

# Gmsh

C. Geuzaine and J.-F. Remacle

Université de Liège and Université catholique de Louvain

August 31, 2019





## 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: <http://gmsh.info>, <http://getdp.info>, <http://onelab.info>



## Some background

- I am a professor at the Université catholique de Louvain in Belgium, where I lead a team of a dozen researchers in the Institute of Mechanics, Materials and Civil Engineering
- We work mainly on mesh generation and high-order methods for PDEs
- Our current main research topic is hex meshing, both unstructured and structured
- I have been co-operating with Christophe for more than 20 years, a fruitful collaboration that has led to the creation of Gmsh

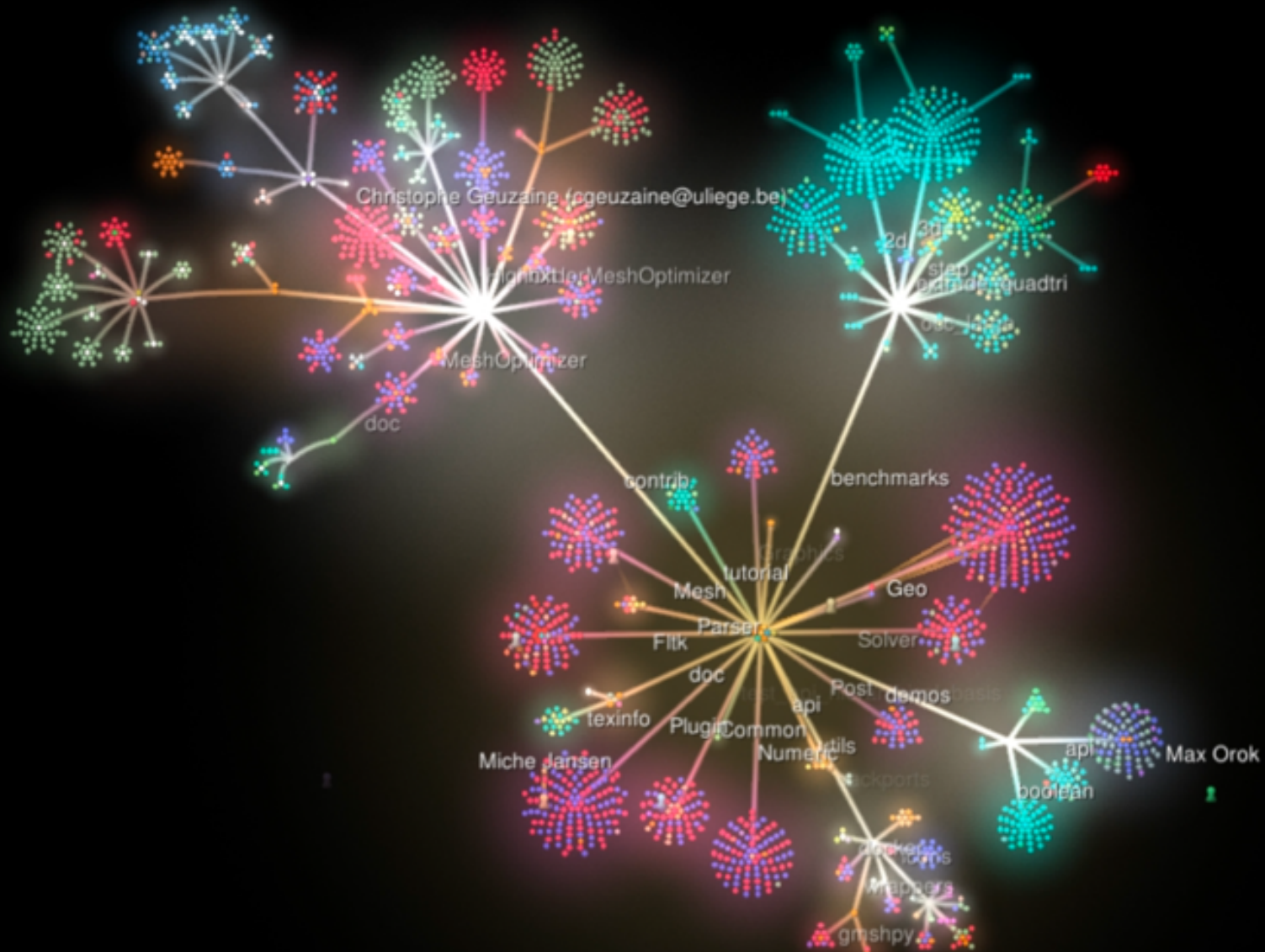
# What is Gmsh?

- Gmsh (<http://gmsh.info>) is an open source 3D finite element mesh generator with a built-in CAD engine and post-processor
- Includes a graphical user interface (GUI) and can drive any simulation code through ONELAB
- Today, Gmsh represents about 500k lines of C++ code
  - still same 2 core developers; about 100 with  $\geq 1$  commit
  - about 1,000 people on mailing lists
  - about 8,000 downloads per month (75% Windows)
  - about 500 citations per year – the Gmsh paper is cited about 3,700 times
  - Gmsh has probably become one of the most popular (open source) finite element mesh generators?





Wednesday, 28 August, 2019 13:39:12



~ 20 years of Gmsh development in 1 minute  
A warm thank you to all the contributors!

## 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: Parallel 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: Parallel 3D meshing (fine-grained), robust STL remeshing

## Strategic choices

- Design goals: fast, light and user-friendly
  - Written in simple C++
  - GUIs: FLTK (desktop), UIKit (iOS), Android, web
  - 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 (BAMG, Metis, Concorde, ...)
- Funding
  - Hobby until 2006, then industry, Wallonia, Belgium & EU

## Strategic choices

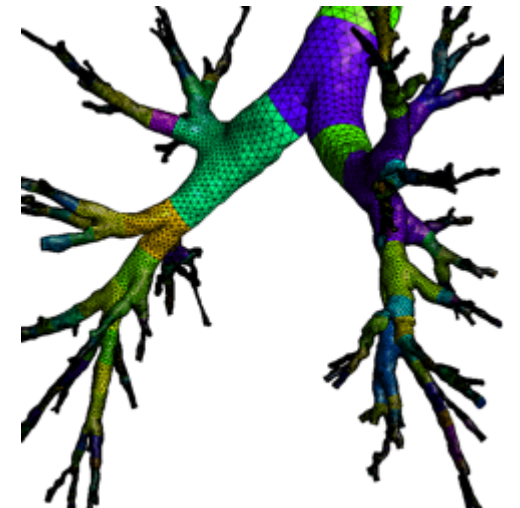
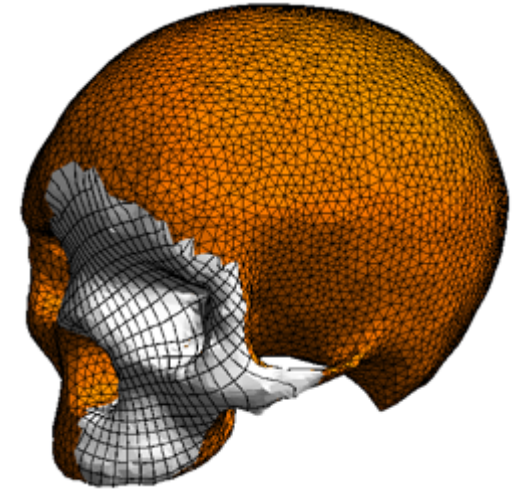
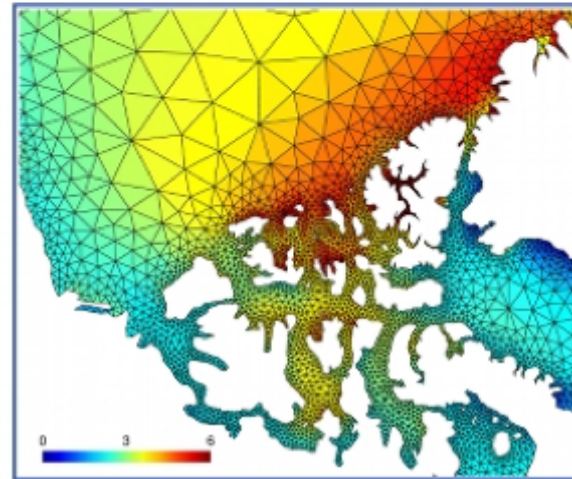
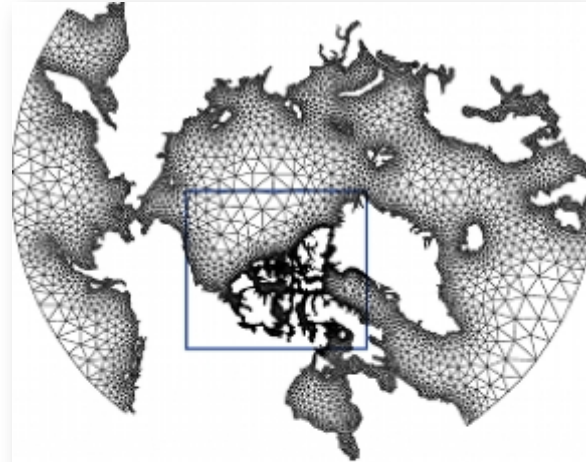
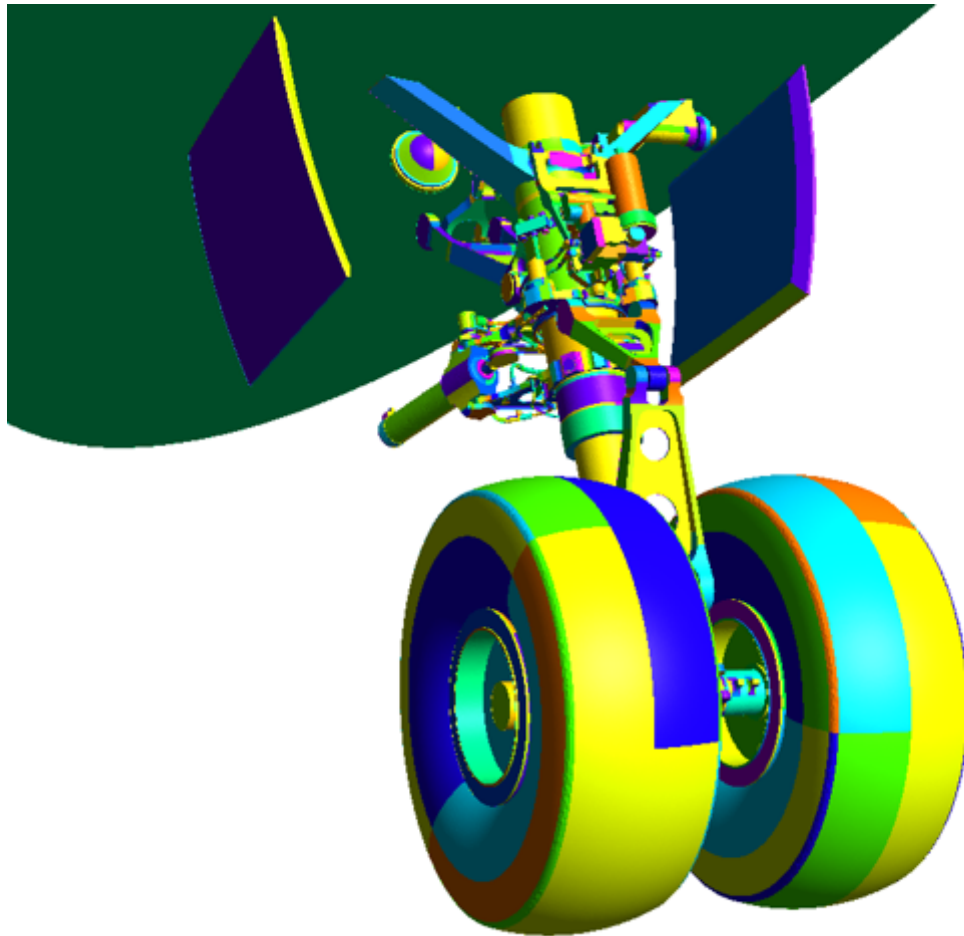
- Community infrastructure
  - Our own (using GitLab) to enable public/private parts (<https://gitlab.onelab.info/gmsh/gmsh>)
  - Continuous integration and delivery (CI/CD) of Gmsh app and Gmsh SDK on Windows, Linux and macOS
  - Two mailing lists and a web site (<http://gmsh.info>) with documentation, 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

## 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 CAD entities, using a Boundary REPresentation (BREP) approach
  - Gmsh never translates from one CAD format to another; it directly accesses each CAD kernel API (OpenCASCADE, Parasolid, Built-in, ...)

