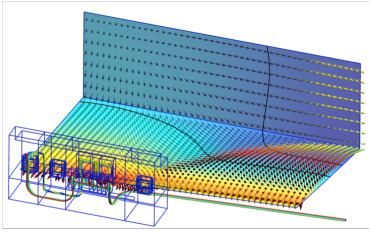
Gmsh - Solver module

- The implementation is based on a client-server model, with a server-side database and local or remote clients communicating in-memory or through TCP/IP sockets
  - Contrary to most solver interfaces, the ONELAB server has no a priori knowledge about any specifics (input file format, syntax, ...) of the clients
  - This is made possible by having any simulation preceded by an analysis phase, during which the clients are asked to upload their parameter set to the server
  - The issues of completeness and consistency of the parameter sets are completely dealt with on the client side: the role of ONELAB is limited to data centralization, modification and re-dispatching

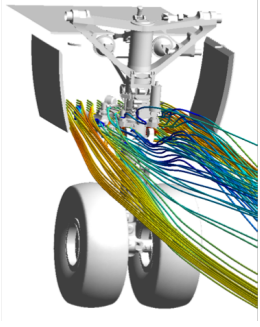
Gmsh - Post-processing module

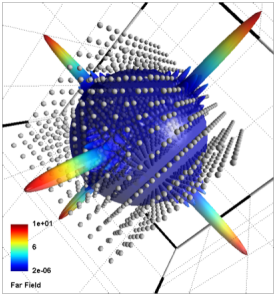
- Post-processing data is made of *views*
- A view stores both display *options* and *data* (unless the view is an *alias* of another view)
- View data can contain several *steps* (e.g. to store time series) and can be either linked to one or more models (*mesh-based* data, as stored in `.msh` or `.med` files) or independent from any model (*list-based* data, as stored in parsed `.pos` files)
- Data is interpolated through arbitrary polynomial interpolation schemes; automatic mesh refinement is used for adaptive visualization of high-order views
- Various *plugins* exist to create and modify views

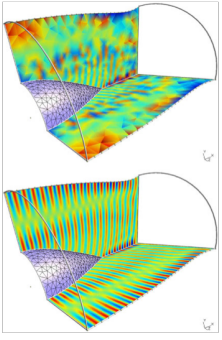
Gmsh - Post-processing module



- Cuts, iso-curves and vectors
- Elevation maps
- Streamlines
- Adaptive high-order visualization



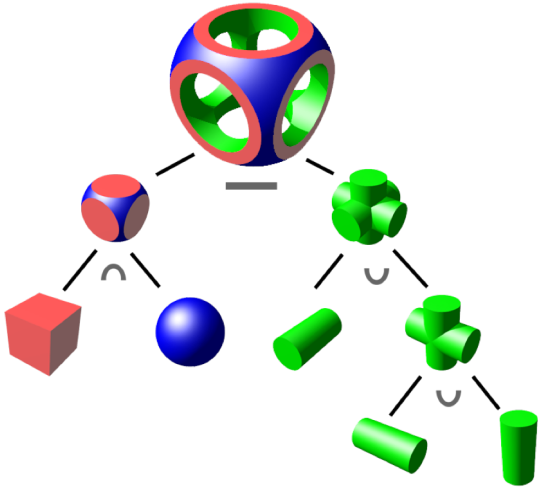




Gmsh - Recent developments

- Constructive Solid Geometry
- Application Programming Interface (API)
- Multi-threaded meshing
- Robust STL remeshing based on parametrizations
- Quasi-structured quad meshing

Gmsh - Constructive Solid Geometry

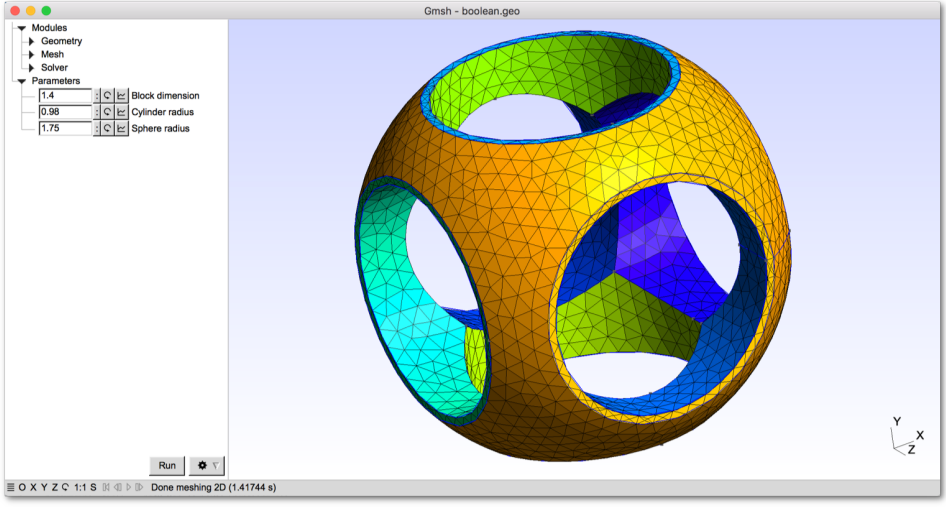


https://en.wikipedia.org/wiki/Constructive_solid_geometry

Gmsh - Constructive Solid Geometry

```
SetFactory("OpenCASCADE"); // use OpenCASCADE kernel

R = DefineNumber[ 1.4 , Min 0.1, Max 2, Step 0.01,
                  Name "Parameters/Box dimension" ];
Rs = DefineNumber[ R*.7 , Min 0.1, Max 2, Step 0.01,
                   Name "Parameters/Cylinder radius" ];
Rt = DefineNumber[ R*1.25, Min 0.1, Max 2, Step 0.01,
                   Name "Parameters/Sphere radius" ];

Box(1) = {-R,-R,-R, 2*R,2*R,2*R}; // explicit entity tag

Sphere(2) = {0,0,0, Rt};

BooleanIntersection(3) = { Volume{1}; Delete; }{ Volume{2}; Delete; };
                                      // delete object and tool
Cylinder(4) = {-2*R,0,0, 4*R,0,0, Rs};
Cylinder(5) = {0,-2*R,0, 0,4*R,0, Rs};
Cylinder(6) = {0,0,-2*R, 0,0,4*R, Rs};

BooleanUnion(7) = { Volume{4}; Delete; }{ Volume{5,6}; Delete; };
BooleanDifference(8) = { Volume{3}; Delete; }{ Volume{7}; Delete; };
```

Gmsh - Constructive Solid Geometry

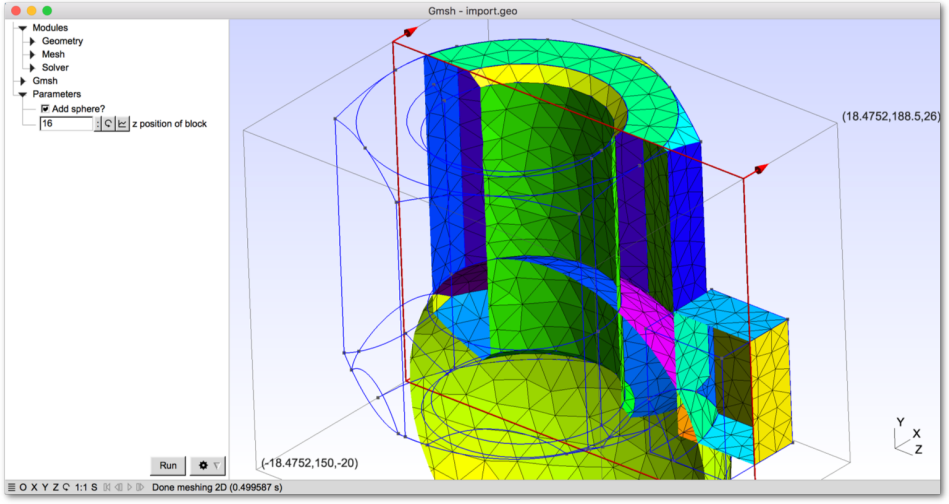


examples/boolean/boolean.geo

Gmsh - Constructive Solid Geometry

```
SetFactory("OpenCASCADE");

DefineConstant[
  z = {16, Name "Parameters/z position of box"}
  sph = {0, Choices{0,1}, Name "Parameters/Add sphere?"}
];

a() = ShapeFromFile("component8.step"); // import STEP shape
b() = 2;
Box(b(0)) = {0,156,z, 10,170,z+10};

If(sph)
  b() += 3;
  Sphere(b(1)) = {0,150,0, 20};
EndIf

// fragmentation intersects everything
r() = BooleanFragments{ Volume{a()}; Delete; }{ Volume{b()}; Delete; };
Save "merged.brep"; // save into native OpenCASCADE format

Physical Volume("Combined volume", 1) = {r()};
Physical Surface("Combined boundary", 2) = CombinedBoundary{ Volume{r()}; }
```

