Towards (very) large scale finite element mesh generation with Gmsh

Christophe Geuzaine

Université de Liège

ELEMENT workshop, October 20, 2020
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.

What is Gmsh?

- Gmsh ([http://gmsh.info](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 (Jean-Francois Remacle from UCLouvain and myself); about 100 with $\geq 1$ commit
  - about 1,000 people on mailing lists
  - about 10,000 downloads per month (70% Windows)
  - about 500 citations per year – the main Gmsh paper is cited about 4,500 times
  - Gmsh has probably become one of the most popular (open source) finite element mesh generators?
~ 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: Gmsh paper and switch to CMake
- 2012: First curved meshing and quad meshing developments
- 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
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, ...)
Basic concepts

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

- Application Programming Interface (API)
- Multi-threaded meshing
- Robust STL remeshing based on parametrizations
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
A simple example written using the Python API:

```python
import gmsh

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

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

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`
... or using the C++ API:

```cpp
#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`
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. `gmsh/demos/api/poisson.py`)
- Create post-processing views
- Run the GUI, or build custom GUIs, e.g. for domain-specific codes (see `gmsh/demos/api/custom_gui.py`) or co-post-processing via ONELAB

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
Multi-threaded meshing

Meshing is 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.

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

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

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

[C. Marot et al., IJNME, 2019]

<table>
<thead>
<tr>
<th></th>
<th>$10^4$</th>
<th>$10^5$</th>
<th>$10^6$</th>
<th>$10^7$</th>
<th>$10^8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>0.032</td>
<td>0.13</td>
<td>0.85</td>
<td>7.40</td>
<td></td>
</tr>
<tr>
<td>Geogram</td>
<td>0.041</td>
<td>0.19</td>
<td>1.73</td>
<td>17.11</td>
<td>/</td>
</tr>
<tr>
<td>CGAL</td>
<td>0.037</td>
<td>0.24</td>
<td>2.20</td>
<td>23.37</td>
<td>/</td>
</tr>
</tbody>
</table>
## Multi-threaded meshing

### Truck tire

<table>
<thead>
<tr>
<th># threads</th>
<th># tetrahedra</th>
<th>Timings (s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BR</td>
<td>Refine</td>
<td>Total</td>
</tr>
<tr>
<td>1</td>
<td>123 640 429</td>
<td>75.9</td>
<td>259.7</td>
</tr>
<tr>
<td>2</td>
<td>123 593 913</td>
<td>74.5</td>
<td>166.8</td>
</tr>
<tr>
<td>4</td>
<td>123 625 696</td>
<td>74.2</td>
<td>107.4</td>
</tr>
<tr>
<td>8</td>
<td>123 452 318</td>
<td>74.2</td>
<td>95.5</td>
</tr>
</tbody>
</table>

### Aircraft

<table>
<thead>
<tr>
<th># threads</th>
<th># tetrahedra</th>
<th>Timings (s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BR</td>
<td>Refine</td>
<td>Total</td>
</tr>
<tr>
<td>1</td>
<td>672 209 630</td>
<td>45.2</td>
<td>1348.5</td>
</tr>
<tr>
<td>2</td>
<td>671 432 038</td>
<td>42.1</td>
<td>1148.9</td>
</tr>
<tr>
<td>8</td>
<td>665 826 109</td>
<td>39.6</td>
<td>714.8</td>
</tr>
<tr>
<td>64</td>
<td>664 587 093</td>
<td>38.7</td>
<td>322.3</td>
</tr>
<tr>
<td>127</td>
<td>663 921 974</td>
<td>38.1</td>
<td>255.0</td>
</tr>
</tbody>
</table>

AMD EPYC 2x 64-core
Multi-threaded meshing

### 100 thin fibers

<table>
<thead>
<tr>
<th># threads</th>
<th># tetrahedra</th>
<th>Timings (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BR</td>
</tr>
<tr>
<td>1</td>
<td>325,611,841</td>
<td>3.1</td>
</tr>
<tr>
<td>2</td>
<td>325,786,170</td>
<td>2.9</td>
</tr>
<tr>
<td>4</td>
<td>325,691,796</td>
<td>2.8</td>
</tr>
<tr>
<td>8</td>
<td>325,211,989</td>
<td>2.7</td>
</tr>
<tr>
<td>16</td>
<td>324,897,471</td>
<td>2.8</td>
</tr>
<tr>
<td>32</td>
<td>325,221,244</td>
<td>2.7</td>
</tr>
<tr>
<td>64</td>
<td>324,701,883</td>
<td>2.8</td>
</tr>
<tr>
<td>127</td>
<td>324,190,447</td>
<td>2.9</td>
</tr>
</tbody>
</table>

### 500 thin fibers

<table>
<thead>
<tr>
<th># threads</th>
<th># tetrahedra</th>
<th>Timings (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BR</td>
</tr>
<tr>
<td>1</td>
<td>723,208,595</td>
<td>18.9</td>
</tr>
<tr>
<td>2</td>
<td>723,098,577</td>
<td>16.0</td>
</tr>
<tr>
<td>4</td>
<td>722,664,991</td>
<td>86.6</td>
</tr>
<tr>
<td>8</td>
<td>722,329,174</td>
<td>15.8</td>
</tr>
<tr>
<td>16</td>
<td>723,093,143</td>
<td>15.6</td>
</tr>
<tr>
<td>32</td>
<td>722,013,476</td>
<td>15.6</td>
</tr>
<tr>
<td>64</td>
<td>721,572,235</td>
<td>15.9</td>
</tr>
<tr>
<td>127</td>
<td>721,591,846</td>
<td>15.9</td>
</tr>
</tbody>
</table>

AMD EPYC 2x 64-core
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
Robust STL remeshing

CT scan of an artery: automatic atlas creation – each patch is provably parametrizable by solving a linear PDE, using mean value coordinates
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
Towards (very) large scale mesh generation

**PARSEC** (PRACE 6iP 2020-2022): Parallel Adaptive Refinement for Simulations on Exascale Computers

Partners:
- Barcelona Supercomputing Center
- KTH Royal Institute of Technology
- Université of Liège
- Cenaero

Aim: Sharing best practices and collaboratively modernize the AMR implementation of three leading edge community codes (Alya, Nek5000, Argo), for the exploitation of future (pre-)Exascale machines
Towards (very) large scale mesh generation

Meshing, refinement, coarsening (including for high-order meshes)

Tools: Alya SFC partitioner, ParMETIS, MAdLib, Gmsh

Strategies:

- Classical “freeze and move interfaces” strategy (Gmsh kernel in Alya and MAdLib; Gmsh parallel partition topology as BRep)
- Proof-of-concept “single step” using coarse partition and remeshing of discrete interfaces
- Particular focus on high-order meshes in Gmsh-MAdLib coupling

Looking forward to sharing results/code with ExCALIBUR!
Thank you

cgeuzaine@uliege.be