# Basic concepts

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





# 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

## 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)



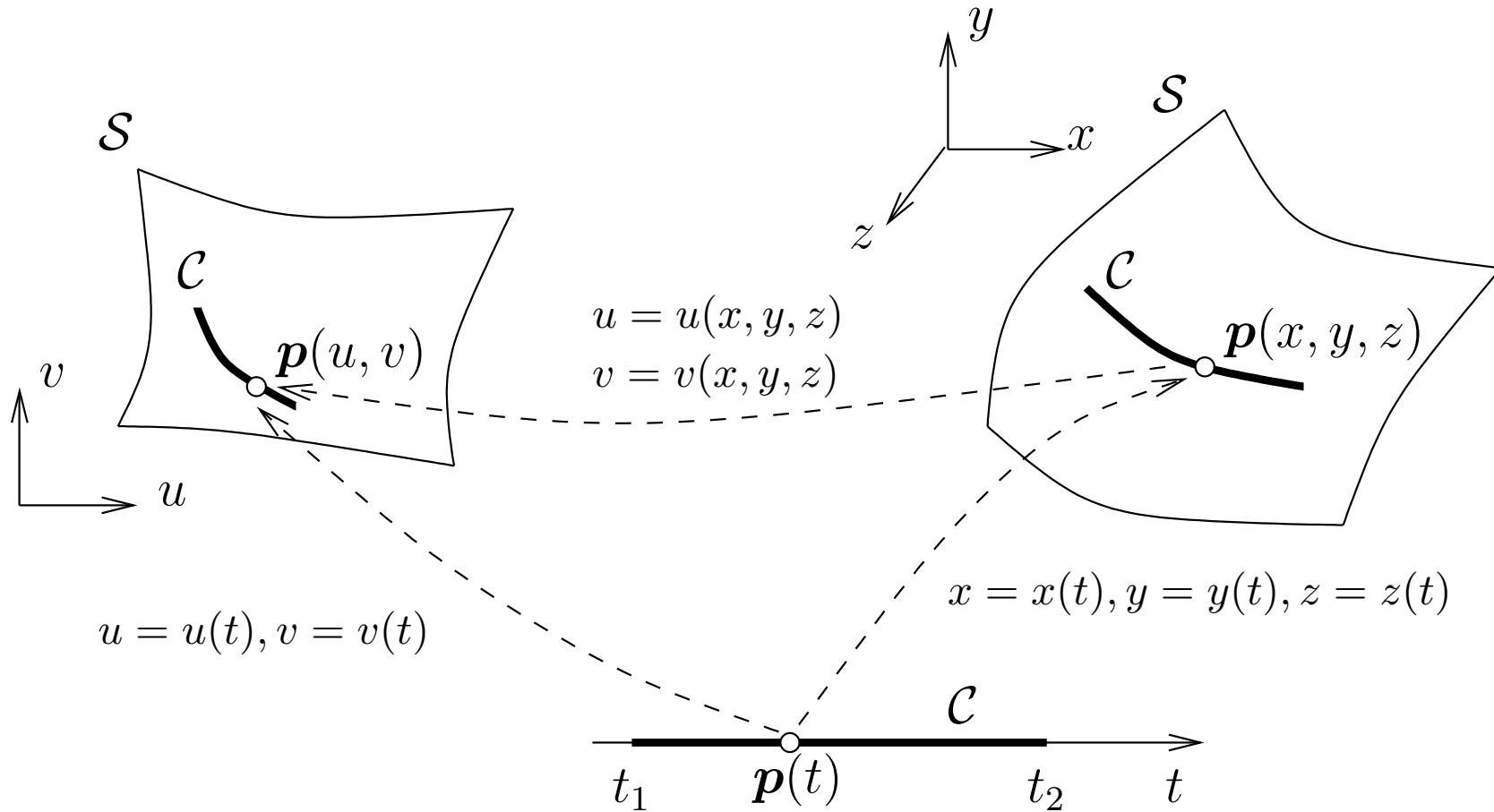
# Geometry module

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  $R^3$

# Geometry module

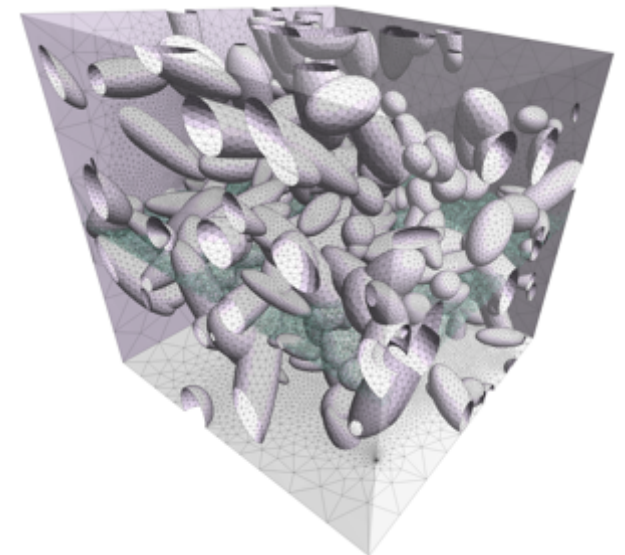
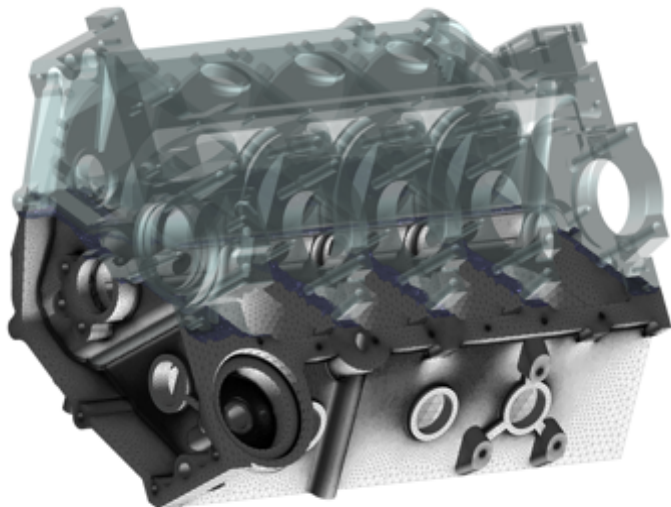


Point  $p$  located on a curve  $C$  that is itself embedded in a surface  $S$

## Geometry module

Operations on CAD model entities are performed directly within their respective CAD kernels (e.g. built-in or OpenCASCADE):

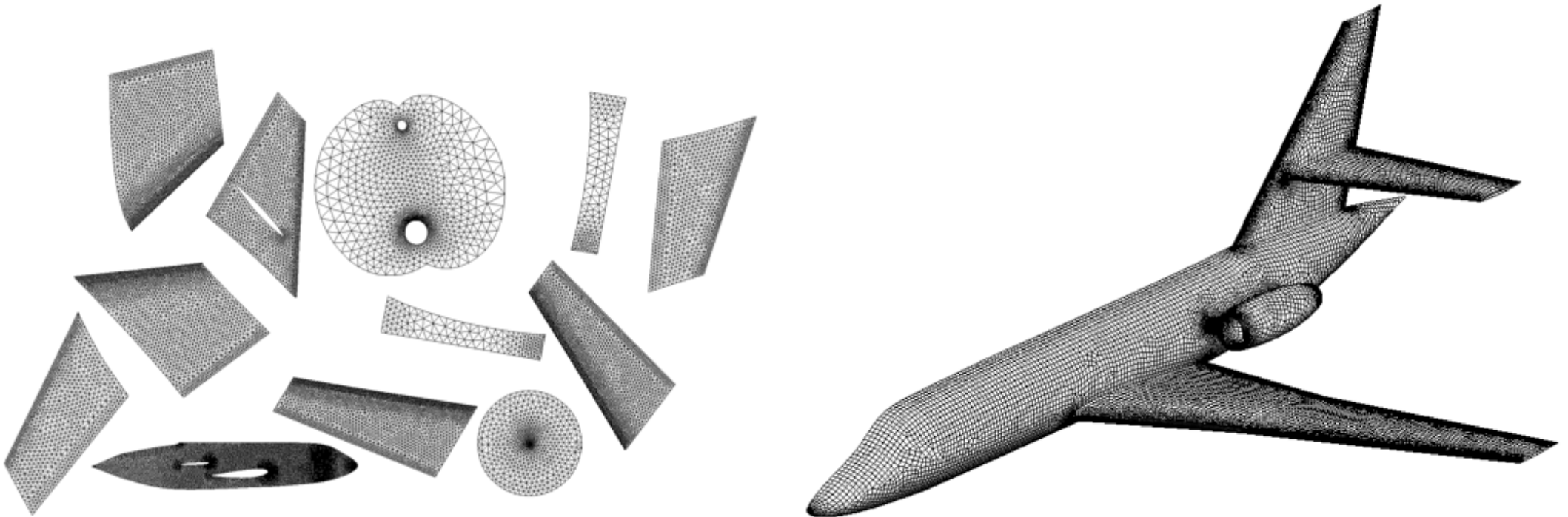
- Gmsh does not translate across CAD formats (there is no internal common geometrical representation)
- Rather, Gmsh directly accesses the native representation using each CAD kernel's own API



# Geometry module

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

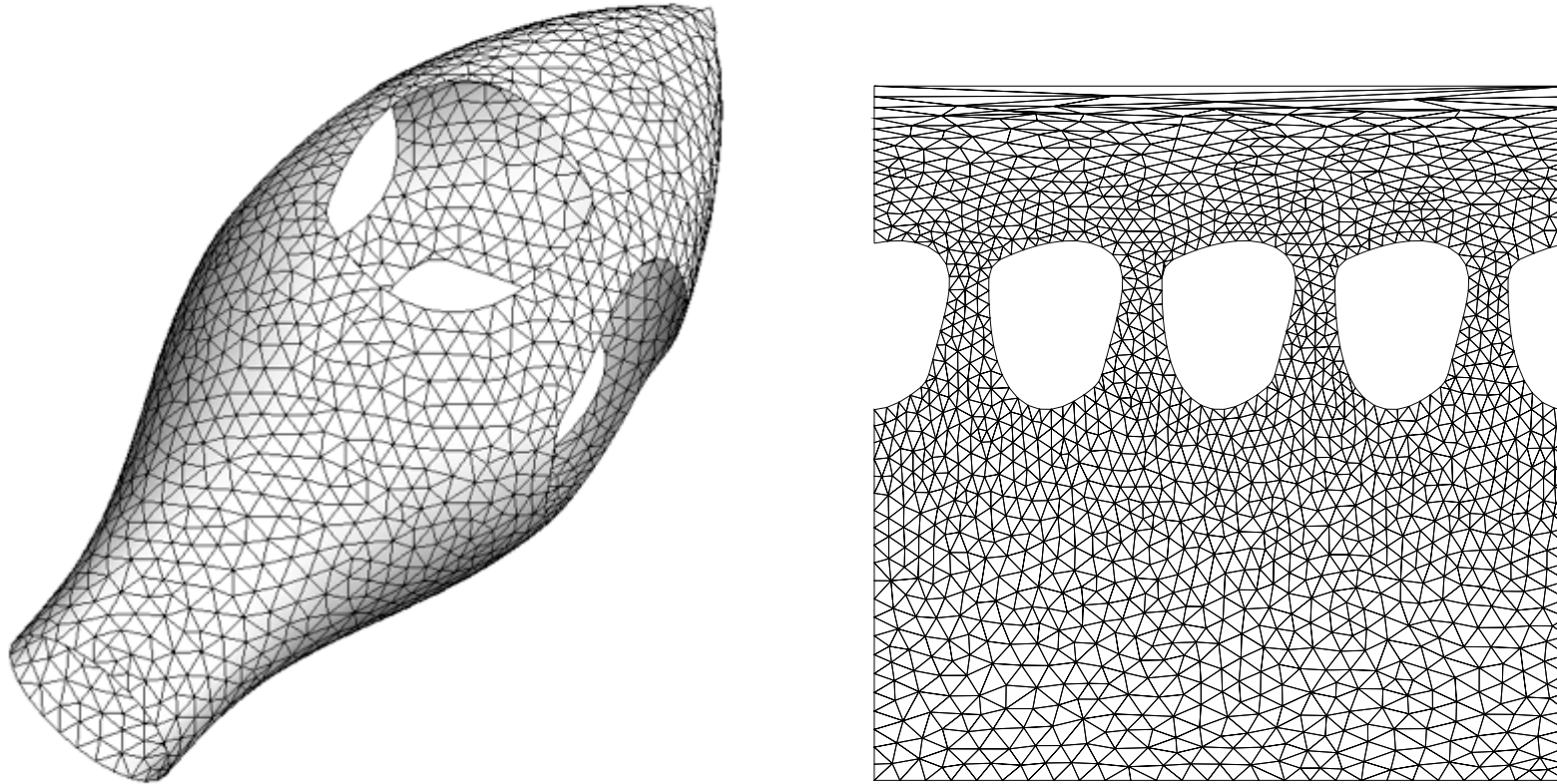
- They can be equipped with a geometry through a *reparametrization* procedure
- This reparametrization is used for remeshing



