fc oogmsh package, User's Guide*

François Cuvelier†

October 21, 2018

Abstract

This Python package make it possible to generate mesh files from *.geo* file by using `gmsh`. It's also possible with the ooGmsh class to read the mesh file and to store its contains in more user-friendly form. This package must be regarded as a very simple interface between gmsh files and Python. So you are free to create any objects you want from an ooGmsh object.

0 Contents

*Compiled with Python 3.7.0

†Université Paris 13, Sorbonne Paris Cité, LAGA, CNRS UMR 7539, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, cuvelier@math.univ-paris13.fr.

This work was partially supported by ANR Dedales.

1 Introduction

The oogmsh Python package is closely related to `gmsh`, see [2] or [1], which is a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. `gmsh` can also build two-dimensional meshes and three-dimensional surface meshes. This package was initialy created to make it possible from Python to rapidly

- generate mesh file from .geo file by using `gmsh`
- efficiently read this mesh file and store its contents in ooGmsh Python object easy to manipulate.

The ooGmsh Python object can be used to create, from a .msh file, any objects needed by your project. For example, the fc-simesh Python package uses this package to create the siMesh object containing all the simplices elements of the mesh.

This toolbox was tested under

OS	**Python**	**gmsh**
Ubuntu 18.04 LTS	python.org,[3] 2.7.15, 3.5.6, 3.6.6, 3.7.0	from 3.0.0 to 4.0.1
MacOS High Sierra	miniconda, [4]: 2.7.15, 3.7.0 python.org, [3]: 2.7.15, 3.5.4, 3.6.6, 3.7.0	from 3.0.0 to 4.0.1
Windows 10 (1803)	python.org, [3]: 2.7.15, 3.5.4, 3.6.6, 3.7.0	from 3.0.0 to 4.0.1

In a first step we quickly present the installation and the configuration of the oogmsh package for using the `gmsh` application. Thereafter, we describe the oogmsh 's functions which use `gmsh` to create mesh files.

2 Installation and configuration

2.1 Installation

- For an installation which isolated to the current user, one can do:

```
$ pip install --user fc_oogmsh
```

- For an installation for all users, one can do:

```
$ sudo pip install fc_oogmsh
```

For Python 3 installation, `pip3` must be used instead of `pip`.

2.2 Configuration

One have to configure the package for using with `gmsh`. For the default configuration we run under Python:

```
>>> import fc_oogmsh
>>> fc_oogmsh.configure()
```

By default, the gmsh binary is supposed to be located in

- `<USERDIR>/bin/gmsh` under linux,
- `<USERDIR>/GMSH/Gmsh.app/Contents/MacOS/gmsh` under Mac OS X,
- `<USERDIR>/Softwares/GMSH/gmsh.exe` under Windows

The `fc_oogmsh.configure()` try to guess where is the `gmsh` binary. If this command failed or if we want to specify the `gmsh` binary location, one can use the `gmsh` option to specify the `gmsh` binary file with full path.

- For example, under Linux:

  ```
  >>> fc_oogmsh.configure(gmsh='/usr/local/GMSH/gmsh-3.0.4-Linux/bin/gmsh')
  ```

- For example, under Windows:

  ```
  >>> fc_oogmsh.configure(gmsh=r'C:\Users\toto\GMSH\gmsh-3.0.4-Windows\gmsh.exe')
  ```

  Note that the `gmsh` string is given as *raw litteral string* (`r'...'`) to avoid using of escape sequences as `'\n'`, `'\t'`, ...

- For example, under MacOs:

  ```
  >>> fc_oogmsh.configure(gmsh='/Users/toto/GMSH/3.0.4/Gmsh.app/Contents/MacOS/gmsh')
  ```

Now, it's possible to run one of the demo functions

Python code with output

```
fc_oogmsh.demo02();
```

```
**********************
Running demo02 function
**********************
*** Build mesh file
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** Read mesh file
*** Print oGh ->
ooGmsh object
    dim : 2
  types : [1 2]
 orders : [1]
     nq : 3479
      q : ndarray object[float64], size (3479, 2)
toGlobal: ndarray object[int64], size (3479,)
  sElts : list of 2 elements
*** Print oGh.sElts[0] ->
Elt object
      d : 1, type : 1, order : 1
    geo : line
    nme : 360
     me : ndarray object[int64], size (2, 360)
phys_lab: ndarray object[int64], size (360,)
geo_lab : ndarray object[int64], size (360,)
part_lab: list of 360 elements
nb_parts: ndarray object[int64], size (360,)
  nTags : list of 0 elements
```

3 gmsh interface functions

All the functions provided in this section use **gmsh** to create a mesh file from a **gmsh** geometry script file (extension *.geo*).

