Yet another Gmsh introduction

Nicolas Marsic

TECHNISCHE UNIVERSITÄT DARMSTADT

#> gmsh -help

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gmsh is:
  - a 3D finite element mesh generator
  - a CAD engine
  - a post-processing tool

  - a free software ⟶ GPL license with exceptions
  - quite mature now ⟶ started in 1997
  - developed mainly by Christophe Geuzaine and Jean-François Remacle

  - available for Windows, Mac OS, Linux, iOS and Android
  - downloaded about 5000 times per week
  - available at http://gmsh.info

`#> gmsh -help | more`

TECHNISCHE UNIVERSITÄT DARMSTADT

- `Gmsh` has:
  - a (not so bad) documentation ⟶ `http://gmsh.info/doc/texinfo/gmsh.html`
  - a (quite active) mailing list ⟶ `http://onelab.info/mailman/listinfo/gmsh/`

- `Gmsh` can be used:
  - through a `GUI`
  - in command line through `Gmsh`'s own scripting language

#> gmsh -help | more

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gmsh has:
  - a (not so bad) documentation ⟶ http://gmsh.info/doc/texinfo/gmsh.html
  - a (quite active) mailing list ⟶ http://onelab.info/mailman/listinfo/gmsh/

- Gmsh can be used:
  - through a GUI
  - in command line through Gmsh's own scripting language

I this presentation I will focus on the CLI and the .geo language

#> emacs HelloWorld.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- The geometry can be described either through the GUI or by a .geo file
  → actually, the GUI is writing a .geo file!
  → it is possible to mix the two approaches...
- Here are the most elementary commands:
  - `Point(id) = {x, y, z, cl};` → `cl` = mesh element size at this point
  - `Line(id) = {start, end};`
  - `Circle(id) = {start, center, end};` → circle arc (1D)!
  - `Spline(id) = {control_point_1, ...};`
  - `BSpline(id) = {control_point_1, ...};`
- Surfaces are created from 1D loops (`Line` or `Circle`) or by extrusion
- Volumes are created from 2D loops or by extrusion
- No 2D or 3D elementary entity available (at least with `Gmsh`'s own engine)

#> emacs HelloWorld.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- The geometry can be described either through the GUI or by a .geo file
  → actually, the GUI is writing a .geo file!
  → it is possible to mix the two approaches...
- Here are the most elementary commands:
  - `Point(id) = {x, y, z, cl};` → `cl` = mesh element size at this point
  - `Line(id) = {start, end};`
  - `Circle(id) = {start, center, end};` → circle arc (1D)!
  - `Spline(id) = {control_point_1, ...};`
  - `BSpline(id) = {control_point_1, ...};`
- Surfaces are created from 1D loops (`Line` or `Circle`) or by extrusion
- Volumes are created from 2D loops or by extrusion
- No 2D or 3D elementary entity available (at least with `Gmsh`'s own engine)

Let's write our first .geo file!

#> emacs HelloUniverse.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- The following commands can be also quite useful:
  - `Include "...";`
  - `Delete{...};`
  - `newp;`, `newl;`, `newll;`, ... → new `Point`, `Line`, `Line Loop`, ... id
  - `Boundary{...};`
  - `CombinedBoundary{...};` → same as `Boundary` but for complex parts
  - `Translate{...};`
  - `Duplicata{...};` → to be used in transformations such as `Translate`

#> emacs HelloUniverse.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- The following commands can be also quite useful:
  - `Include "...";`
  - `Delete{...};`
  - `newp;`, `newl;`, `newll;`, ... → new `Point`, `Line`, `Line Loop`, ... id
  - `Boundary{...};`
  - `CombinedBoundary{...};` → same as `Boundary` but for complex parts
  - `Translate{...};`
  - `Duplicata{...};` → to be used in transformations such as `Translate`

Let's try to insert our cylinder in a box!

#> emacs boubouchons/boubouchons.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- To conclude the `.geo` file introduction, let us add:
  - `For`-style loop is available
  - `If`-style control is available
  - `variable~{i}` is a short cut for `variable_i` ⟶ can be useful to loop on `i`!
  - this can be mixed with arrays: `array~{i}[j]`
  - macros can be created (fairly new)
  - all variables are global!

`Merge "HelloCST.stp";`

TECHNISCHE UNIVERSITÄT DARMSTADT

- In addition to its own `CAD` engine, `Gmsh` is linked with `OpenCASCADE` (`OCC`):
  - `Gmsh` can read or write a lot of `CAD` file, such as `.brep` for instance
  - It is possible to drive a part of the `OCC` engine through `python` bindings

    → Ask me for more details if you are interested
  - Interactions between the two engines are possible → but limited!

    → It is possible to include your model in a box to model its surrounding

`Merge "HelloCST.stp";`

TECHNISCHE UNIVERSITÄT DARMSTADT

- In addition to its own `CAD` engine, `Gmsh` is linked with `OpenCASCADE` (`OCC`):
  - `Gmsh` can read or write a lot of `CAD` file, such as `.brep` for instance
  - It is possible to drive a part of the `OCC` engine through `python` bindings

    → Ask me for more details if you are interested
  - Interactions between the two engines are possible → but limited!