# 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)

## Mesh module



Typical CAD kernel idiosyncrasies: seam edges and degenerated edges



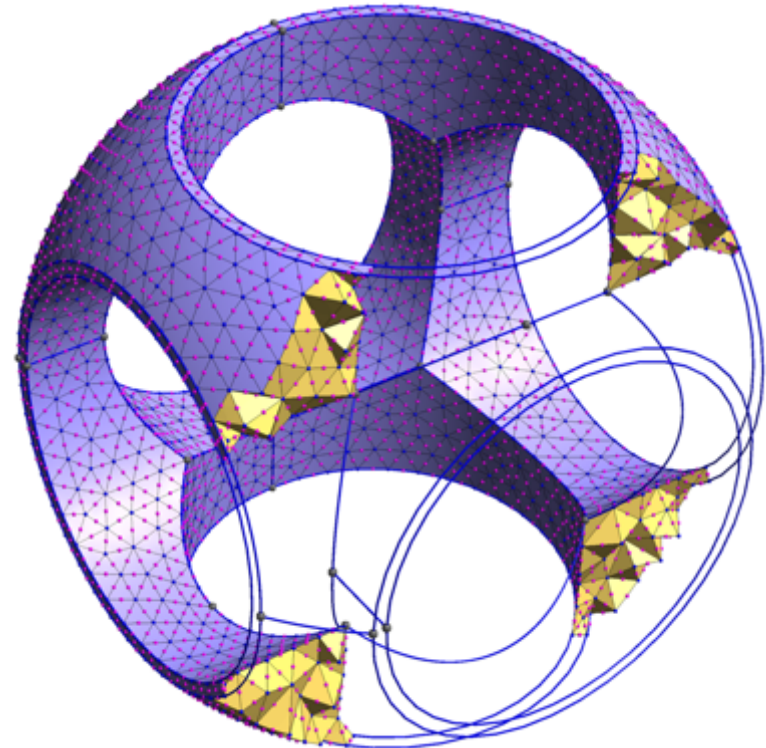
## Mesh module

- 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)

# Mesh module

- 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)

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

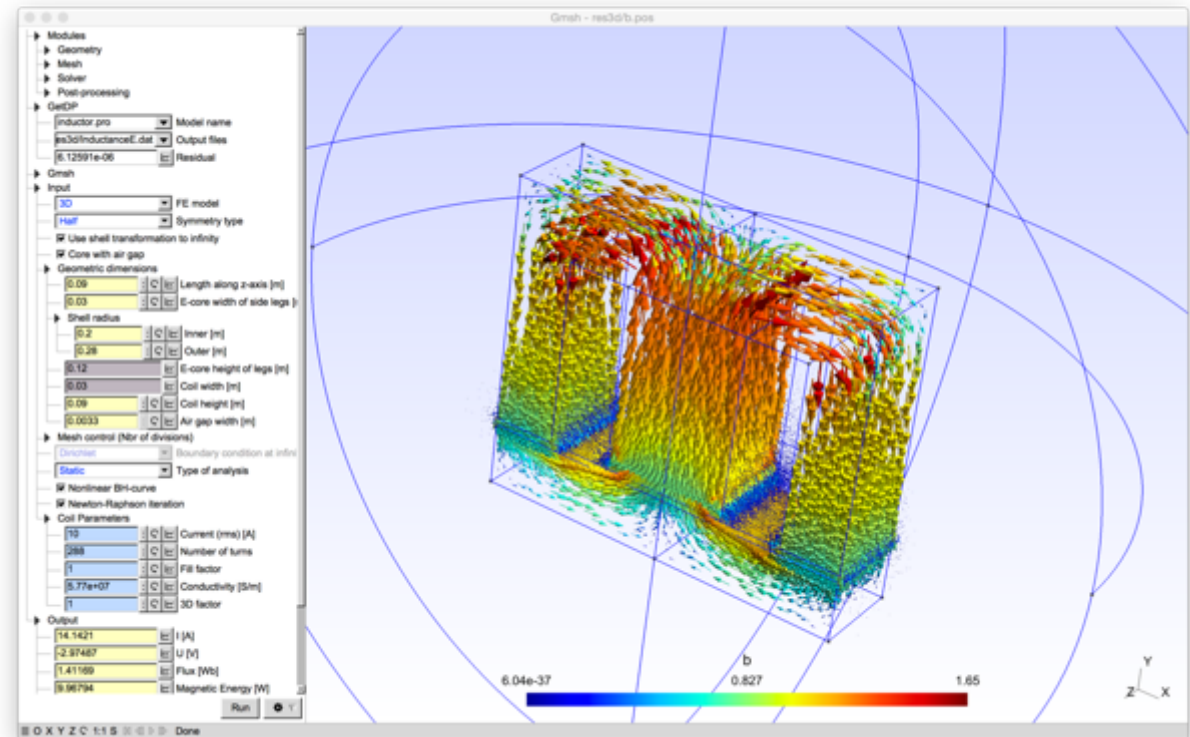




# Solver module

- Gmsh implements a ONELAB (<http://onelab.info>) server to pilot external solvers, called “clients”
- Example client: GetDP finite element solver (<http://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



## Solver module

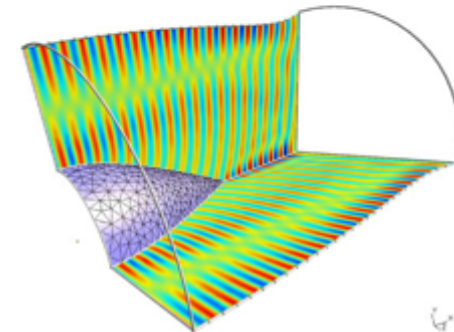
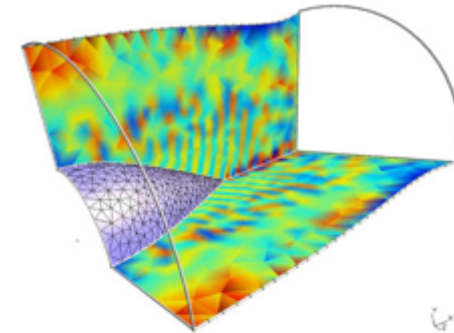
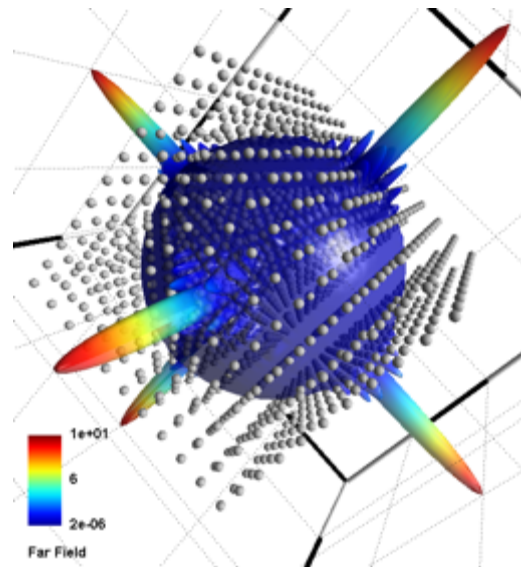
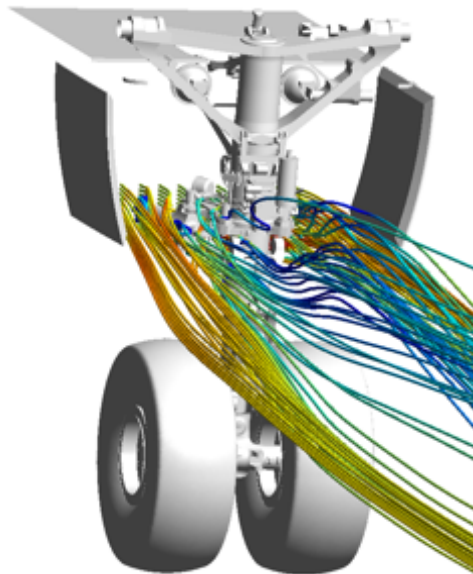
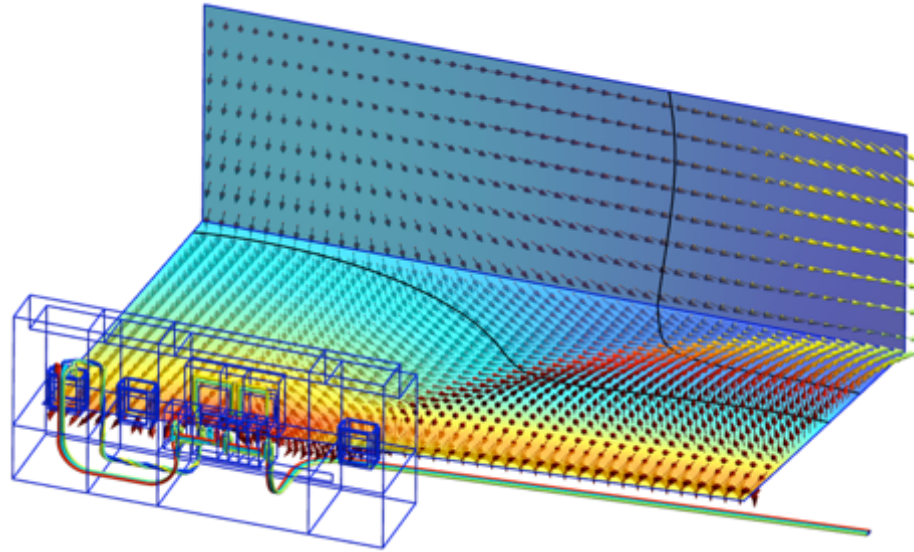
- The implementation is based on a client-server model, with a server-side database and local or remote clients communicating in-memory or through TCP/IP sockets
  - Contrary to most solver interfaces, the ONELAB server has no a priori knowledge about any specifics (input file format, syntax, ...) of 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