3.1 function fc_oogmsh.buildmesh2d

This function uses **gmsh** and a *.geo* file (describing a 2D-geometry) to generate a 2D-mesh.

Syntaxe

```
meshfile=fc_oogmsh.buildmesh2d(geofile,N)

meshfile=fc_oogmsh.buildmesh2d(geofile,N,Key=Value)
```

Description

meshfile=fc_oogmsh.buildmesh2d(geofile,N) create a 2D-mesh using **gmsh** and the *geo* file geofile (without path). The integer N has two functions : numbering the name of the generated mesh as <geofile without extension and path> + <-N.msh> and passing this number to **gmsh** via the option "-setnumber N <N>". Usually we used this parameter in **gmsh** to set the prescribed mesh element size at the points. (see given *geo* files)
As output return a file name (with full path) corresponding to the mesh generated by **gmsh**.

meshfile=fc_oogmsh.buildmesh2d(geofile,N,Key=Value, ...) specifies function options using one or more Key,Value pair arguments. The Key options can be

- `meshdir` : to specify the directory where the mesh file will be written, (default: output of the function fc_oogmsh.Sys.getDefaultMeshDir())
- `meshfile` : to specify the name of the mesh file. Without directory path, the mesh file is saved in directory given by `meshdir` option. (default: `<meshdir>/<meshfile>-<N>.msh`)
- `force` : to force meshing even if the mesh file already exists if Value is true (default : false)
- `verbose` : to specify the degree of verbosity (0, silence; 1, default; 2, command and **gmsh** output)
- `options` : string which contains command-line options used with gmsh (see **gmsh** documentation)

Usage of the `N` parameter To illustrate the usage of the `N` parameter, the `square.geo` file, used to mesh the square $[0, L] \times [0, L]$, is given in Listing 1. In this file, the `N` parameter is used to set the **gmsh** parameter `Mesh.CharacteristicLengthFactor`. As we can see in the output of the Python command, the number of vertices in the mesh increase with `N` value.

```
DefineConstant[
  N = {10, Name "Input/1Refine param."},
  L = {1., Name "Input/2Side length"}
];
Mesh.CharacteristicLengthFactor=5*L/N;
Point (1) = {0, 0, 0};
Point (2) = {L, 0, 0};
Point (3) = {L, L, 0};
Point (4) = {0, L, 0};
Line (3) = {1, 2};
Line (2) = {2, 3};
Line (4) = {3, 4};
Line (1) = {4, 1};
Line Loop (100) = { 1, 2,  3,  4};
Plane Surface (1) = {100};
Physical Line(1) = {1};
Physical Line(2) = {2};
Physical Line(3) = {3};
Physical Line(4) = {4};
Physical Surface(1) = {1};
Physical Point(101) = {1};
Physical Point(102) = {2};
Physical Point(103) = {3};
Physical Point(104) = {4};
```

Listing 1: square.geo file

Python code with output

```
meshfile=fc_oogmsh.buildmesh2d('geofiles/square.geo',20,
    force=True)
oGh1=fc_oogmsh.ooGmsh(meshfile)
print('Generated mesh file:\n  -> %s'%meshfile)
print('    -> number of vertices: %d'%oGh1.nq)
meshfile=fc_oogmsh.buildmesh2d('geofiles/square.geo',200,
    force=True,verbose=2)
oGh2=fc_oogmsh.ooGmsh(meshfile)
print('Generated mesh file:\n  -> %s'%meshfile)
print('    -> number of vertices: %d'%oGh2.nq)
```

```
[fc_oogmsh] Using input file: geofiles/square.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square-20.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square-20.msh
    -> number of vertices: 1155
[fc_oogmsh] Using input file: geofiles/square.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square-200.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
[fc_oogmsh] Command line:
  /fcopt/GMSH/64bit/4.0.1/bin/gmsh -2 -setnumber N 200 -string " Mesh.MshFileVersion=2;" "geofiles/square.geo" -o
        "/home/cuvelier/.local/share/fc_oogmsh/meshes/square-200.msh"
[fc_oogmsh] Running command. Be patient...
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square-200.msh
    -> number of vertices: 106569
```