Gmsh - Constructive Solid Geometry



examples/boolean/import.geo

Gmsh - API

Gmsh 4 introduces a new stable Application Programming Interface (API) for C++, C, Python and Julia, with the following design goals:

- Allow to do everything that can be done in `.geo` files
  - ... and then much more!
- Be robust, in particular to wrong input data (i.e. "never crash")
- Be efficient; but still allow to do simple things, simply
- Be maintainable over the long run

Gmsh - API

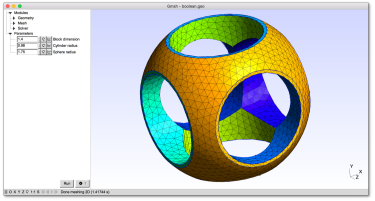
To achieve these goals the Gmsh API

- is purely functional
- only uses basic types from the target language (C++, C, Python or Julia)
- is automatically generated from a master API description file
- is fully documented

Gmsh - API

Same boolean example as before, but using the Python API:

```
import gmsh

gmsh.initialize()
gmsh.model.add("boolean")

R = 1.4; Rs = R*.7; Rt = R*1.25

gmsh.model.occ.addBox(-R,-R,-R, 2*R,2*R,2*R, 1)
gmsh.model.occ.addSphere(0,0,0,Rt, 2)
gmsh.model.occ.intersect([(3, 1)], [(3, 2)], 3)
gmsh.model.occ.addCylinder(-2*R,0,0, 4*R,0,0, Rs, 4)
gmsh.model.occ.addCylinder(0,-2*R,0, 0,4*R,0, Rs, 5)
gmsh.model.occ.addCylinder(0,0,-2*R, 0,0,4*R, Rs, 6)
gmsh.model.occ.fuse([(3, 4), (3, 5)], [(3, 6)], 7)
gmsh.model.occ.cut([(3, 3)], [(3, 7)], 8)

gmsh.model.occ.synchronize()
gmsh.model.mesh.generate(3)
gmsh.fltk.run()
gmsh.finalize()
```

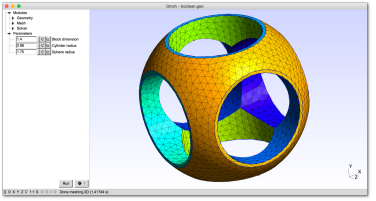


examples/api/boolean.py

Gmsh - API

... or using the C++ API:

```
#include <gmsh.h>

int main(int argc, char **argv)
{
  gmsh::initialize(argc, argv);
  gmsh::model::add("boolean");

  double R = 1.4, Rs = R*.7, Rt = R*1.25;

  std::vector<std::pair<int, int> > ov;
  std::vector<std::vector<std::pair<int, int> > > ovv;
  gmsh::model::occ::addBox(-R,-R,-R, 2*R,2*R,2*R, 1);
  gmsh::model::occ::addSphere(0,0,0,Rt, 2);
  gmsh::model::occ::intersect({{3, 1}}, {{3, 2}}, ov, ovv, 3);
  gmsh::model::occ::addCylinder(-2*R,0,0, 4*R,0,0, Rs, 4);
  gmsh::model::occ::addCylinder(0,-2*R,0, 0,4*R,0, Rs, 5);
  gmsh::model::occ::addCylinder(0,0,-2*R, 0,0,4*R, Rs, 6);
  gmsh::model::occ::fuse({{3, 4}, {3, 5}}, {{3, 6}}, ov, ovv, 7);
  gmsh::model::occ::cut({{3, 3}}, {{3, 7}}, ov, ovv, 8);

  gmsh::model::occ::synchronize();

  gmsh::model::mesh::generate(3);
  gmsh::fltk::run();
  gmsh::finalize();
  return 0;
}
```



examples/api/boolean.cpp

Gmsh - API

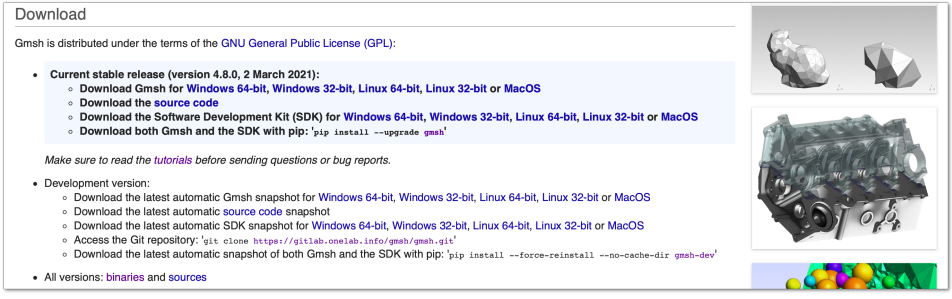
In addition to CAD creation and meshing, the API can be used to

- Access mesh data (`getNodes`, `getElements`)
- Generate interpolation (`getBasisFunctions`) and integration (`getJacobians`) data to build Finite Element and related solvers (see e.g. `examples/api/poisson.py`)
- Create post-processing views
- Run the graphical user-interface
- Build custom graphical user-interfaces, e.g. for domain-specific codes (see `examples/api/prepro.py` or `examples/api/custom_gui.py`) or co-post-processing via ONELAB

Gmsh - API

In order to make this API easy to use, we publish a binary Software Development Toolkit (SDK):

- Continuously delivered (for each commit in master), like the Gmsh app
- Contains the dynamic Gmsh library together with the corresponding C++/C header files, and Python and Julia modules



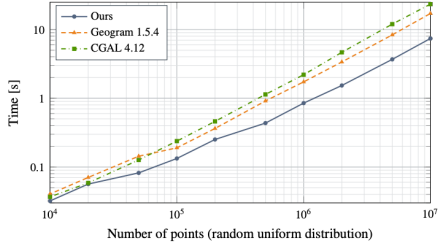
Gmsh - Multi-threaded meshing

Most meshing algorithms are now multi-threaded using OpenMP:

- 1D and 2D algorithms are multithreaded using coarse-grained approach, i.e. several curves/surfaces are meshed concurrently
- The new 3D Delaunay-based algorithm is multi-threaded using a fine-grained approach. It currently lacks several features (embedded entities, hybrid meshes, ...), which will eventually be supported

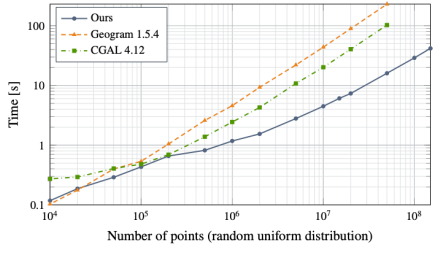
You need to recompile Gmsh with `-DENABLE_OPENMP=1` to enable this; then e.g. `gmsh file.geo -3 -nt 8 -algo hxt`

Gmsh - Multi-threaded meshing



# vertices	10^4	10^5	10^6	10^7
Ours	0.032	0.13	0.85	7.40
Geogram	0.041	0.19	1.73	17.11
CGAL	0.037	0.24	2.20	23.37

(a) 4-core Intel® Core™ i7-6700HQ CPU.