## 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 modify and create views

# Post-processing module

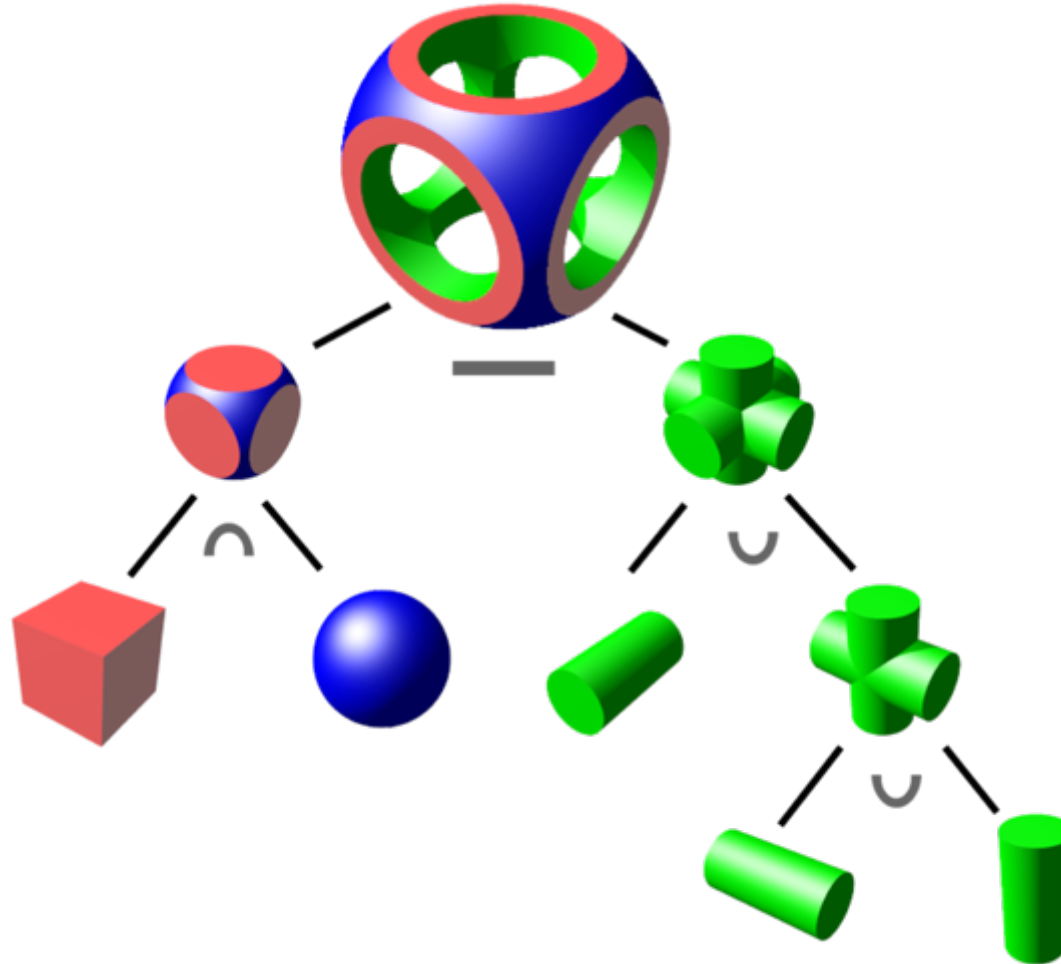
- Cuts, iso-curves and vectors
- Elevation maps
- Streamlines
- Adaptive high-order visualization



# Recent developments

- Constructive Solid Geometry
- Application Programming Interface (API)
- Parallel meshing
- Robust STL remeshing

# Constructive Solid Geometry



[http://en.wikipedia.org/wiki/Constructive\\_solid\\_geometry](http://en.wikipedia.org/wiki/Constructive_solid_geometry)

# Constructive Solid Geometry

Use OpenCASCADE CAD kernel

```
SetFactory("OpenCASCADE");
```

```
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};
```

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

```
BooleanIntersection(3) = { Volume{1}; Delete; }{ Volume{2}; Delete; };
```

```
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; };
```

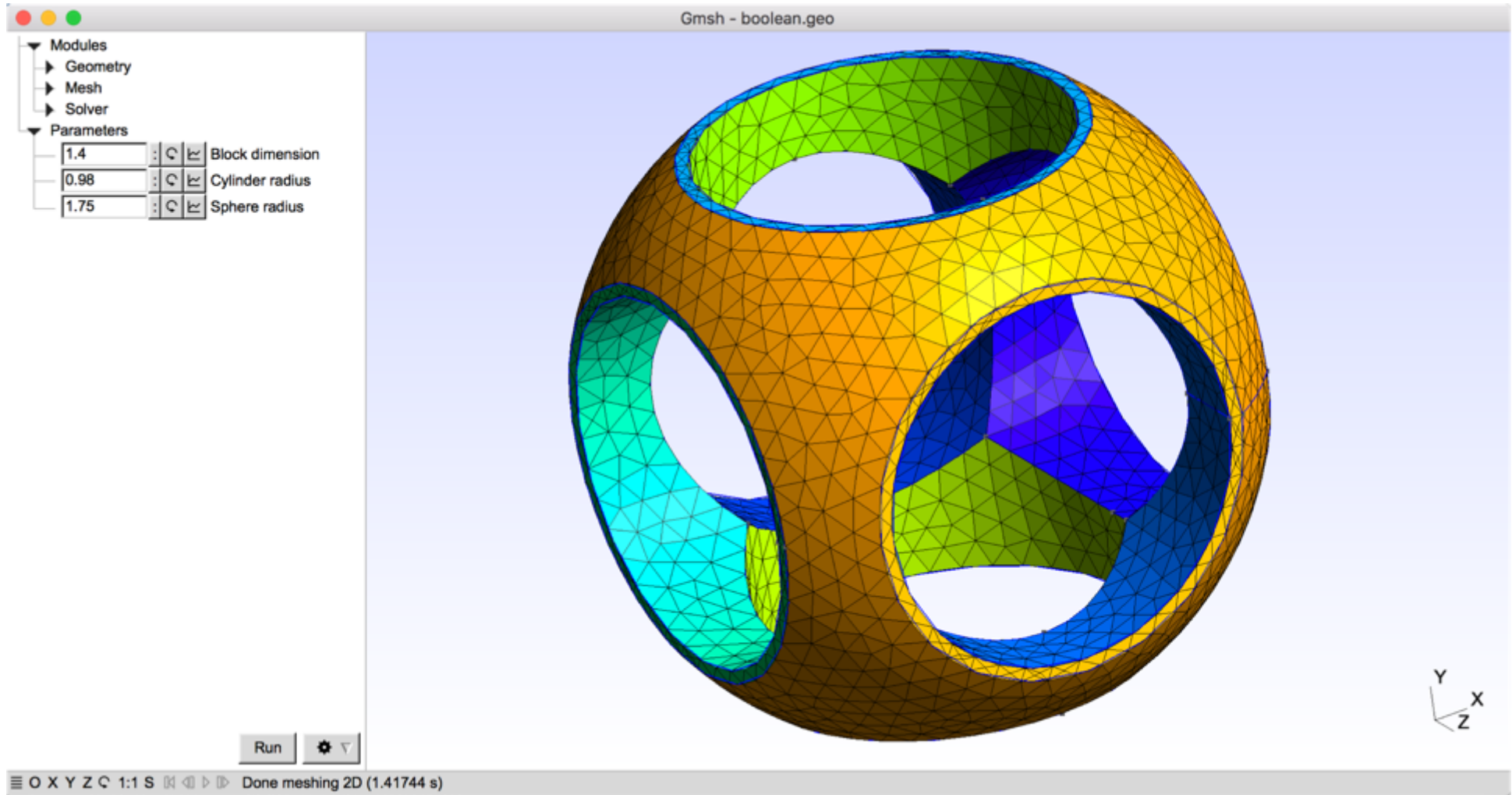
```
BooleanDifference(8) = { Volume{3}; Delete; }{ Volume{7}; Delete; };
```

Explicit tags

Delete object and tool



# Constructive Solid Geometry



[gmsh/demos/boolean/boolean.geo](https://gmsh.org/demos/boolean/boolean.geo)



# 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");
b() = 2;
Box(b(0)) = {0,156,z, 10,170,z+10};
```

← Import shapes from STEP or BREP

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

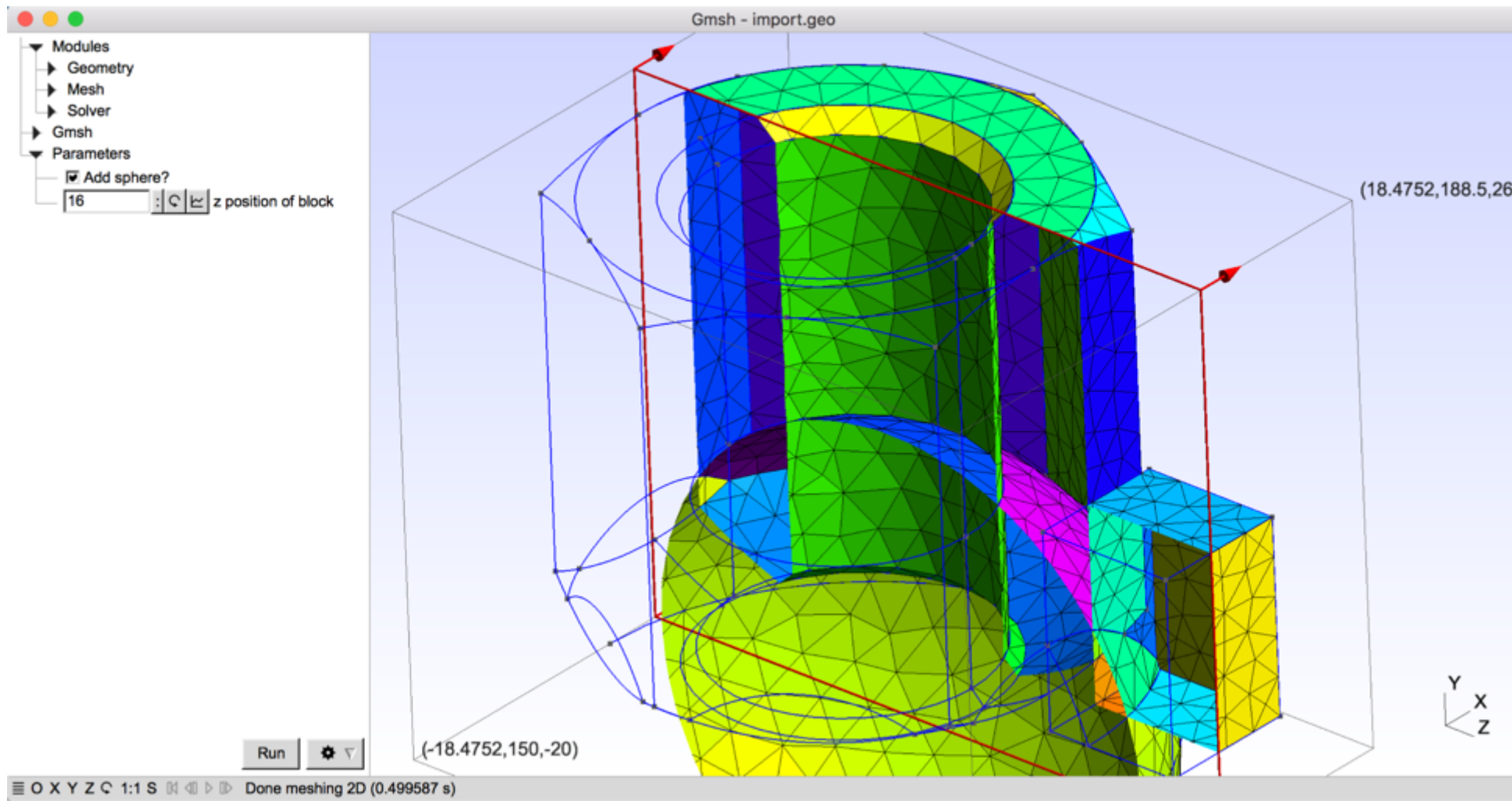
BooleanFragments intersects everything