An another way is given in Listing 2.

```
DefineConstant[
  N = {10, Name "Input/1Refine param."},
  L = {1., Name "Input/2Side length"}
];
h=L/N;
Point (1) = {0, 0, 0, h};
Point (2) = {L, 0, 0, h};
Point (3) = {L, L, 0, h};
Point (4) = {0, L, 0, h};
Line (3) = {1, 2};
Line (2) = {2, 3};
Line (4) = {3, 4};
Line (1) = {4, 1};
Line Loop (100) = { 1, 2,  3,  4};
Plane Surface (1) = {100};
Physical Line(1) = {1};
Physical Line(2) = {2};
Physical Line(3) = {3};
Physical Line(4) = {4};
Physical Surface(1) = {1};
Physical Point(101) = {1};
Physical Point(102) = {2};
Physical Point(103) = {3};
Physical Point(104) = {4};
```

Listing 2: square2.geo file

Python code with output

```
meshfile=fc_oogmsh.buildmesh2d('geofiles/square2.geo',20,
    force=True)
oGh1=fc_oogmsh.ooGmsh(meshfile)
print('Generated mesh file:\n  -> %s'%meshfile)
print('    -> number of vertices: %d'%oGh1.nq)
meshfile=fc_oogmsh.buildmesh2d('geofiles/square2.geo',200,
    force=True,verbose=2)
oGh2=fc_oogmsh.ooGmsh(meshfile)
print('Generated mesh file:\n  -> %s'%meshfile)
print('    -> number of vertices: %d'%oGh2.nq)
```

```
[fc_oogmsh] Using input file: geofiles/square2.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-20.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-20.msh
    -> number of vertices: 567
[fc_oogmsh] Using input file: geofiles/square2.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-200.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
[fc_oogmsh] Command line:
  /fcopt/GMSH/64bit/4.0.1/bin/gmsh -2 -setnumber N 200 -string " Mesh.MshFileVersion=2;" "geofiles/square2.geo" -o
        "/home/cuvelier/.local/share/fc_oogmsh/meshes/square2-200.msh"
[fc_oogmsh] Running command. Be patient...
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-200.msh
    -> number of vertices: 53343
```

Examples All the following examples use the *.geo* file `condenser11.geo` which is in the directory `geodir` of the toolbox.

Python code with output

```
print('*** fc_oogmsh.buildmesh2d : 1st call')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25,force=True)
print('*** fc_oogmsh.buildmesh2d : 2nd call')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25)
```

```
*** fc_oogmsh.buildmesh2d : 1st call
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** fc_oogmsh.buildmesh2d : 2nd call
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh already exist.
  -> Use "force" flag to rebuild if needed.
```

Python code with output

```
meshfile=fc_oogmsh.buildmesh2d('condenser11',25,force=True,
    verbose=2, options='-string "Mesh.Algorithm=1;" -string "Mesh.ScalingFactor=2;"')
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
[fc_oogmsh] Command line:
  /fcopt/GMSH/64bit/4.0.1/bin/gmsh -2 -setnumber N 25 -string "Mesh.Algorithm=1;" -string "Mesh.ScalingFactor=2;"
        -string " Mesh.MshFileVersion=2;"
        "/fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser11.geo" -o
        "/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh"
[fc_oogmsh] Running command. Be patient...
```

3.2 function `fc_oogmsh.buildmesh3d`

This function uses `gmsh` and a *.geo* file (describing a 3D-geometry) to generate a 3D-mesh. See function `fc_oogmsh.buildmesh2d` for usage and options.

3.3 function `fc_oogmsh.buildmesh3ds`

This function uses `gmsh` and a *.geo* file (describing a 3D surface geometry or a 3D-geometry) to generate a 3D surface mesh. See function `fc_oogmsh.buildmesh2d` for usage and options.

3.4 function `fc_oogmsh.buildpartmesh2d`

This function uses `gmsh` and a *.msh* file (containing of a 2D-mesh) to generate and save a 2D partioned mesh. Returns the filename of the partitioned mesh.