# vertices	10^4	10^5	10^6	10^7	10^8
Ours	0.11	0.43	1.17	4.48	28.95
Geogram	0.10	0.54	4.58	43.70	/
CGAL	0.27	0.48	2.44	20.15	/

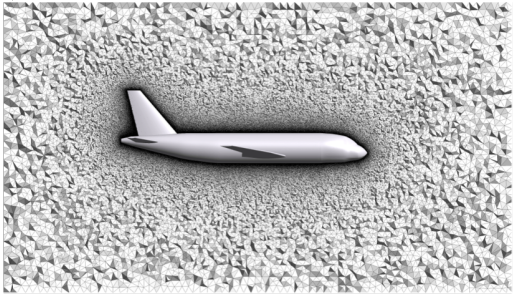
(b) 64-core Intel® Xeon Phi™ 7210 CPU.

[C. Marot et al., IJNME 2019]

Gmsh - Multi-threaded meshing

Truck tire

# threads	# tetrahedra	Timings (s)		
		BR	Refine	Total
1	123 640 429	75.9	259.7	364.7
2	123 593 913	74.5	166.8	267.1
4	123 625 696	74.2	107.4	203.6
8	123 452 318	74.2	95.5	190.0

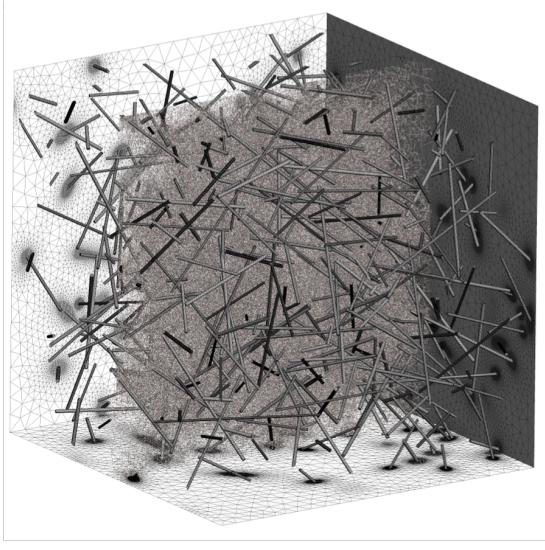


Aircraft

# threads	# tetrahedra	Timings (s)		
		BR	Refine	Total
1	672 209 630	45.2	1348.5	1418.3
2	671 432 038	42.1	1148.9	1211.5
8	665 826 109	39.6	714.8	774.8
64	664 587 093	38.7	322.3	380.9
127	663 921 974	38.1	255.0	313.3

AMD EPYC 2x 64-core

Gmsh - Multi-threaded meshing



100 thin fibers

# threads	# tetrahedra	Timings (s)		
		BR	Refine	Total
1	325 611 841	3.1	492.1	497.2
2	325 786 170	2.9	329.7	334.3
4	325 691 796	2.8	229.5	233.9
8	325 211 989	2.7	154.6	158.7
16	324 897 471	2.8	96.8	100.9
32	325 221 244	2.7	71.7	75.8
64	324 701 883	2.8	55.8	60.1
127	324 190 447	2.9	47.6	52.0

500 thin fibers

# threads	# tetrahedra	Timings (s)		
		BR	Refine	Total
1	723 208 595	18.9	1205.8	1234.4
2	723 098 577	16.0	780.3	804.8
4	722 664 991	86.6	567.1	659.8
8	722 329 174	15.8	349.1	370.1
16	723 093 143	15.6	216.2	236.5
32	722 013 476	15.6	149.7	169.8
64	721 572 235	15.9	119.7	140.4
127	721 591 846	15.9	114.2	135.2

AMD EPYC 2x 64-core

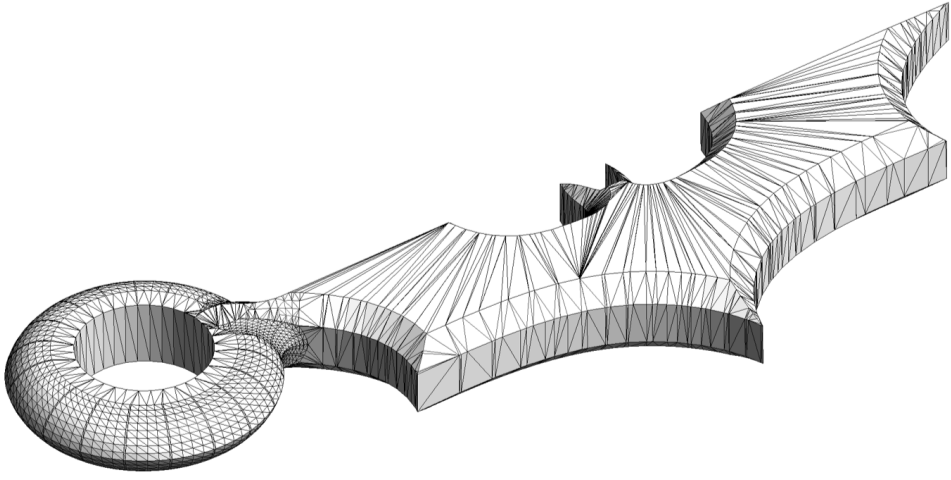
Gmsh - Robust STL remeshing

New pipeline to remesh discrete surfaces (represented by triangulations):

- Automatic construction of a set of parametrizations that form an atlas of the model
- Each parametrization is guaranteed to be one-to-one, amenable to meshing using existing algorithms
- New nodes are guaranteed to be on the input triangulation ("no modelling")
- Optional pre-processing (i.e. edge detection) to color sub-patches if sharp features need to be preserved

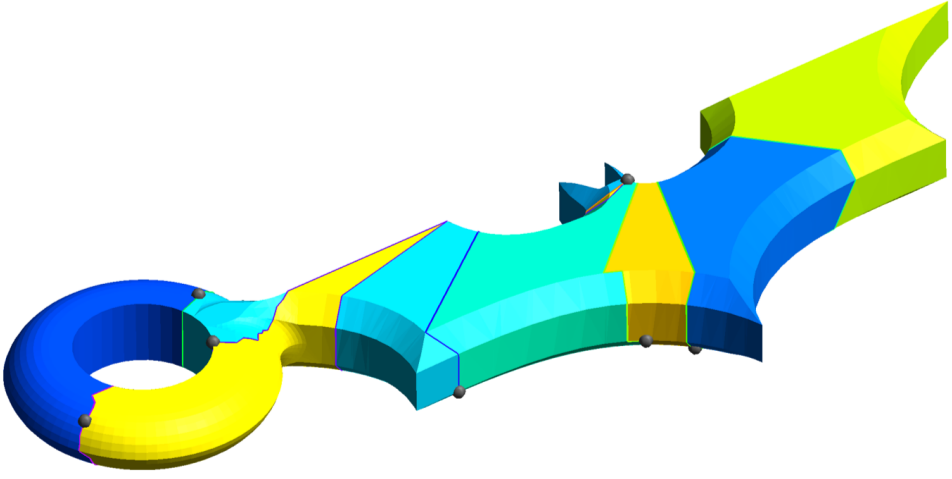
[P. A. Beaufort et al., JCP 2020]

Gmsh - Robust STL remeshing



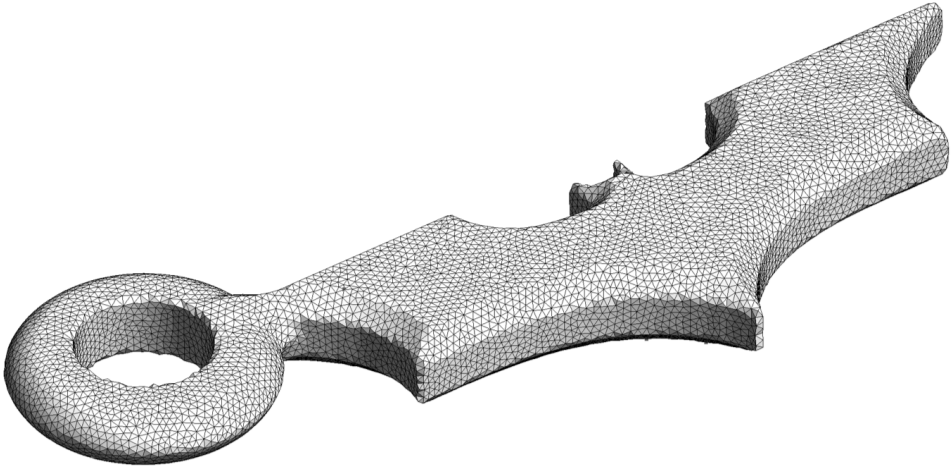
Batman STL mesh

Gmsh - Robust STL remeshing



Automatic atlas creation: each patch is provably parametrizable by solving a linear PDE, using mean value coordinates