```
r() = BooleanFragments{ Volume{a()}; Delete; }{ Volume{b()}; Delete; };
Save "merged.brep";
```

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

Implicit tags

# Constructive Solid Geometry



[gmsh/demos/boolean/import.geo](https://gmsh.org/doc/demos/boolean/import.geo)

# Constructive Solid Geometry

- All existing .geo commands are conserved
- New or modified .geo commands:
  - Shapes (with explicit numbering): Circle, Ellipse, Wire, Surface, Sphere, Box, Torus, Rectangle, Disk, Cylinder, Cone, Wedge, ThickSolid, ThruSections, Ruled ThruSections
  - Operations (implicit numbering): ThruSections, Ruled ThruSections, Fillet, Extrude
  - Boolean operations (explicit or implicit numbering): BooleanUnion, BooleanIntersection, BooleanDifference, BooleanFragments
  - Other: ShapeFromFile, Recursive Delete

# Application Programming Interface

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

# Application Programming Interface

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 😊

# Application Programming Interface

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

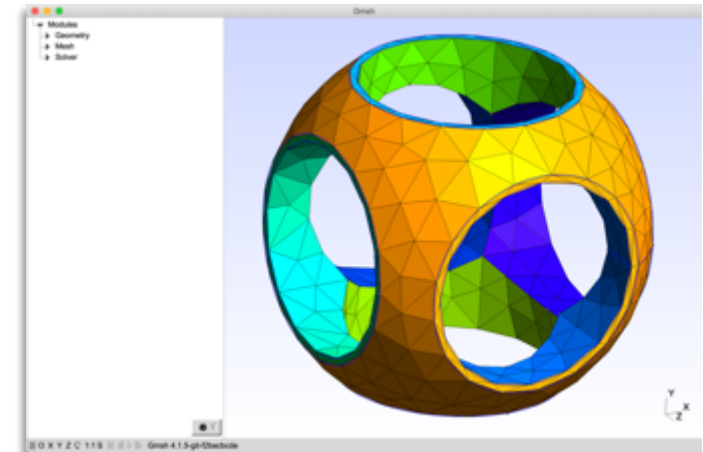
gmsh.model.occ.addBox(-R,-R,-R, 2*R,2*R,2*R, 1)
gmsh.model.occ.addSphere(0,0,0,Rt, 2)
gmsh.model.occ.intersect([(3, 1)], [(3, 2)], 3)
gmsh.model.occ.addCylinder(-2*R,0,0, 4*R,0,0, Rs, 4)
gmsh.model.occ.addCylinder(0,-2*R,0, 0,4*R,0, Rs, 5)
gmsh.model.occ.addCylinder(0,0,-2*R, 0,0,4*R, Rs, 6)
gmsh.model.occ.fuse([(3,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



[gmsh/demos/api/boolean.py](https://gmsh.org/doc/demos/api/boolean.py)

# Application Programming Interface

Or using the C++ API:

```
#include <gmsh.h>

int main(int argc, char **argv)
{
    gmsh::initialize(argc, argv);

    gmsh::model::add("boolean");

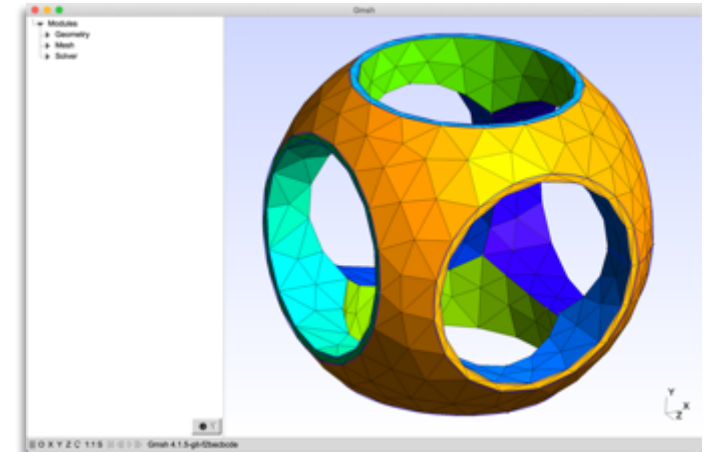
    double R = 1.4, Rs = R*.7, Rt = R*1.25;