Syntaxe

```
pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np)

pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np,Key=Value)
```

Description

pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np) create a 2D partitioned mesh using `gmsh` and the *msh* file meshfile (with path). The integer np is the number of partitions.
As output return a file name (with full path) corresponding to the partitioned mesh generated by `gmsh`. The output file name is construct as following : <meshfile without extension>-part<np>.msh

pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np,Key=Value, ...) specifies function options using one or more Key,Value pair arguments. The Name options can be

- `'force'` : to force meshing even if the mesh file already exists if Value is true (default : false)
- `'verbose'` : to specify the degree of verbosity (0, silence; 2, default; ...)
- `options` : string which contains command-line options used with gmsh (see `gmsh` documentation)

Examples All the following examples use the `meshfile` as output of the command :
meshfile=fc_oogmsh.buildmesh2d('condenser11',25);

Python code with output

```
print('*** Building the mesh')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25);
print('*** Partitioning the mesh')
pmfile=fc_oogmsh.buildpartmesh(meshfile,5,force=True);
```

```
*** Building the mesh
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh already exist.
  -> Use "force" flag to rebuild if needed.
*** Partitioning the mesh
[fc_oogmsh] Using input file: /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
```

Python code with output

```
print('*** Building the mesh')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25, ↪
    verbose=True);
print('*** Partitioning the mesh')
pmfile=fc_oogmsh.buildpartmesh(meshfile,5,force=True, ↪
    verbose=True, options='-string ↪
    "Mesh.MetisAlgorithm=3;"')
```

```
*** Building the mesh
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh already exist.
  -> Use "force" flag to rebuild if needed.
*** Partitioning the mesh
[fc_oogmsh] Using input file: /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
```

3.5 function `fc_oogmsh.buildpartmesh3d`

This function uses `gmsh` and a *.msh* file (containing of a 3D-mesh) to generate a 3D partioned mesh.

3.6 function `fc_oogmsh.buildpartmesh3ds`

This function uses `gmsh` and a *.msh* file (containing of a 3D surface mesh) to generate a 3D partioned surface mesh.

3.7 function `fc_oogmsh.buildPartRectangle`

This function uses `gmsh` and the *geodir/rectanglepart.geo* file to generate a 2D regular partioned mesh of the rectangle $[0, Lx] \times [0, Ly]$ with $Nx \times Ny$ partitions.

Syntaxe

```
meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N)

meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N,
    Key=Value)
```

Description

`meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N)` create a 2D regular partitioned mesh using `gmsh` of the rectangle $[0, Lx] \times [0, Ly]$ with $Nx \times Ny$ partitions. The N parameter is passed to `gmsh` to set the prescribed mesh element size at the points
As output return a file name (with full path) corresponding to the partitioned mesh generated by `gmsh`. The default output file name is construct as following : rectanglepart-Lx%.3f-Ly%.3f-Nx%d-Ny%d-N%d.msh

`meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N,Key=Value, ...)` specifies function options using one or more Key,Value pair arguments. The Key options can be

- `'force'` : to force meshing even if the mesh file already exists if Value is true (default : false)
- `'verbose'` : to specify the degree of verbosity (0, silence; 2, default; ...)
- `options` : string which contains command-line options used with gmsh (see `gmsh` documentation)

Examples All the following examples ...

Python code with output

```
pmfile=fc_oogmsh.buildpartrectangle(1,1,3,2,100,force=True,
    verbose=True)
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/rectanglepart.geo
[fc_oogmsh] Use option verbose=3 to see gmsh output
```

Python code with output

```
pmfile=fc_oogmsh.buildpartrectangle(1,1,3,2,100,verbose=True,
    force=True,meshfile='./toto.msh')
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/rectanglepart.geo
[fc_oogmsh] Use option verbose=3 to see gmsh output
```

4 Usefull functions

4.1 function fc_oogmsh.configure

This function configure the fc_oogmsh Python package to be used with `gmsh` by generating a configuration file. The name of this file is returned by the fc_oogmsh.Sys.getLocalConfFile() function.

Syntaxe

```
fc_oogmsh.configure()

fc_oogmsh.configure(Key=Value)
```

Description

fc_oogmsh.configure() uses default values to set:

- the location of the `gmsh` binary application,
- the directory of the *.geo* files used by `gmsh`,
- the directory where to save the *.msh* generated by `gmsh`.

fc_oogmsh.configure(Key=Value) specifies function options using one or more Key,Value pair arguments. The Key options can be

- `gmsh` : to specify the location of the `gmsh` binary application,
- `geodir` : to specify the directory of the *.geo* files used by `gmsh`,
- `meshdir` : to specify the directory where to save the *.msh* generated by `gmsh`,
- `reset` : if True configure with default values (default : False)

5 ooGmsh class

The ooGmsh class can be used to read `gmsh` mesh files with the MSH ASCII file format described for example in [1], section 9.1.