Gmsh - Robust STL remeshing

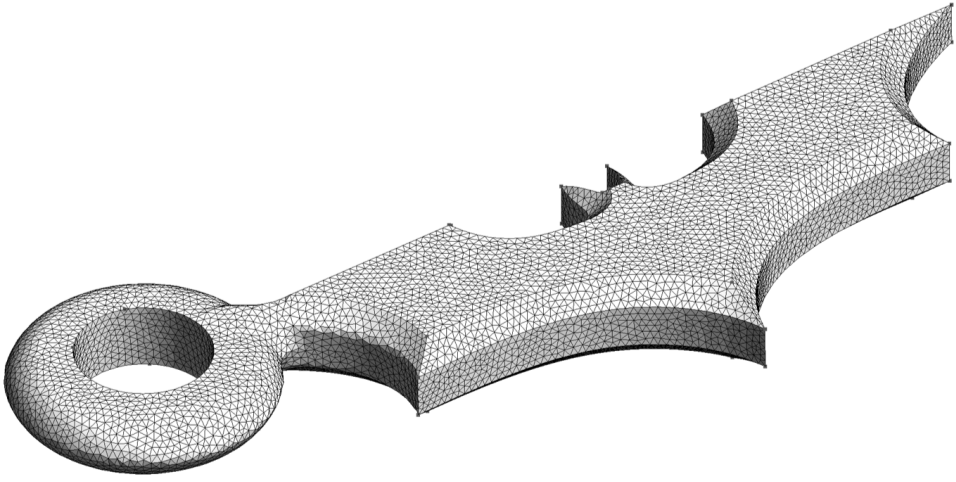


Remeshing

Gmsh - Robust STL remeshing

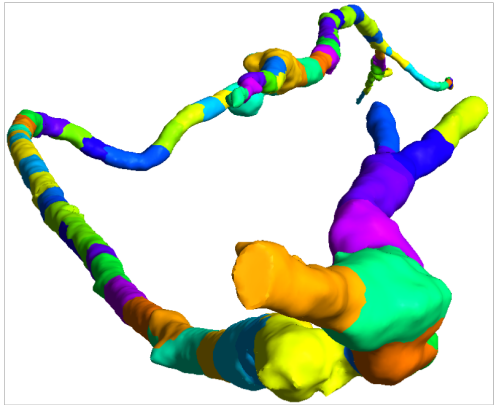
Automatic atlas creation, this time with feature edge detection

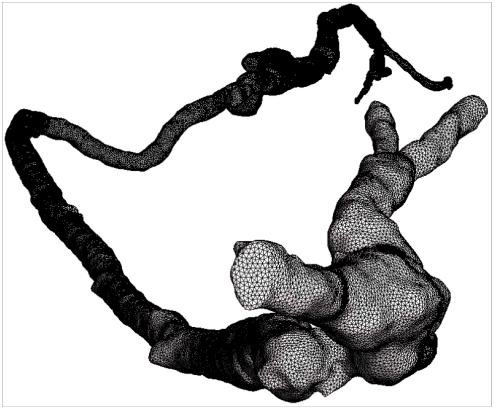
Gmsh - Robust STL remeshing



Remeshing with feature edge detection

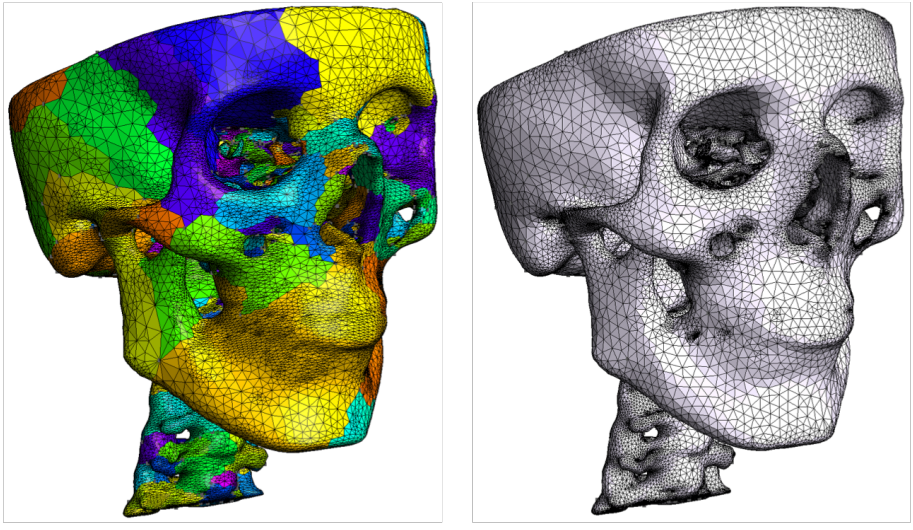
Gmsh - Robust STL remeshing





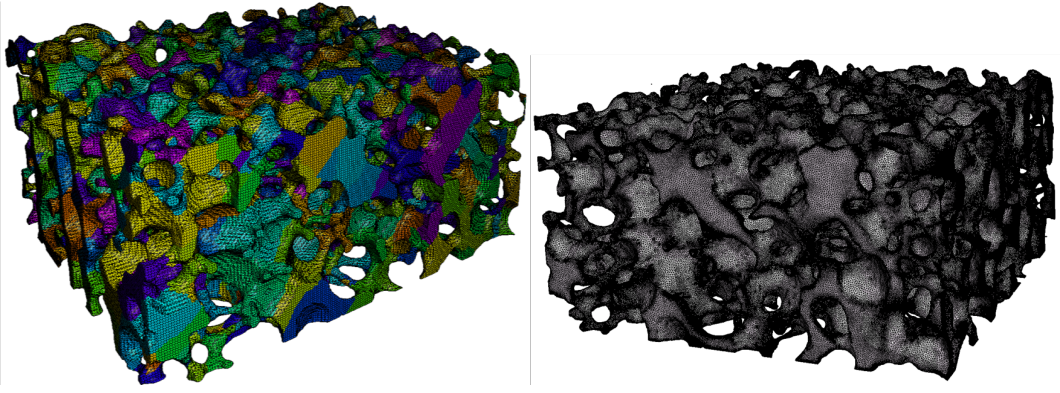
CT scan of an artery: 101 patches created, most because of the large aspect ratio

Gmsh - Robust STL remeshing



Remeshing of a skull: 715 patches created for reparametrization; mesh adapted to curvature

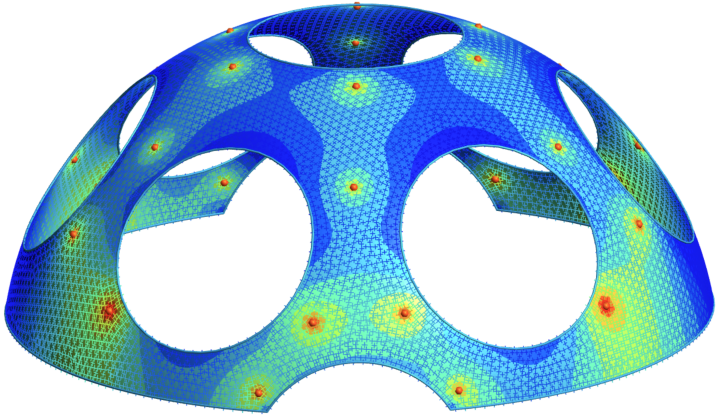
Gmsh - Robust STL remeshing



Remeshing of an X-ray tomography image of a silicon carbide foam by P. Duru, F. Muller and L. Selle (IMFT, ERC Advanced Grant SCIROCCO): 1,802 patches created for reparametrization