    → It is possible to include your model in a box to model its surrounding

Let's try to import a geometry from `CST`

#> gmsh HelloWorld.geo -2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gmsh is also a 1D, 2D and 3D conforming mesh generator
  - → In GUI just hit 1, 2 or 3
  - → In CLI use -1, -2 or -3
- In 2D, Gmsh implements the following algorithm:
  1. MeshAdapt
  2. Auto → default choice
  5. Delaunay
  6. Frontal
  7. Bamg → good for anisotropic meshes
  8. Delaunay for quads → good for further recombinations into quads
  9. Packing of parallelograms → idem
  - → Can be set in .geo file with Mesh.Algorithm = num;
  - → Can be set in GUI in Tools/Options/Mesh/General
  - → Can be set in CLI with -algo <string> (see -help)

#> gmsh HelloWorld.geo -3

TECHNISCHE UNIVERSITÄT DARMSTADT

- In 3D, Gmsh implements the following algorithm:
  1. Delaunay ⟶ default choice
  2. New Delaunay
  4. Frontal
  5. Frontal Delaunay
  6. Frontal Hex ⟶ good for further recombinations into hexes
  7. MMG3d ⟶ good for anisotropic meshes
  9. R-tree ⟶ good for further recombinations into hexes

  ⟶ Can be set in .geo file with `Mesh.Algorithm3D = num;`

  ⟶ Can be set in GUI in Tools/Options/Mesh/General

  ⟶ Can be set in CLI with -algo <string> (see -help)

#> emacs HelloHex.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- `Gmsh` can generate and recombine simplices into quads or hexes:
  1. it uses the Blossom algorithm
  2. just add `Recombine Surface "*";` or `Surface {ids, ...};` for 2D
     - → a 100% quadrangular mesh is technically guaranteed
     - → don't forget to try the different algorithm to find the best mesh
     - → `Packing of parallelograms` could be a good start...
  3. just add `Recombine Volume "*";` or `Volume {ids, ...};` for 3D
     - → should create a hex-dominant mesh...
     - → highly experimental: use at your own risk ;-)!
     - → be ready for some seg. faults...
  4. don't mix `Recombine Volume` and `Recombine Surface`
     - → `Recombine Volume` implies `Recombine Surface` ...

#> emacs HelloHex.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- `Gmsh` can generate and recombine simplices into quads or hexes:
  1. it uses the Blossom algorithm
  2. just add `Recombine Surface "*";` or `Surface {ids, ...};` for 2D
     - → a 100% quadrangular mesh is technically guaranteed
     - → don't forget to try the different algorithm to find the best mesh
     - → `Packing of parallelograms` could be a good start...
  3. just add `Recombine Volume "*";` or `Volume {ids, ...};` for 3D
     - → should create a hex-dominant mesh...
     - → highly experimental: use at your own risk ;-)!
     - → be ready for some seg. faults...
  4. don't mix `Recombine Volume` and `Recombine Surface`
     - → `Recombine Volume` implies `Recombine Surface` ...

Let's have a look at the 3D automatic hex mesh

#> emacs HelloTransfinite.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gmsh can also generate hex or quad mesh using transfinites:
  - the number of points and the progression to use on each line can be specified
    - → `Transfinite Line "*" = points;`
    - → `Transfinite Line {lines_id, ...} = points;`
  - surfaces and volumes must also be transfinite
    - → `Transfinite Surface "*";` or `Transfinite Volume {id, ...};` or ...
    - → a surface must have four corners
    - → a volume must have five or six faces
  - triangles or tets can then be recombined into quads with `Recombine`

- It is also possible to create a structured by extrusion with `Layers`

#> emacs HelloTransfinite.geo

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gmsh can also generate hex or quad mesh using transfinites:
  - the number of points and the progression to use on each line can be specified
    - → `Transfinite Line "*" = points;`
    - → `Transfinite Line {lines_id, ...} = points;`
  - surfaces and volumes must also be transfinite
    - → `Transfinite Surface "*";` or `Transfinite Volume {id, ...};` or ...
    - → a surface must have four corners
    - → a volume must have five or six faces
  - triangles or tets can then be recombined into quads with `Recombine`

- It is also possible to create a structured by extrusion with `Layers`

Let's try it!

`#> gmsh HelloWorld.geo -3 -order 2 -optimize -optimize_ho`

TECHNISCHE UNIVERSITÄT DARMSTADT

- Curved elements can be generated easily:
  - call `Gmsh` with `-order int`
  - or add `Mesh.ElementOrder = int;` in your `.geo` file
  - or go into `Tools/Options/Mesh/General` in the `GUI`

  → up to now, the `GetDP` solver cannot handle curved elements

  → you may have a look in `projects/small_fem` from the `Gmsh`'s repository for a `FEM` code with high-order capabilities

- Further optimisation steps can also be applied:
  - call `Gmsh` with `-optimize` and/or `-optimize_ho`
  - or add `Mesh.Optimize = 1;` amd/or `Mesh.HighOrderOptimize = 1;`

`Compound; Homology; Plugin; Background Mesh; CreateTopology; AnswerToLiveUniverseAndEverything;`

TECHNISCHE UNIVERSITÄT DARMSTADT

- That's all I wanted to show!
- But there is more, just look at this slide title...
- You may have a look at the tutorials in the `Gmsh` documentation

`Compound; Homology; Plugin; Background Mesh; CreateTopology; AnswerToLiveUniverseAndEverything;`

TECHNISCHE UNIVERSITÄT DARMSTADT

- That's all I wanted to show!
- But there is more, just look at this slide title...
- You may have a look at the tutorials in the `Gmsh` documentation

Thank you for your attention!

`return 0;`