In a .msh file the kind of mesh elements are identified by their *elm-type* integer values :

elm-type	**description**
1	2-node line
2	3-node triangle
3	4-node quadrangle
4	4-node tetrahedron
5	8-node hexahedron
6	6-node prism
7	5-node pyramid
8	3-node second order line (2 nodes associated with the vertices and 1 with the edge)
9	6-node second order triangle (3 nodes associated with the vertices and 3 with the edges)
10	9-node second order quadrangle (4 nodes associated with the vertices, 4 with the edges and 1 with the face)
11	10-node second order tetrahedron (4 nodes associated with the vertices and 6 with the edges)
12	27-node second order hexahedron (8 nodes associated with the vertices, 12 with the edges, 6 with the faces and 1 with the volume)
13	18-node second order prism (6 nodes associated with the vertices, 9 with the edges and 3 with the quadrangular faces)
14	14-node second order pyramid (5 nodes associated with the vertices, 8 with the edges and 1 with the quadrangular face)
15	1-node point
16	8-node second order quadrangle (4 nodes associated with the vertices and 4 with the edges)
17	20-node second order hexahedron (8 nodes associated with the vertices and 12 with the edges)
18	15-node second order prism (6 nodes associated with the vertices and 9 with the edges)
19	13-node second order pyramid (5 nodes associated with the vertices and 8 with the edges)
20	9-node third order incomplete triangle (3 nodes associated with the vertices, 6 with the edges)
21	10-node third order triangle (3 nodes associated with the vertices, 6 with the edges, 1 with the face)
22	12-node fourth order incomplete triangle (3 nodes associated with the vertices, 9 with the edges)
23	15-node fourth order triangle (3 nodes associated with the vertices, 9 with the edges, 3 with the face)

24	15-node fifth order incomplete triangle (3 nodes associated with the vertices, 12 with the edges)
25	21-node fifth order complete triangle (3 nodes associated with the vertices, 12 with the edges, 6 with the face)
26	4-node third order edge (2 nodes associated with the vertices, 2 internal to the edge)
27	5-node fourth order edge (2 nodes associated with the vertices, 3 internal to the edge)
28	6-node fifth order edge (2 nodes associated with the vertices, 4 internal to the edge)
29	20-node third order tetrahedron (4 nodes associated with the vertices, 12 with the edges, 4 with the faces)
30	35-node fourth order tetrahedron (4 nodes associated with the vertices, 18 with the edges, 12 with the faces, 1 in the volume)
31	56-node fifth order tetrahedron (4 nodes associated with the vertices, 24 with the edges, 24 with the faces, 4 in the volume)
92	64-node third order hexahedron (8 nodes associated with the vertices, 24 with the edges, 24 with the faces, 8 in the volume)
93	125-node fourth order hexahedron (8 nodes associated with the vertices, 36 with the edges, 54 with the faces, 27 in the volume)

When reading a .msh file generated by `gmsh`, we split the mesh elements by *elm-type* and generate an array of Elmt object. The dimension of this array is the number of differents *elm-type* founds on the .msh file. The properties of the `Elmt` object are:

Properties of `Elmt` object

type	:	integer refers to the type of the element : 1 for 2-node line, 2 for 3-node triangle, ... See the *elm-type* description of [1], section 9.1.
geo	:	string contains the kind of geometry: 'line', 'triangle', 'tetrahedron', ...
d	:	integer space dimension or d-simplex.
order	:	integer order of the element
n_{me}	:	integer number of mesh elements
me	:	array of $d+1$-by-n_{me} integers connectivity array
phys_lab	:	array of n_{me}-by-... integers physical labels of the elements
geo_lab	:	array of n_{me}-by-... integers geometrical labels of the elements
nb_parts	:	array of n_{me}-by-1 integers number of mesh partitions to which the element belongs
part_lab	:	array of n_{me}-by-max(nb_parts) integers part_lab$(i, 1 : \text{nb_parts}(i))$ contains all the partitions index to which the i-th element belongs.

The ooGmsh class was created to store a maximum of(all the) information(s) contained in the .msh file. The properties of this class are:

Properties of ooGmsh class

dim	:	integer space dimension
n_q	:	integer number of vertices/nodes
q	:	dim-by-n_q array of reals array of vertex coordinates
types	:	array of integers List of the element types found in the mesh file.
orders	:	array of integers List of the orders of the element types found in the mesh file.
sElts	:	array of `Elmt` objets One `Elmt` objet by element type, such that sElts(i) contains all the elements of type types(i) and order orders(i).