    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;
}
```

[gmsh/demos/api/boolean.cpp](https://gmsh.info/demos/api/boolean.cpp)



# Application Programming Interface

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

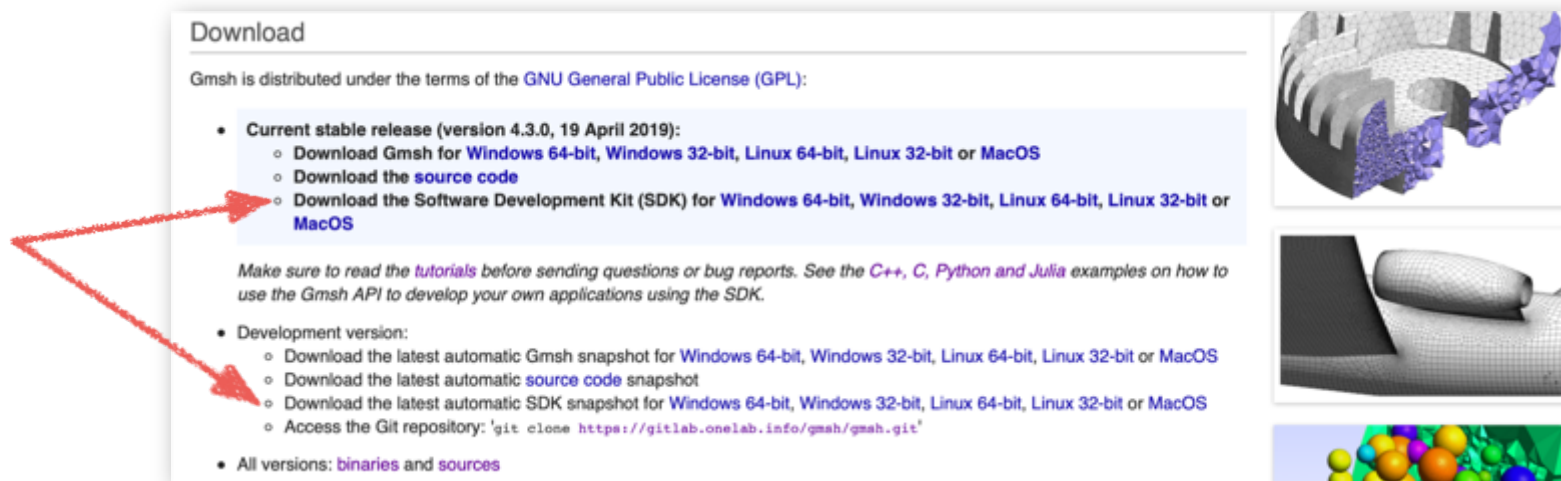
- Access mesh data (`getNodes`, `getElements`)
- Generate interpolation (`getBasisFunctions`) and integration (`getJacobians`) data to build Finite Element and related solvers (see e.g. [demos/api/poisson.py](#))
- Create post-processing views
- Run the graphical user-interface
- Build custom graphical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



# Application Programming Interface

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



**Download**

Gmsh is distributed under the terms of the [GNU General Public License \(GPL\)](#):

- **Current stable release (version 4.3.0, 19 April 2019):**
  - Download Gmsh for [Windows 64-bit](#), [Windows 32-bit](#), [Linux 64-bit](#), [Linux 32-bit](#) or [MacOS](#)
  - Download the [source code](#)
  - Download the Software Development Kit (SDK) for [Windows 64-bit](#), [Windows 32-bit](#), [Linux 64-bit](#), [Linux 32-bit](#) or [MacOS](#)

*Make sure to read the [tutorials](#) before sending questions or bug reports. See the [C++](#), [C](#), [Python](#) and [Julia](#) examples on how to use the Gmsh API to develop your own applications using the SDK.*

- **Development version:**
  - Download the latest automatic Gmsh snapshot for [Windows 64-bit](#), [Windows 32-bit](#), [Linux 64-bit](#), [Linux 32-bit](#) or [MacOS](#)
  - Download the latest automatic [source code](#) snapshot
  - Download the latest automatic SDK snapshot for [Windows 64-bit](#), [Windows 32-bit](#), [Linux 64-bit](#), [Linux 32-bit](#) or [MacOS](#)