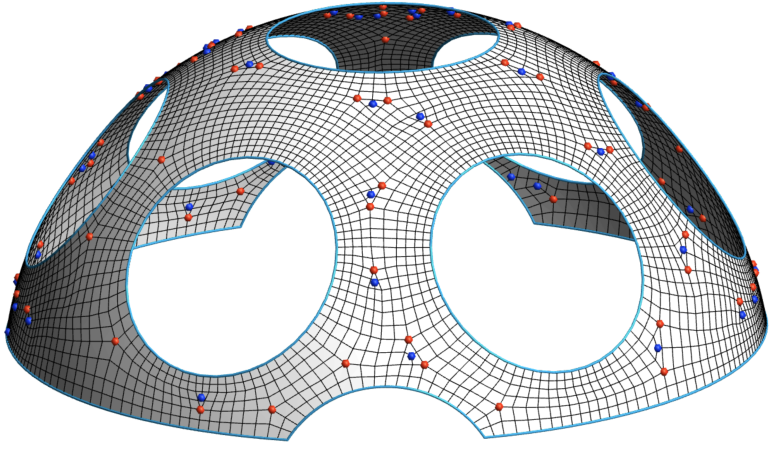
Gmsh - Quasi-structured quad meshing

New experimental algorithm by [M. Reberol et al. 2021] that has just landed in the development version



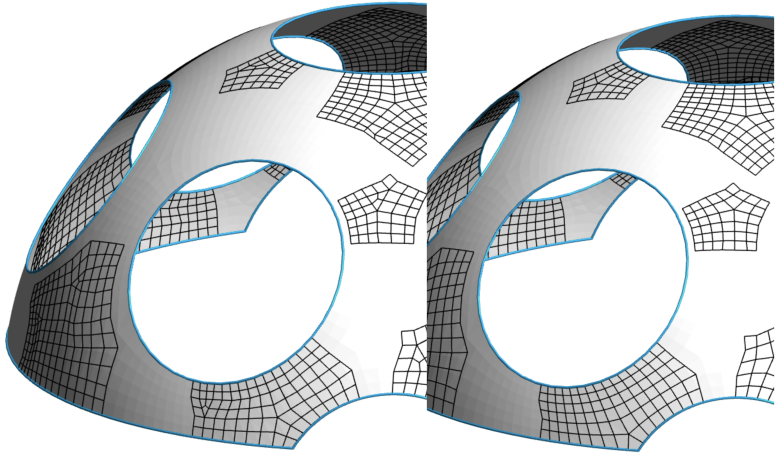
Compute a (scaled) cross-field with multilevel diffusion

Gmsh - Quasi-structured quad meshing



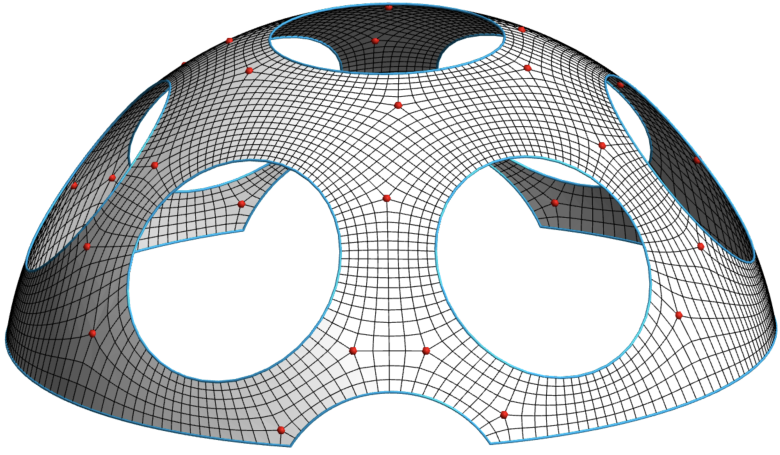
Build a unstructured quadrilateral mesh with a frontal approach guided by the scaled cross field

Gmsh - Quasi-structured quad meshing



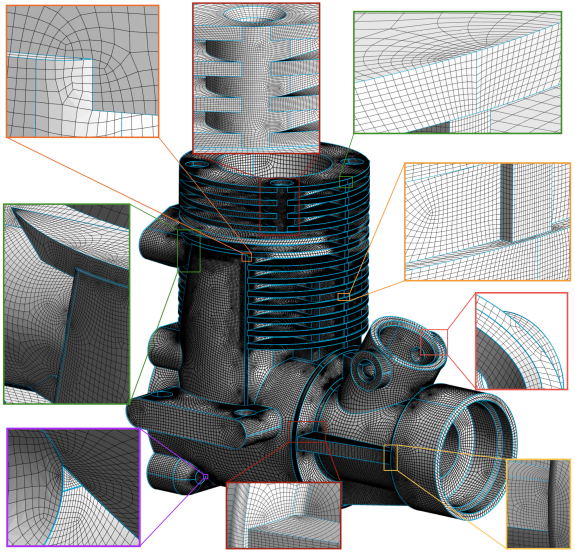
Pattern-based quadrilateral meshing and cavity remeshing to eliminate unnecessary irregular vertices while preserving the cross field singularities

Gmsh - Quasi-structured quad meshing



The final quad mesh is very similar to the one obtained with the global parametrization approach and has the same number of irregular vertices

Gmsh - Quasi-structured quad meshing



- "Block" model: 533 surfaces, 1584 curves, 261.5k vertices, 261.6k quads
- Average SICN quality: 0.87 (minimum: 0.11)
- 58 sec. (initial unstructured quad mesh) + 33 sec. (quasi-structured improvement) on Intel Core i7 4 cores
- Quasi-structured improvement reduces the number of irregular from 14.4k to 3.6k

LIÈGE université

An overview of GetDP

LIÈGE université

What is GetDP?

- GetDP (https://getdp.info) is a flexible finite element solver using mixed elements
- Open source (GNU GPL v2+); same community infrastructure as Gmsh
- Main feature: closeness between
  - the input data defining discrete problems (written in plain text `.pro` files), and
  - the symbolic mathematical expressions of these problems
- New models developed through `.pro` files — no compilation
- Highly portable: exact same `.pro` files on iPhone and 10,000 core supercomputer

GetDP - A little bit of history

- GetDP was started at the end of 1996
- 1998: First feature-complete release, binary-only; original paper
- 2002: Socket-based communication with Gmsh
- 2004: Open sourced under GNU GPL
- 2010: Major refactoring and move to C++ (GetDP 2)
- 2011: New default solvers based on PETSc and SLEPc
- 2012: ONELAB interface
- 2014: Parallel domain decomposition methods (GetDDM)
- 2015: Macros, run-time variables, Python and Octave interpreters
- 2017: Polynomial and rational nonlinear eigenvalue problems
- 2018: Curved meshes and enhanced Gmsh support (GetDP 3)
- 2020: Auto-similar trees of edges

GetDP - Basic concepts

In a `.pro` file, the weak formulation: Find $u(x) \in H_0^1(\Omega)$ such that

$$-\int_\Omega a(x)\nabla u \cdot \nabla u' \, d\Omega = \int_\Omega f(x)\, u' \, d\Omega, \quad \forall u' \in H_0^1(\Omega)$$

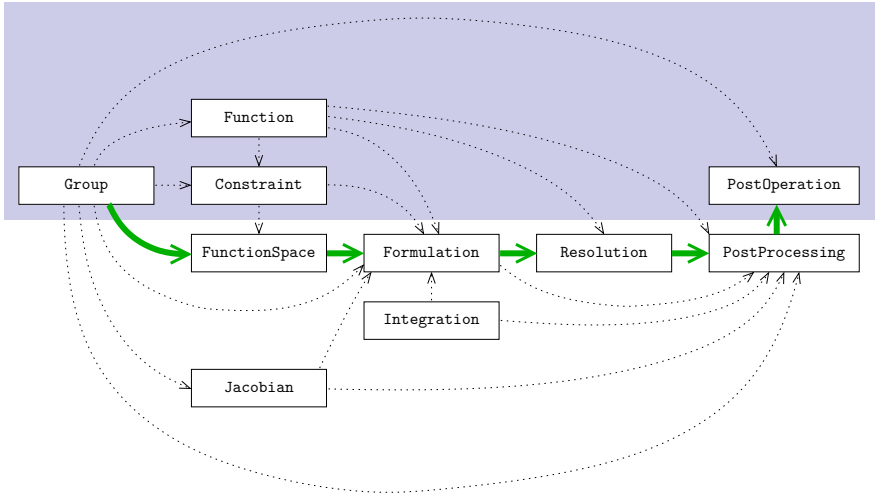
is transcribed as

```
Formulation{
  { Name MyFirstFormulation; Type FemEquation;
    Quantity {
      { Name u; Type Local; NameOfSpace H1_0; }
    }
    Equation {
      Integral { [ -a[] * Dof{d u}, {d u} ];
                                  In Omega; Integration I; Jacobian J; }
      Integral { [ -f[], {u} ]; In Omega; Integration I; Jacobian J; }
    }
  }
}
```