The ooGmsh class have only one constructor :

```
Gh=ooGmsh(meshfile)
```

where meshfile is the name of ... a mesh file

5.1 Sample 1

The 2d .geo file *condenser.geo* is used to create a .msh file : `condenser-25.msh`. This `.msh` file contains only 1 (2-node line) and 2 (3-node triangle) *elm-type*.

Python code with output

```
meshfile=fc_oogmsh.buildmesh2d('condenser',25);
Gh = fc_oogmsh.ooGmsh(meshfile)
print('*** Gh:')
print(Gh)
print('*** Gh.sElts[0]:')
print(Gh.sElts[0])
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/2d/condenser.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser-25.msh already exist.
  -> Use "force" flag to rebuild if needed.
*** Gh:
ooGmsh object
    dim : 2
  types : [ 1 2 15]
 orders : [1]
     nq : 55651
      q : ndarray object[float64], size (55651, 2)
toGlobal: ndarray object[int64], size (55651,)
  sElts : list of 3 elements
*** Gh.sElts[0]:
Elt object
      d : 1, type : 1, order : 1
    geo : line
    nme : 1486
     me : ndarray object[int64], size (2, 1486)
phys_lab: ndarray object[int64], size (1486,)
geo_lab : ndarray object[int64], size (1486,)
part_lab: list of 1486 elements
nb_parts: ndarray object[int64], size (1486,)
  nTags : list of 0 elements
```

5.2 Sample 2

The 3d .geo file *cylinderkey.geo* is used to create a .msh file : `cylinderkey-10.msh`. This `.msh` file contains 1 (2-node line), 2 (3-node triangle) and 4 (4-node tetrahedron) *elm-type*.

Python code with output

```
meshfile=gmsh.buildmesh3d('cylinderkey',10, force=True)
Gh = gmsh.ooGmsh(meshfile)
print('*** Gh:')
print(Gh)
print('*** Gh.sElts[1]:')
print(Gh.sElts[1])
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/3d/cylinderkey.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/cylinderkey-10.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** Gh:
ooGmsh object
   dim : 3
 types : [1 2 4]
orders : [1]
    nq : 5132
     q : ndarray object[float64], size (5132, 3)
toGlobal: ndarray object[int64], size (5132,)
 sElts : list of 3 elements
*** Gh.sElts[1]:
Elt object
     d : 2, type : 2, order : 1
   geo : triangle
   nme : 6576
    me : ndarray object[int64], size (3, 6576)
phys_lab: ndarray object[int64], size (6576,)
geo_lab : ndarray object[int64], size (6576,)
part_lab: list of 6576 elements
nb_parts: ndarray object[int64], size (6576,)
 nTags : list of 0 elements
```

5.3 Sample 3

The 3d .geo file *sphere8surf.geo* is used to create a 3d surface .msh file : `sphere8surf-40.msh`. This `.msh` file contains 1 (2-node line), 2 (3-node triangle) and 15 (1-node point) *elm-type*.

Python code with output

```
meshfile=gmsh.buildmesh3ds('sphere8surf',40,force=True)
Gh = gmsh.ooGmsh(meshfile)
print('*** Gh:')
print(Gh)
print('*** Gh.sElts[0]:')
print(Gh.sElts[0])
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.7.0/lib/python3.7/site-packages/fc_oogmsh/geodir/3ds/sphere8surf.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/sphere8surf-40.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** Gh:
ooGmsh object
   dim : 3
 types : [ 1 2 15]
orders : [1]
    nq : 23476
     q : ndarray object[float64], size (23476, 3)
toGlobal: ndarray object[int64], size (23476,)
 sElts : list of 3 elements
*** Gh.sElts[0]:
Elt object
     d : 1, type : 1, order : 1
   geo : line
   nme : 756
    me : ndarray object[int64], size (2, 756)
phys_lab: ndarray object[int64], size (756,)
geo_lab : ndarray object[int64], size (756,)
part_lab: list of 756 elements
nb_parts: ndarray object[int64], size (756,)
 nTags : list of 0 elements
```

5 References

[1] Gmsh 2.15.0. `http://gmsh.info`, 2016.

[2] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[3] Python.org. Python. `http://www.python.org/`, 2013.

[4] Conda team. Miniconda, minimal anaconda distribution. `https://conda.io/miniconda.html`, 2017.