  - Access the Git repository: `git clone https://gitlab.onelab.info/gmsh/gmsh.git`
- All versions: [binaries and sources](#)

The screenshot also features three 3D mesh models: a gear-like structure, a cylindrical object, and a cluster of colored spheres.

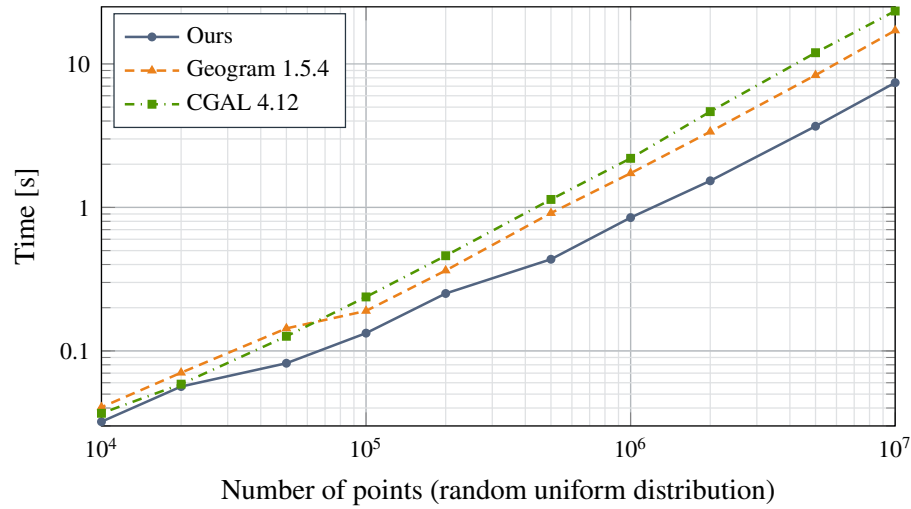
## Parallel Meshing

Most meshing algorithms are now multi-threaded using OpenMP:

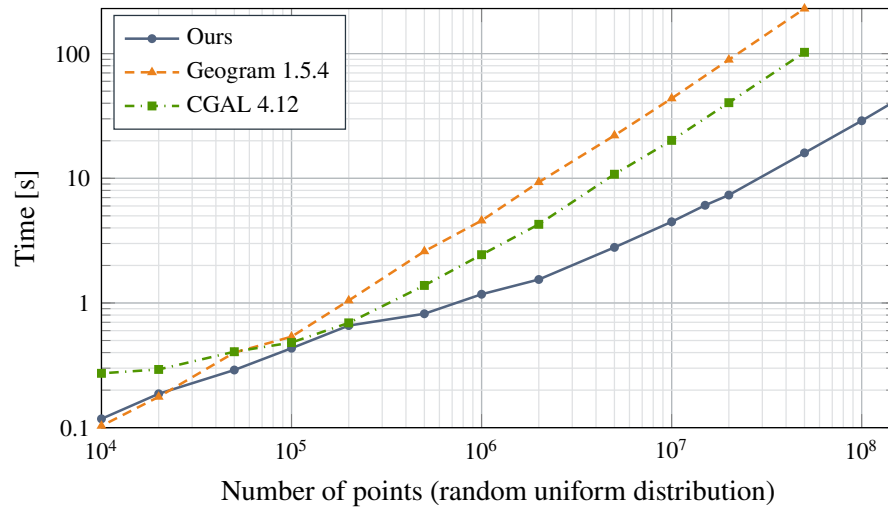
- 1D and 2D algorithms are multithreaded using coarse-grained approach, i.e. several curves/surfaces are meshed concurrently
- The new 3D Delaunay-based algorithm is multi-threaded using a fine-grained approach. It currently lacks several features (embedded entities, hybrid meshes, ...), which will eventually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

# Parallel Meshing

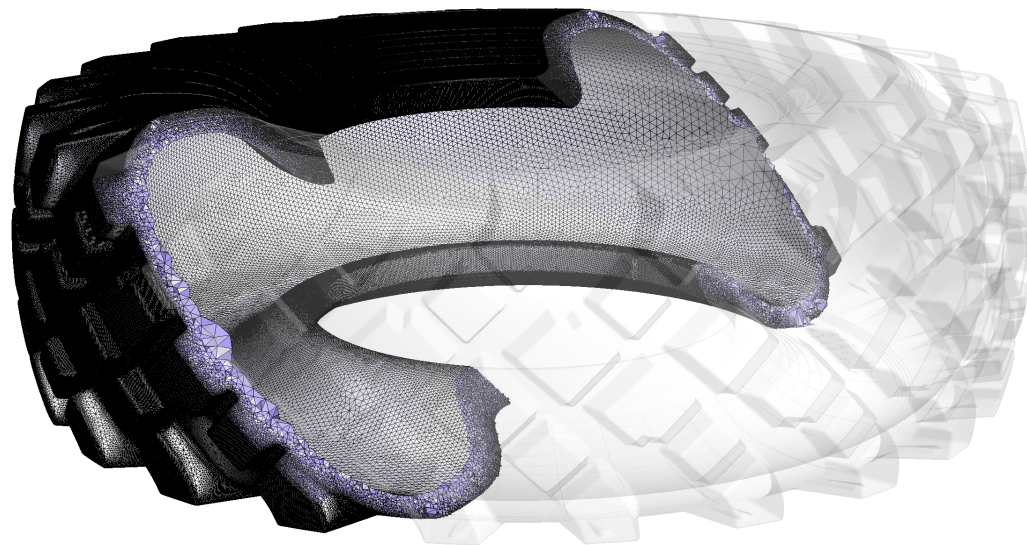


(a) 4-core Intel® Core™ i7-6700HQ CPU.



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

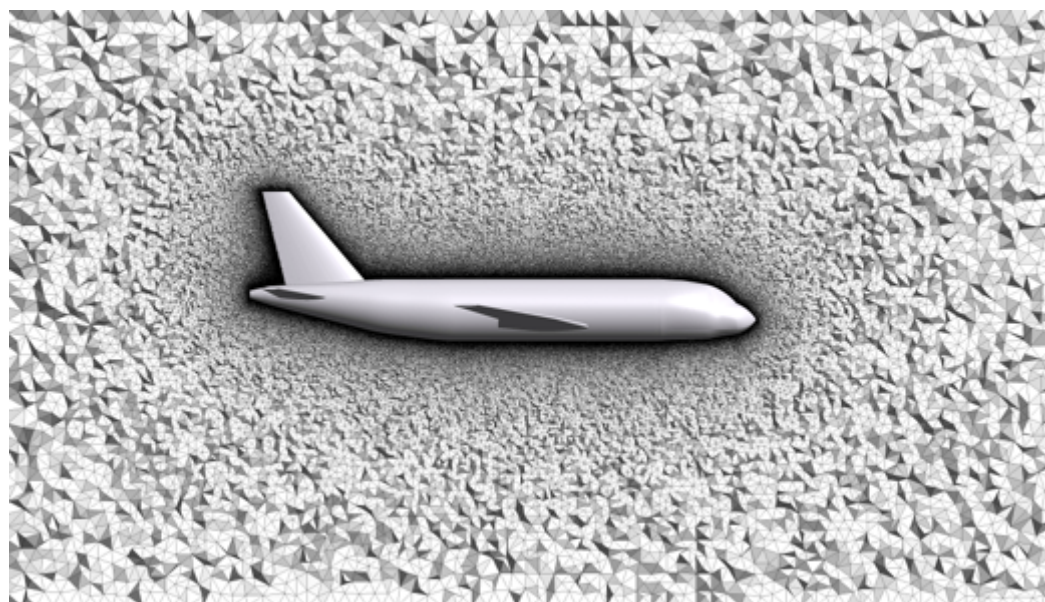
# Parallel Meshing



**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

AMD® EPYC 64 core



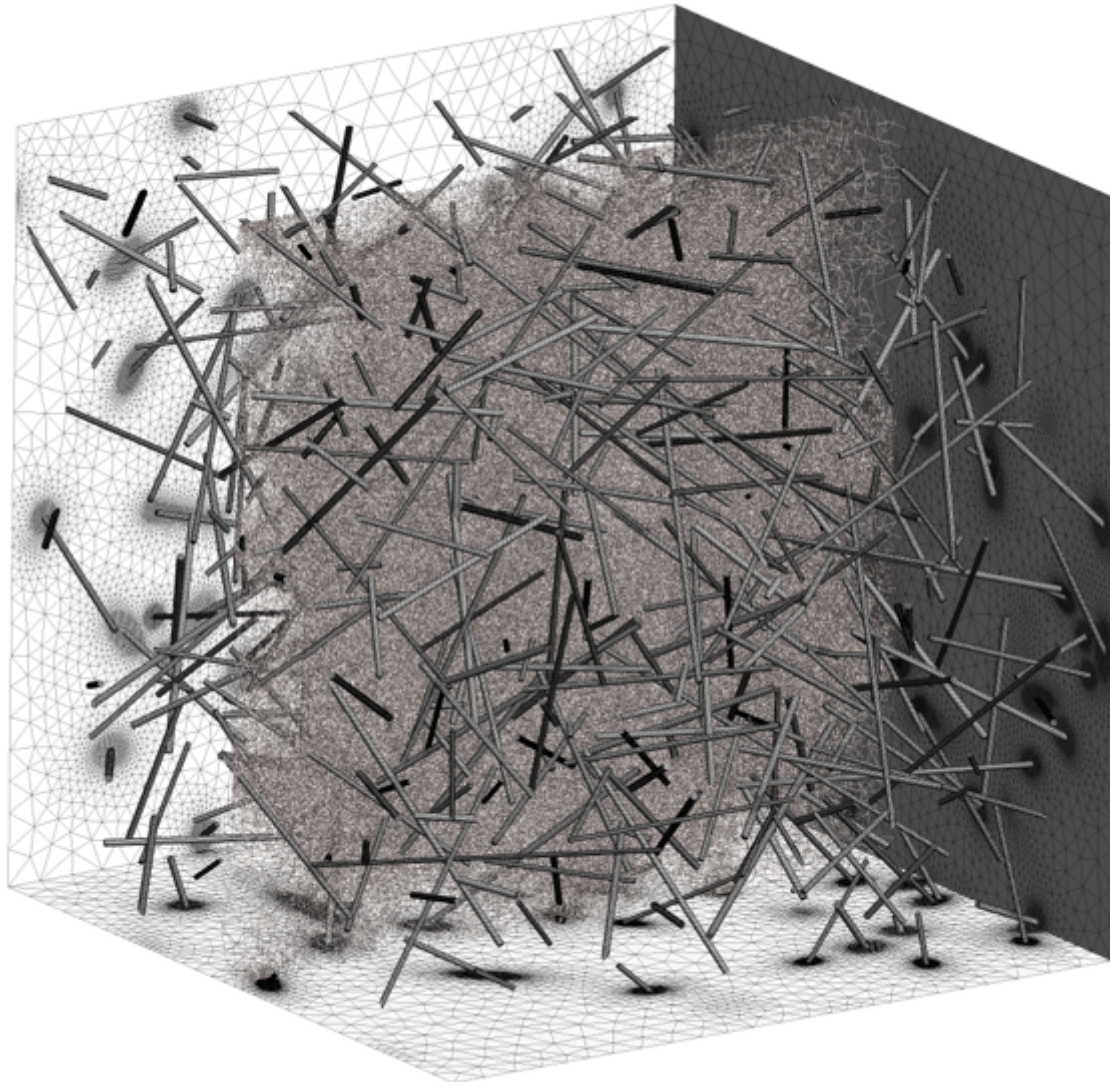
**Aircraft**

# 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 64 core



# Parallel Meshing



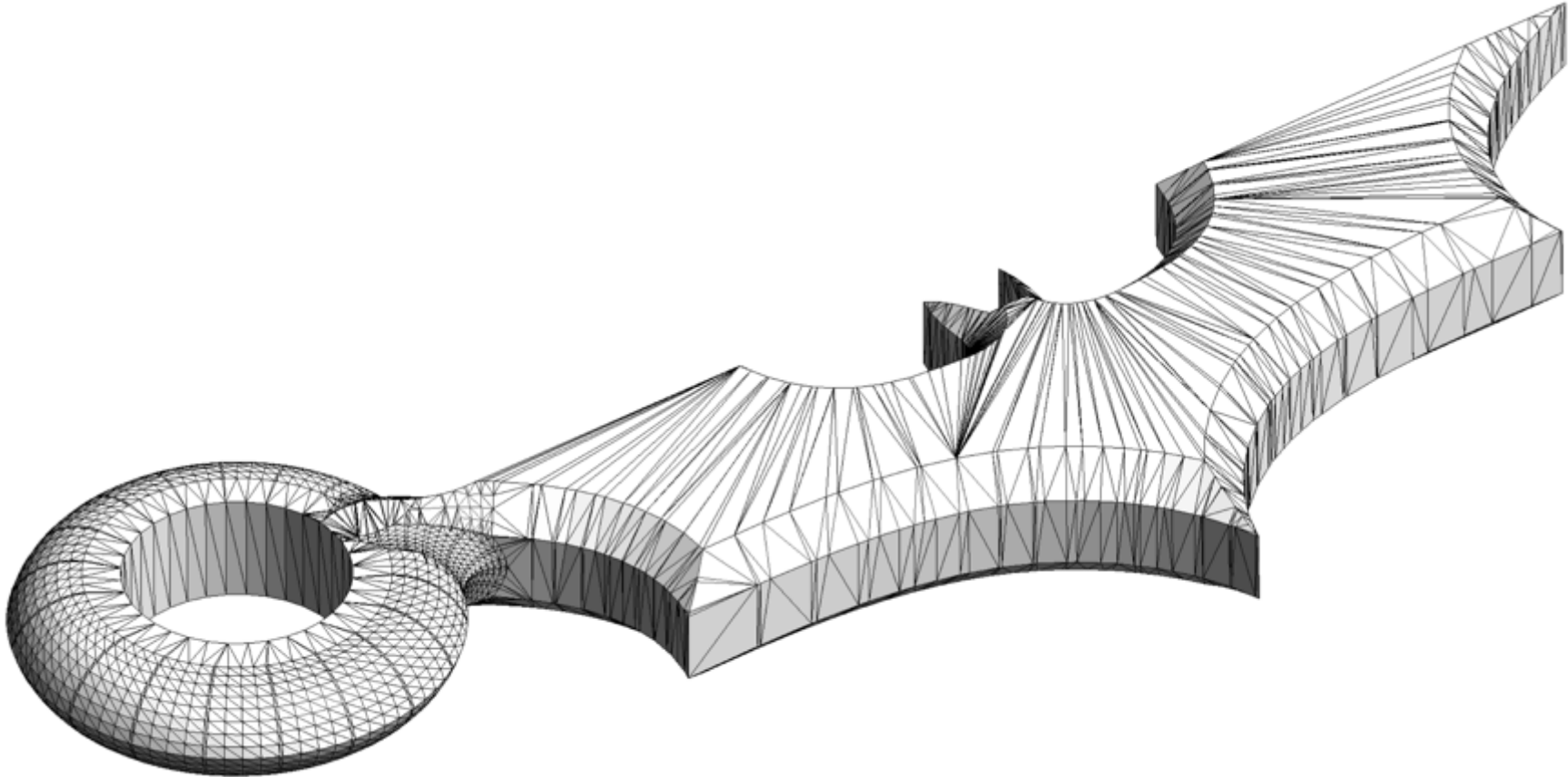
## 100 thin fibers

# 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

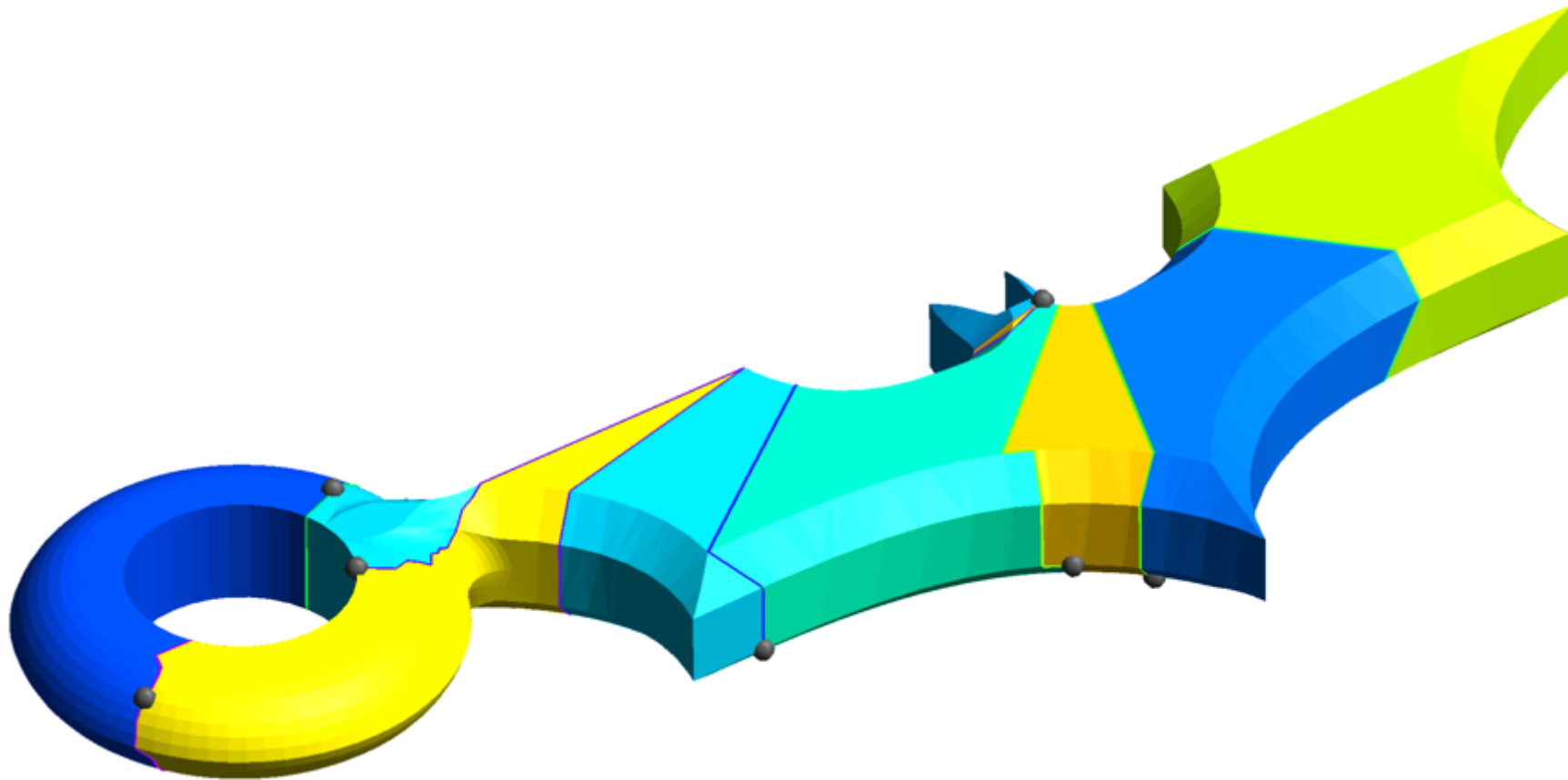
# threads	# tetrahedra	Timings (s)		
		BR	Refine	Total
1	723 208 595	18.9	1205.8	1234.4
2	723 098 577	16.0	780.3	804.8
4	722 664 991	86.6	567.1	659.8
8	722 329 174	15.8	349.1	370.1
16	723 093 143	15.6	216.2	236.5
32	722 013 476	15.6	149.7	169.8
64	721 572 235	15.9	119.7	140.4