GetDP - Main features

- H^1, $H(\text{curl})$, $H(\text{div})$ and L^2 bases in 1D, 2D and 3D; first and second order for all element types; higher order for some
- Natural handling of non-local (global) quantities
- Circuit coupling
- Static, transient, harmonic, multi-harmonic resolutions
- Linear and nonlinear eigenvalue problems
- Easy coupling of fields and formulations, staggered or monolithic, e.g. for explicit Jacobian matrices of strongly coupled nonlinear problems
- Linear algebra through PETSc/SLEPc
- Built-in Python and Octave interpreters
- Flexible templating mechanism, allowing one to build a library of generic formulations

GetDP - Ten objects



On top, over the blue background, the "user" objects that are sufficient to parameterize a problem once it has been templated

GetDP - Group

- Link with the (physical groups) in a mesh
- Functions on groups: nodes, edges, tree of edges, ...

```
Group{
  Air = Region[1]; // link with physical group 1 in the mesh
  Core = Region[2];
  Gamma = Region[{3, 4}];

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

  nodes = NodesOf[Omega]; // nodes of the elements in Omega
  edgesOfSpanningTree = EdgesOfTreeIn[Omega, StartingOn Gamma];
}
```

GetDP - Function

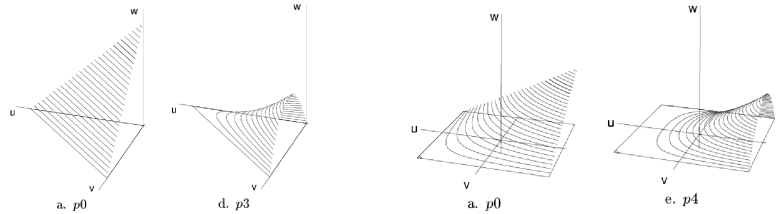
- Piecewise definitions
- Space-time dependent
- Physical characteristics, sources, constraints, ...

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

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

  // using a runtime variable
  TimeFct[] = Cos[2*Pi*f*$Time] * Exp[-$Time];
}
```

GetDP - FunctionSpace

- 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; }
    }
  }
}
```




```
// second-order version, using hierarchical functions
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; }
    }
  }
}
```

Exact sequence of the de Rham Complex

$$H^1_h(\Omega) \xrightarrow{\textbf{grad}_h} \boldsymbol{H}_h(\textbf{curl};\Omega) \xrightarrow{\textbf{curl}_h} \boldsymbol{H}_h(\mathrm{div};\Omega) \xrightarrow{\mathrm{div}_h} L^2(\Omega)$$

preserved at the discrete level using Whitney elements. E.g. for $\boldsymbol{H}_h(\textbf{curl};\Omega)$:

```
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; }
    }
  }
}
```

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

LIÈGE université

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; }
    }
  }
```

LIÈGE université

Global Quantities

$$v = \sum_{n \in N_v} v_n s_n + v_f \left(\sum_{n \in N_f} s_n \right) = \sum_{n \in N_v} v_n s_n + v_f s_f$$

```
FunctionSpace {
  { Name Hgrad_v_floating; Type Form0;
    BasisFunction {
      { Name sn; NameOfCoef vn; Function BF_Node;
        Support Domain; Entity NodesOf[All, Not SkinDomainC]; }
      { Name sf; NameOfCoef vf; Function BF_GroupOfNodes;
        Support Domain; Entity GroupsOfNodesOf[SkinDomainC]; }
    }
    GlobalQuantity {
      { Name GlobalElectricPotential; Type AliasOf; NameOfCoef vf; }
      { Name GlobalElectricCharge; Type AssociatedWith; NameOfCoef vf; }
    }
  }
}
```

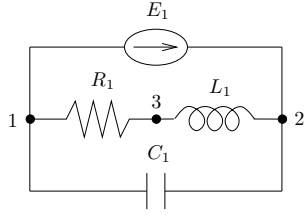
GetDP - Constraint

- Boundary conditions (Dirichlet, periodic)
- 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; }
    }
  }
}
```

Network constraints and circuit coupling

```
Constraint{
  { 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}; }
    }
  }
}
```



GetDP - Formulation

- Equation builder, with symbolic expression of weak forms
- 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 {
      Integral { [ 1/mu[] * Dof{Curl e}, {Curl e} ];
                 In Omega; Jacobian Jac1; Integration Int1; }
      Integral { 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 {
      Integral { [ Z0 * OSRC_C0[]{N, th} * Dof{nxh}, {nxh} ];
                 In Gama; Jacobian JSur; Integration I1; }
      Integral { [ -{psi}, {nxh} ];
                 In Gama; Jacobian JSur; Integration I1; }
      For j In{1:N}
        Integral { [ Z0 * OSRC_Aj[]{j, N, th} * Dof{phi~{j}}, {nxh} ];
                   In Gama; Jacobian JSur; Integration I1; }
        Integral { [ Dof{phi~{j}}, {phi~{j}} ];
                   In Gama; Jacobian JSur; Integration I1; }
        Integral { [ -OSRC_Bj[]{j, N, th} / keps[]^2 * Dof{d phi~{j}}, {d phi~{j}} ];
                   In Gama; Jacobian JSur; Integration I1; }
        Integral { [ 1 / keps[]^2 * Dof{d nxh}, {d phi~{j}} ];
                   In Gama; Jacobian JSur; Integration I1; }
      EndFor
    }
  }
}
```

GetDP - Jacobian

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

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

GetDP - Integration

- 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; }
    }
  }
}
```

GetDP - Resolution

- Description of the sequence of operations to solve the problem
- 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]; }
      }
    }
  }
```

GetDP - PostProcessing

- "Front-end" to computational data
- Piecewise definition of any post-processing 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; }
          }
      }
   }
}
```

GetDP - PostOperation

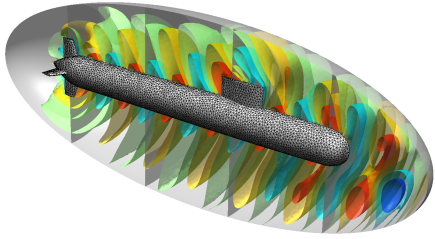
- Evaluation of post-processing quantities (e.g. maps, sections, local or global evaluation, ...)
- Various output formats (e.g. space or time oriented, text, binary, ...)
- Output to disk or to memory as ONELAB parameters or Gmsh post-processing views

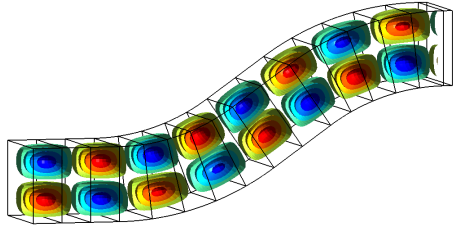
```
PostOperation {
  { Name Map_b; NameOfPostProcessing magfields;
    Operation {
      Print[ b, OnElementsOf Omega, File "b.pos", Format Gmsh ];
      Print[ b, OnLine {{0, 0, 0} {1, 0, 0}} {100}, File "b.txt" ];
    }
  }
}
```

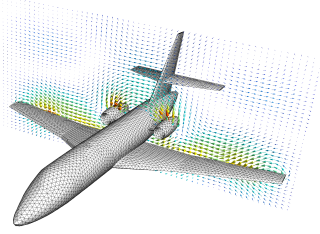
GetDP - Recent developments

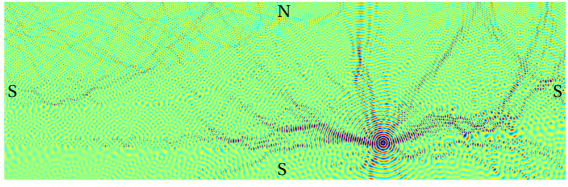
- Optimized Schwarz domain decomposition methods
- Vector energy-based hysteresis models
- Sensitivity calculations for shape and topology optimization
- Computational homogenization
- 3D moving band with symmetries
- Nonlinear eigenvalue problems