127	721 591 846	15.9	114.2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

# Robust STL remeshing based on parametrizations



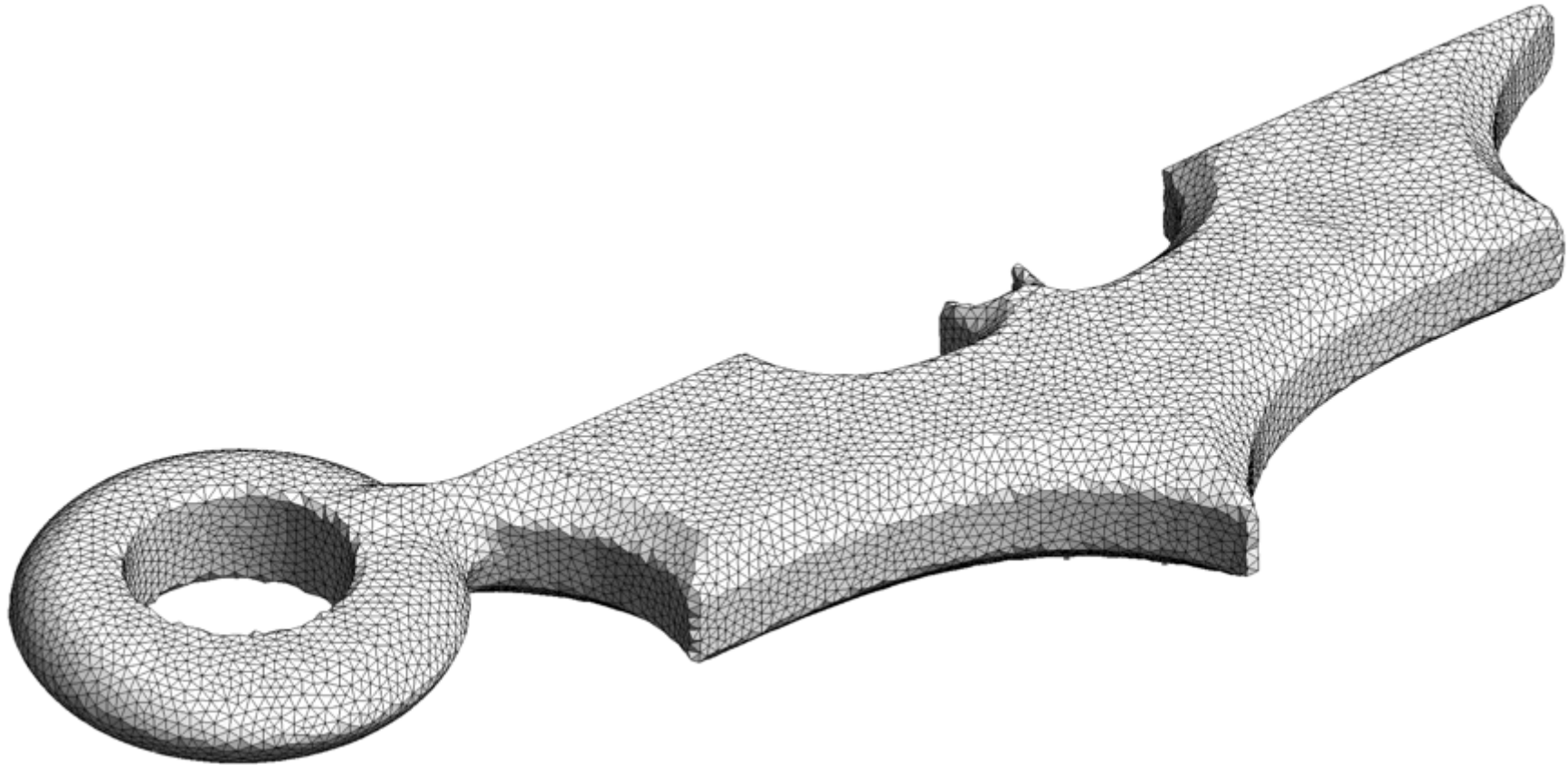
Batman STL mesh

# Robust STL remeshing based on parametrizations



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

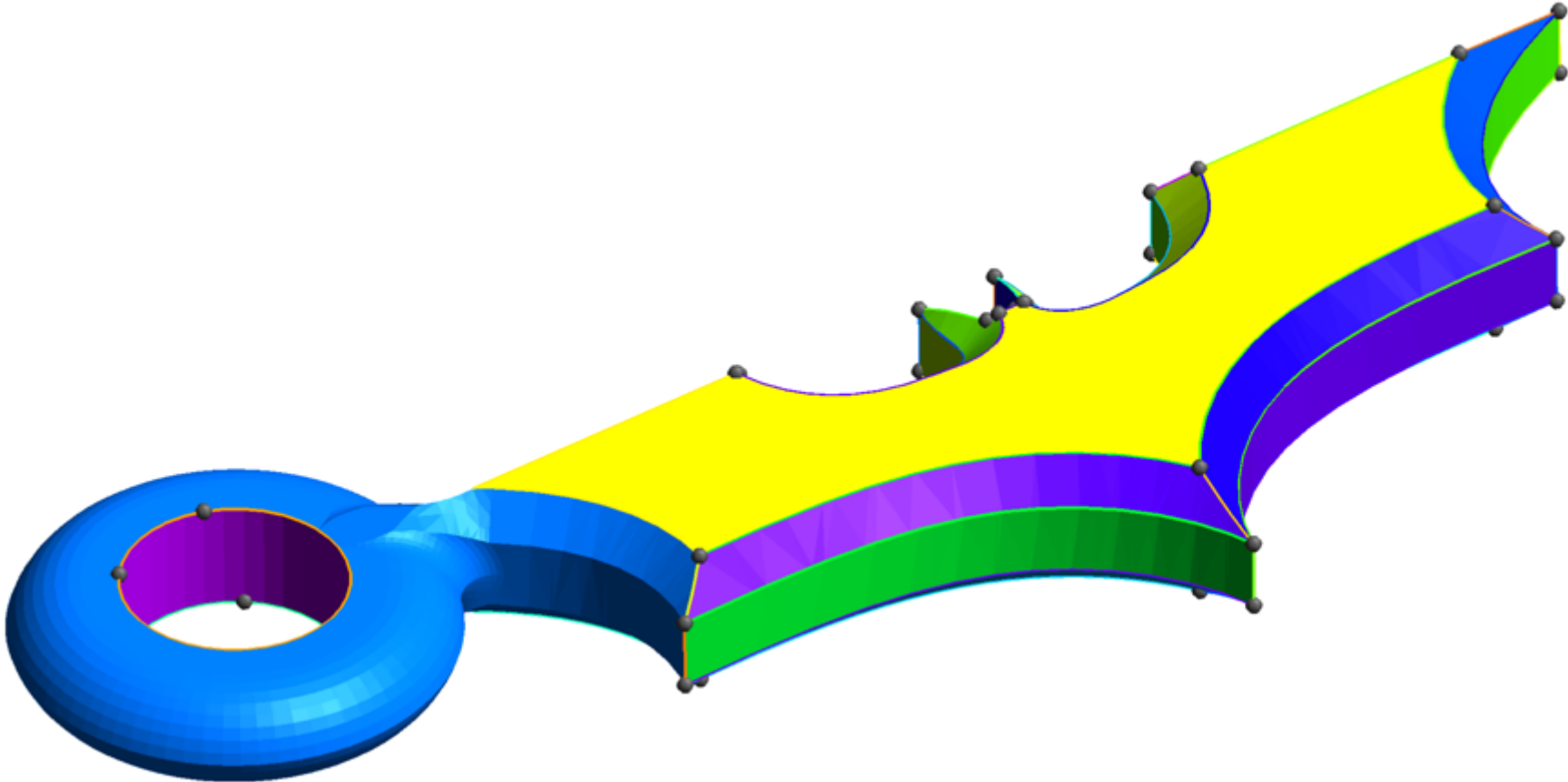
# Robust STL remeshing based on parametrizations



Remeshing

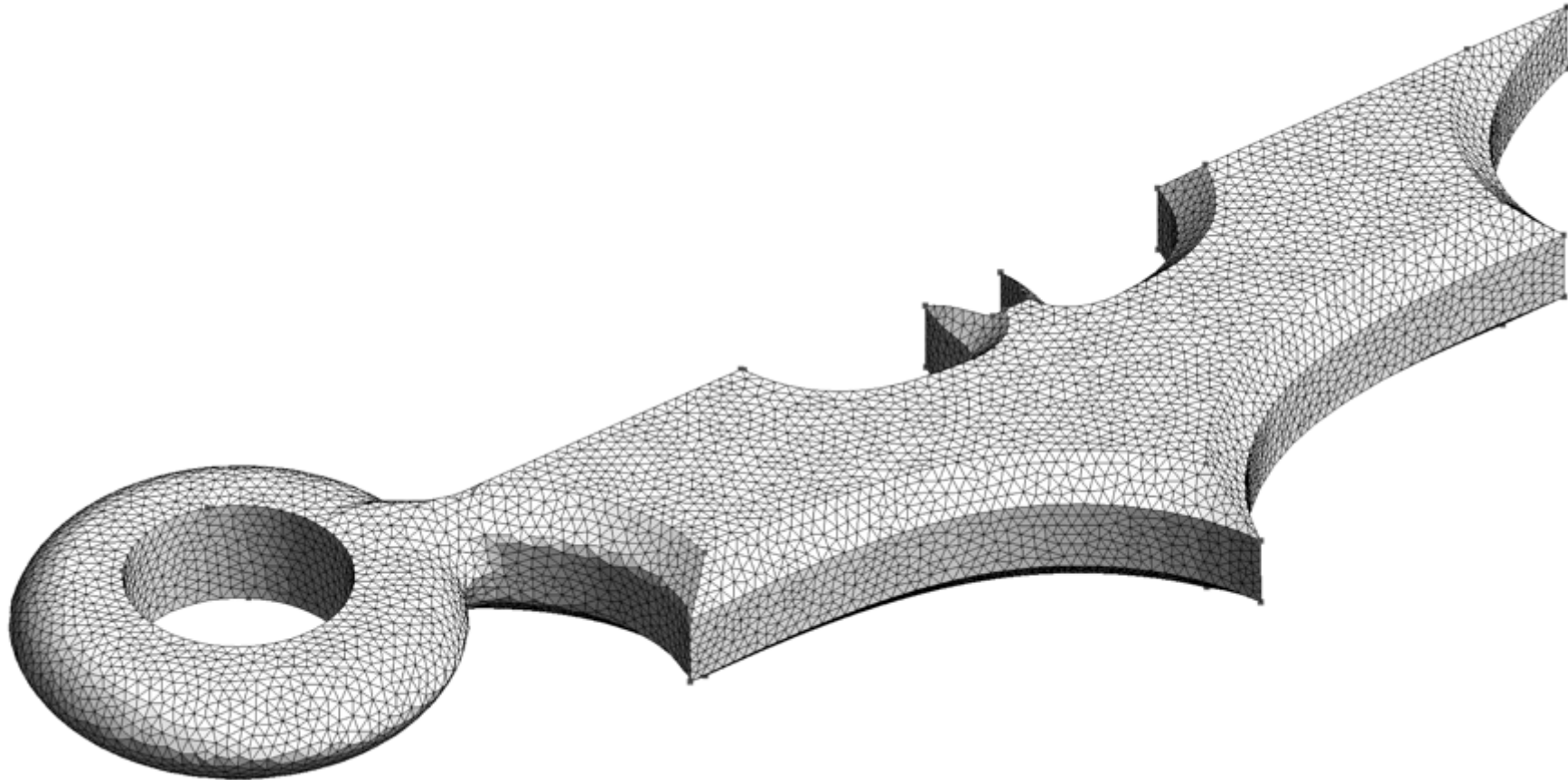


# Robust STL remeshing based on parametrizations



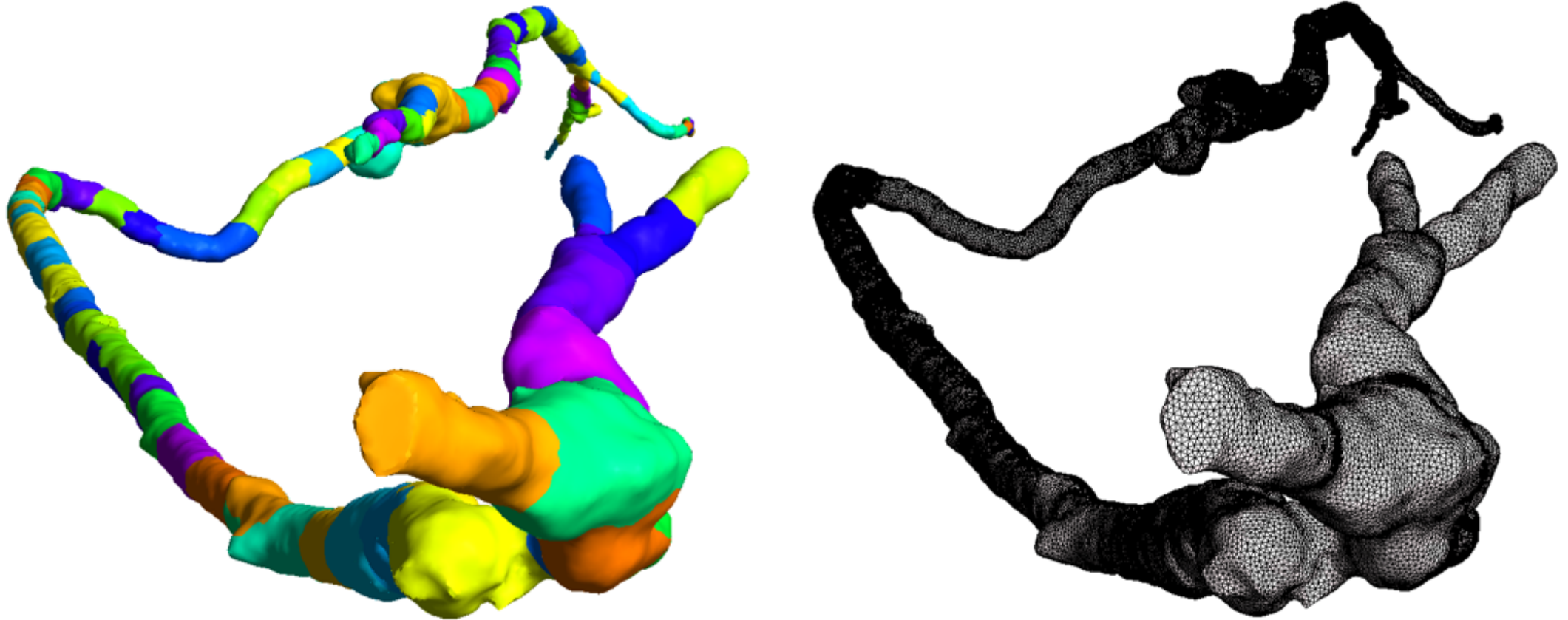
Automatic atlas creation, this time with feature edge detection

# Robust STL remeshing based on parametrizations



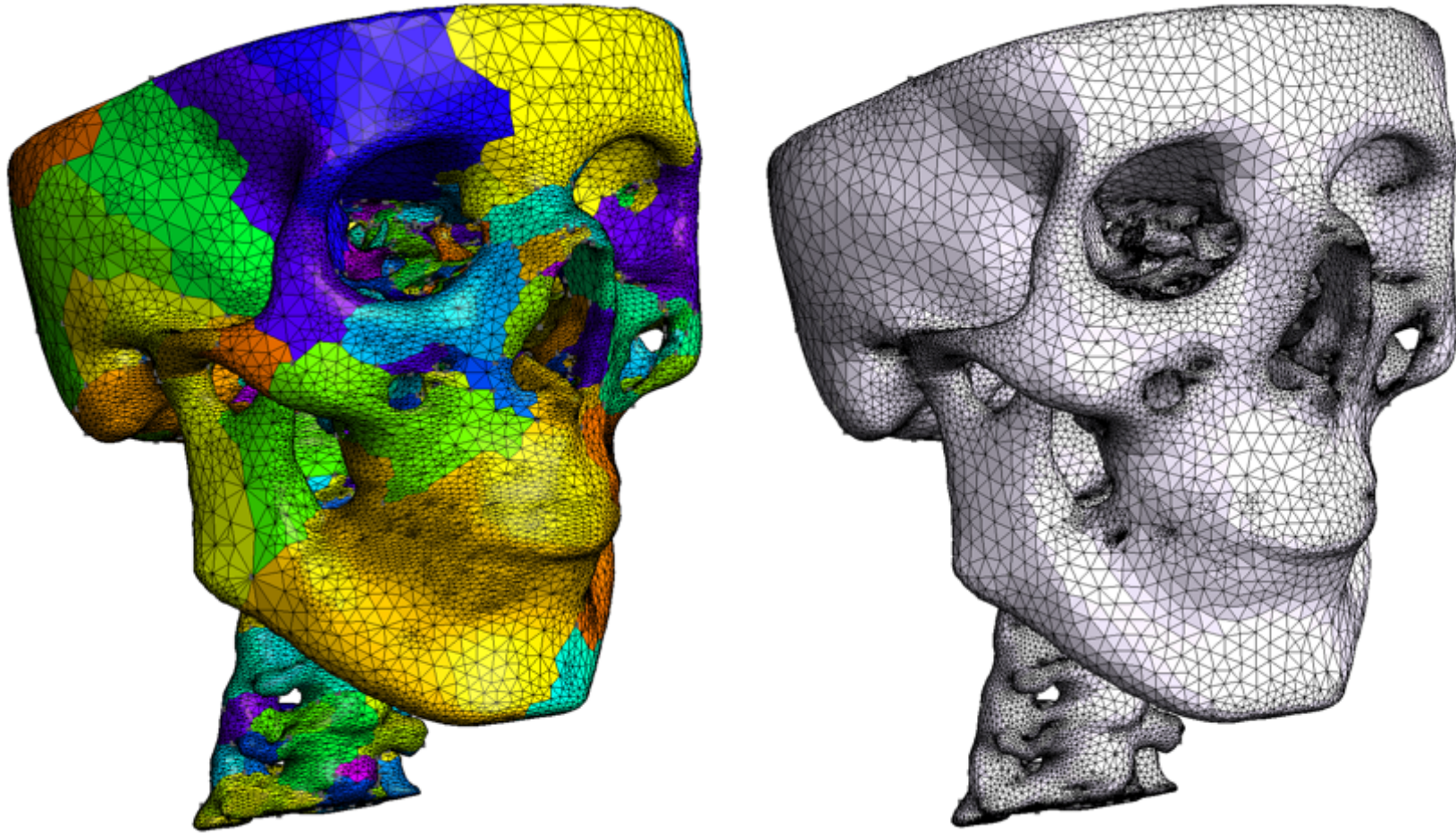
Remeshing with feature edge detection

# Robust STL remeshing based on parametrizations



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

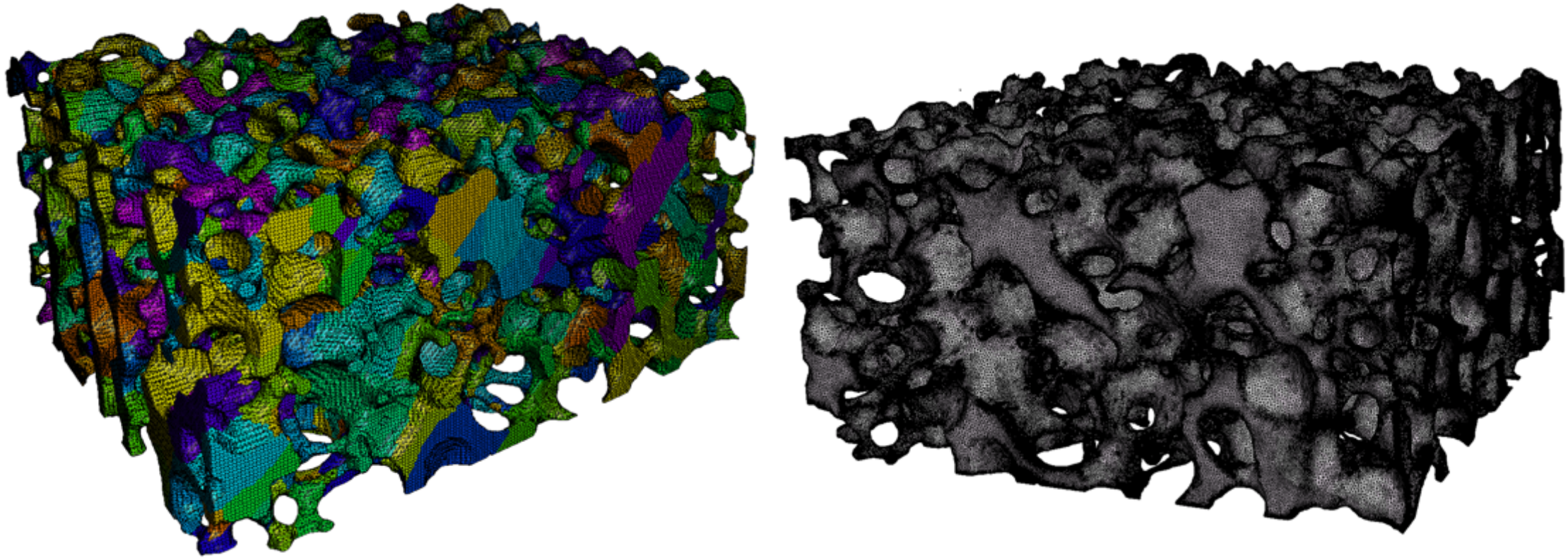
## Robust STL remeshing based on parametrizations



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



# Robust STL remeshing based on parametrizations



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

## Conclusions and perspectives

- Overview of Gmsh and recent developments:
  - Constructive Solid Geometry
  - Application Programming Interface
  - New parallel algorithms
  - Robust STL remeshing
- Many exciting developments in the pipeline:
  - Improved high-order remeshing
  - Hex-dominant meshes
  - (Semi-)automatic bloc-structured decompositions
  - Boundary layers

# Post-Scriptum

- To download Gmsh: <http://gmsh.info>
- For references, see <http://gmsh.info/#References>
- 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 + the finite element solver GetDP

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