GetDP - Optimized Schwarz DDM





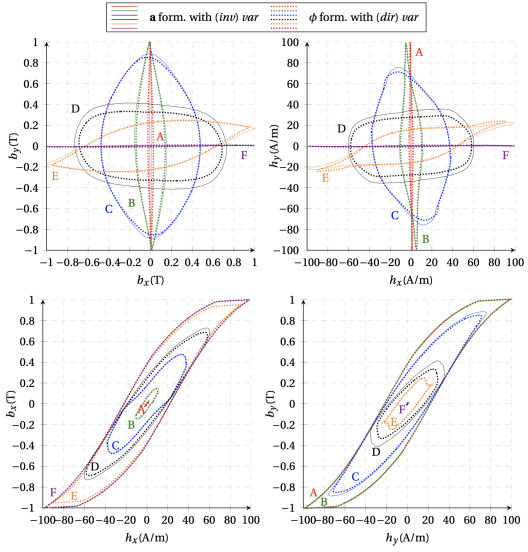


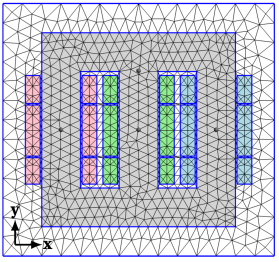


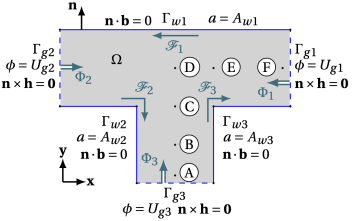
Massively parallel, distributed (MPI) optimized Schwarz domain decomposition methods for high-frequency wave scattering

[B. Thierry et al., CPC 2016]

GetDP - Energy-based hysteresis models

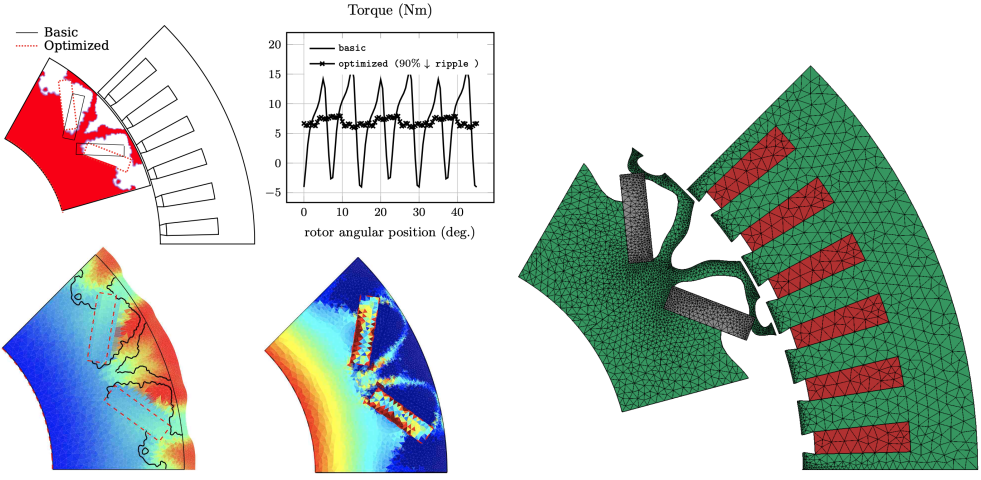






[K. Jacques et al., 2018]

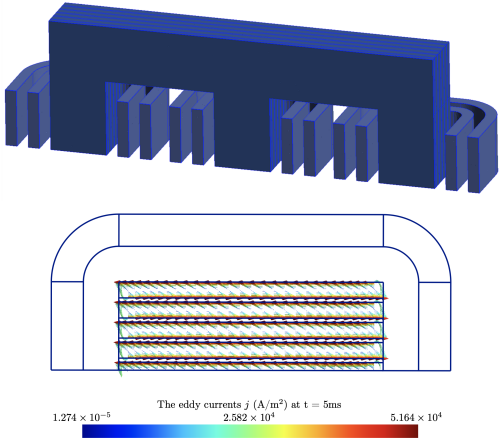
GetDP - Sensitivities and optimization

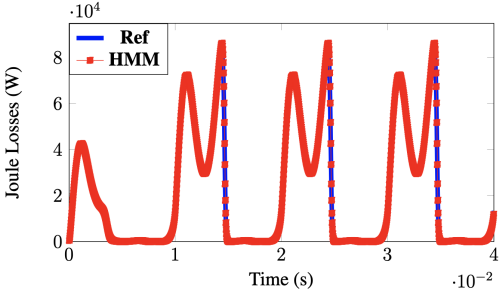


Design sensitivity analysis for combined topology and shape optimization, based on the Lie derivative

[E. Kuci et al., SMO 2019]

GetDP - Computational homogenization

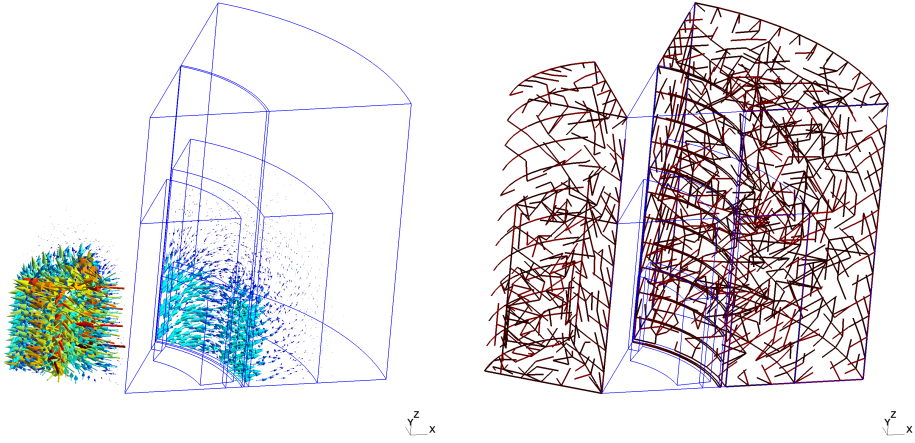




Massively parallel (MPI) Heterogeneous Multi-scale Method (HMM) for 3D lamination stacks

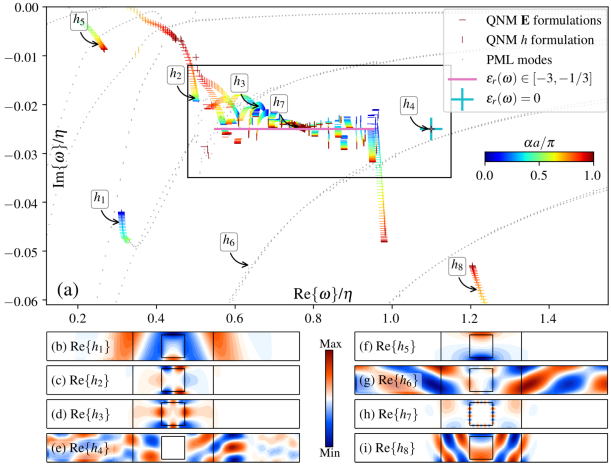
[I. Niyonzima et al, 2021]

GetDP - 3D moving band



Auto-similar trees for efficient and accurate 3D sliding surfaces

GetDP - Nonlinear eigenvalue problems



General polynomial and rational eigenvalue problems for the analysis of frequency-dispersive photonic open structures

[G. Demésy et al, CPC 2020]

LIÈGE université

Tying it all together with ONELAB

ONELAB - Context

- Economic
  - Growing importance of numerical simulation in education and industry
  - Prohibitive cost of commercial packages for a significant subset of potential users (SMEs, education, occasional use)
- Scientific
  - High quality of free/open-source software developed in universities and research centers
  - Sometimes ahead of commercial equivalents
- Practical
  - No user-friendly interface and/or sparse documentation (e.g. missing complete examples) for many (most?) open source Finite Element Analysis (FEA) codes

ONELAB - Objective

Develop an interface for integrating FEA software:

- allowing to share parameters between any code
- with an intuitive GUI allowing new users to get started and guided into the world of FE modeling—but with the possibility to construct sophisticated, upgradable, multi-code, multi-platform scripts for the specialized user
- and with the possibility to construct application-specific tools for both education and industry

ONELAB - Design

- Don't reimplement, interface the existing
- Make it as small, lightweight and as easy to maintain as possible (no solver-dependent code in the interface)
- Make it easy to provide templates, with interactive parameter modification
- ONELAB role = data centralization, modification and redispatching
- Issues of completeness and consistency of the parameter set are completely dealt with on the client (e.g. FEA software) side

ONELAB - Design

- Client-server:
  - Clients: CAD kernels, meshers, solvers, post-processors
  - Server: Gmsh + database
- Abstract interface:
  - The server has no a priori knowledge of the clients (no meta-language or exchange file format)
  - The client communicates with the server to define what information should be exchanged
  - Information is exchanged either directly in-memory, or through TCP/IP or Unix sockets

ONELAB - Sharing parameters

Sharing parameter through ONELAB in `.geo` and `.pro` files is as simple as:

```
DefineConstant[ N = { 50, Name "Input/Number of steps"} ];
```

In Python one can use the `onelab.py` module:

```
c = onelab.client()
N = c.defineNumber('Input/Number of steps', value=50)
```

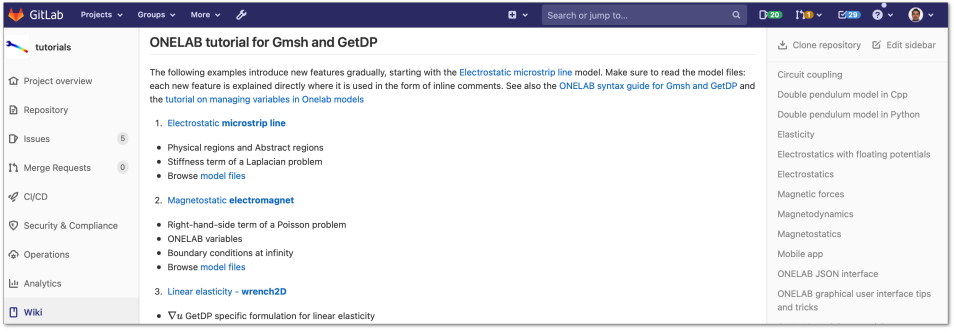
Or one can directly use the Gmsh API, with JSON encoded data:

```
gmsh.onelab.set("""[
  {
    "type":"number",
    "name":"Input/Number of steps",
    "values":[50]
  }
]""")
N = gmsh.onelab.getNumber('Input/Number of steps')
```

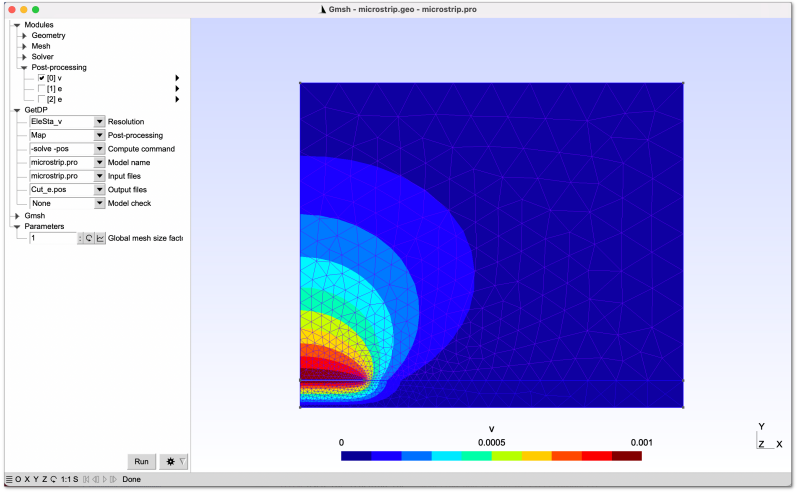
ONELAB - Tutorial

The best way to get started is the ONELAB tutorial:

- Examples of increasing complexity using both Gmsh and GetDP
- Used for teaching the "Modeling and design of electromagnetic systems" class at ULiège since 2018

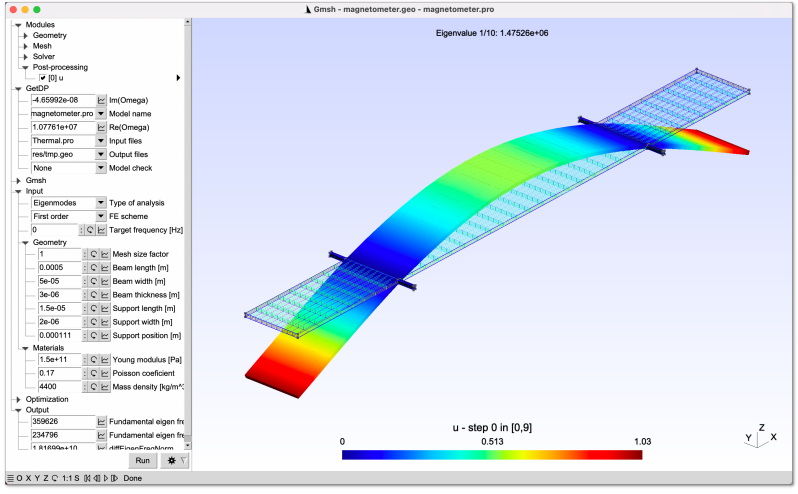


ONELAB - Laplace example



Open `tutorials/Electrostatics/microstrip.pro` with Gmsh

ONELAB - Multiphysics example



Open `models/Magmetomater/magnetometer.pro` with Gmsh

ONELAB - Going further

- In addition to the tutorial, various models are available on https://gitlab.onelab.info/doc/models
- Shape and topology optimization examples are also available on https://gitlab.onelab.info/conveks/tutorials
- ONELAB has proved to be very useful in building application-specific codes for industry
  - Transformer design (Gmsh+GetDP+LTSpice), filter design (Gmsh+GetDP), busbar design (Gmsh+GetDP), behavior of structures subjected to fire (Gmsh+Safir), thermal laser skin treatment (Gmsh+GetDP+Elmer), building resistance (Gmsh+Code_Aster), ...
  - Using the Gmsh API allows one to create dedicated graphical user interfaces, with advanced interactivity for the selection of boundary conditions, specification of material properties, etc.

LIÈGE université

Conclusions and perspectives

Conclusions and perspectives

- Overview of Gmsh, GetDP and ONELAB
  - Modelling freedom (but some coding is necessary)
  - Accessibility, reproducibility and interoperability (free and open source)
  - Easy installation (binary distribution)
  - Encapsulated and scriptable
  - Successfully used both in academia and in industry
  - More “Swiss Army knife” than “bazooka”
  - Some open source alternatives: deal.II, dune-fem, FEMM, FEniCS, FreeFEM, NGSolve, Elmer, MFEM, ...
- Many exciting developments in the pipeline:
  - Meshing: improved curved mesh generation, hex-dominant meshes, boundary layers, ...
  - Solver: next-generation assembler GmshFEM, domain decomposition framework GmshDDM and full wave inversion code GmshFWI; GPU acceleration, ...

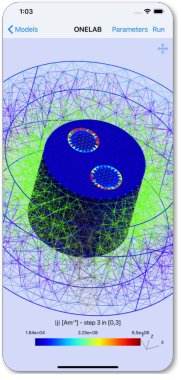
Post-Scriptum

- To download the ONELAB software bundle, including Gmsh and GetDP: `https://onelab.info`

- For fun, go to the
  - Google Play Store (if you are on Android)
  - Apple AppStore (if you are on iOS)

  and download the **ONELAB app**: it contains a full-featured version of Gmsh & GetDP

  ... so you can impress your friends by solving finite element problems on your smartphone!



LIÈGE université

Thanks for your attention

http://people.montefiore.ulg.ac.be/geuzaine

✉ cgeuzaine@uliege.be