Advanced Scientific Computing. First Practical Session

Mourad Ismail and Christophe Prud'homme
Université Joseph Fourier Grenoble 1

November 4, 2015

The aim of this practical session is to learn to use GMSH[1] for creating meshes. We describe briefly how to create a simple mesh, import and handle it with FEEL++[2]. Recall that you can find more details about these softwares by consulting the corresponding documentation.

1 Meshing using Gmsh

1.1 Geometry of the domain

The geometry can be described using an intuitive language. For example, if our domain is the square $[-1,1]^2$, we write a script `square.geo` as follows :

```
 1  lc = 1;

 3  Point(1) = {-1, -1, 0, lc};
    Point(2) = { 1, -1, 0, lc} ;
 5  Point(3) = { 1,  1, 0, lc} ;
    Point(4) = {-1,  1, 0, lc} ;
 7
    Line(1) = {1,2} ;
 9  Line(2) = {2,3} ;
    Line(3) = {3,4} ;
11  Line(4) = {4,1} ;

13
    Line Loop(5) = {1,2,3,4} ;
15  Plane Surface(5) = {5} ;

17
    Physical Line("Gamma") = {1,2,3,4} ;
19  Physical Surface("Omega") = {5} ;
```

In the first line `lc` denotes the characteristic length (defining the mesh step h). The 4 vertices of our square, numbered 1 to 4, are introduced in lines 3 to 6 using four numbers. The first three ones designate the coordinates of the point and the last one is dedicated to the characteristic length. Note that all points must have three components although we work in two dimensions (in this case the third component is zero).

The four edges of the square, numbered 1 to 4, are introduced by lines $8-11$ of the script. Each edge is a segment defined by both ends. For example, `Line(2)={2,3}` means that the second edge of our square is defined by the points numbers 2 and 3.

The four segments defined above are combined with the keyword `Loop` to form a loop defining our domain. It is used in the line 15 of the script to define the domain as a plane surface. The last two lines

[1] `http://www.geuz.org/gmsh/`
[2] `http://www.feelpp.org/`

indicate specific regions of the domain. For example in our example, `Omega` designates the interior of the domain and `Gamma` its boundary.

1.2 ".msh" ASCII File Format

Thanks to the script `square.geo` described in previous section we can obtain a simple mesh given by the Figure 1. Note that in practice we use meshes much finer. For simplicity reasons, we considered a mesh with very few elements to better understand the use of GMSH.

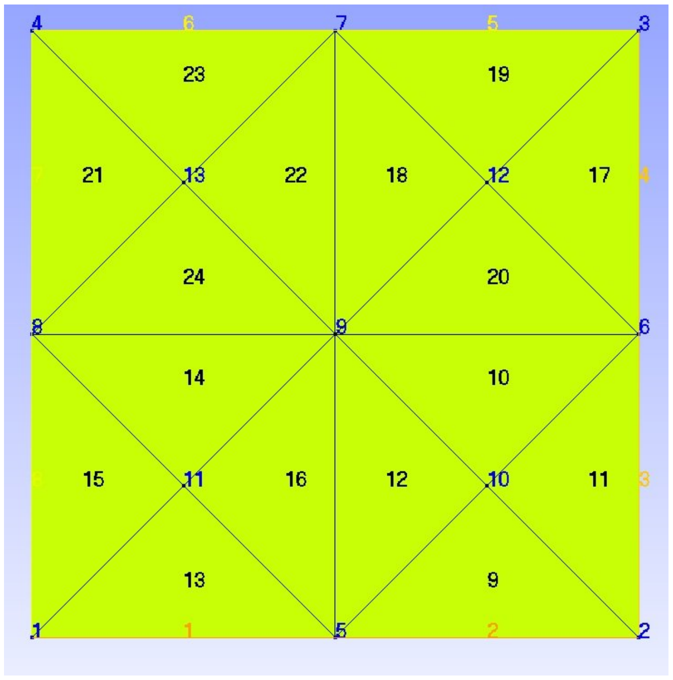

Figure 1: Example of a Coarse Mesh

The so created mesh is saved in the file `square.msh`. Here are its contents:

```
$MeshFormat
2 0 8
$EndMeshFormat
$PhysicalNames
2
1 1 "Gamma"
2 2 "Omega"
$EndPhysicalNames
$Nodes
13
1   -1   -1    0
2    1   -1    0
3    1    1    0
4   -1    1    0
5    0   -1    0
6    1    0    0
7    0    1    0
8   -1    0    0
9    0    0    0
10   0.5 -0.5  0
11  -0.5 -0.5  0
12   0.5  0.5  0
```

```
23  13  -0.5  0.5  0
    $EndNodes
25  $Elements
    24
27  1   1  3  1  1  0  1  5
    2   1  3  1  1  0  5  2
29  3   1  3  1  2  0  2  6
    4   1  3  1  2  0  6  3
31  5   1  3  1  3  0  3  7
    6   1  3  1  3  0  7  4
33  7   1  3  1  4  0  4  8
    8   1  3  1  4  0  8  1
35  9   2  3  2  5  0  5  2  10
    10  2  3  2  5  0  6  9  10
37  11  2  3  2  5  0 10  2   6
    12  2  3  2  5  0 10  9   5
39  13  2  3  2  5  0 11  1   5
    14  2  3  2  5  0 11  9   8
41  15  2  3  2  5  0  8  1  11
    16  2  3  2  5  0  5  9  11
43  17  2  3  2  5  0  6  3  12
    18  2  3  2  5  0  7  9  12
45  19  2  3  2  5  0 12  3   7
    20  2  3  2  5  0 12  9   6
47  21  2  3  2  5  0 13  4   8
    22  2  3  2  5  0 13  9   7
49  23  2  3  2  5  0  7  4  13
    24  2  3  2  5  0  8  9  13
51  $EndElements
```

The first three lines in the file `square.msh` describe the mesh format. The first number indicates the version number (2 in our case). The second one concerns the mesh file type (0 for ASCII and 1 for binary). The third index indicates how data are stored (using 8 bits in this case. *i.e.* *sizeof(double)=8*).

Between keywords `$PhysicalNames` and `$EndPhysicalNames` we have dimension, number and names of physical parts of the mesh. We find also the numbers which they are assigned automatically. In our case we have considered two “Physical Names” : `Omega` for the domain (two dimensions) and `Gamma` (one dimension) for its boundary. The mesh generator has attributed the 1 index for `Gamma` and 2 for `Omega`.

From the 9th line begins the mesh nodes description : their number first, then, in each line we have the number of the corresponding node followed by its coordinates.

The description of various elements of the mesh begins after the keyword `$Elements`. First, we have the number of elements (26th line of the file `square.msh`. Then, there is a detailed description of all elements of the mesh. In our case we have two types of elements: segments belonging to the boundary `Gamma` and triangles of the domain `Omega`. Each element is described by several indices. We detail those which correspond to the element of line 31 (an edge of the boundary `Gamma` and those corresponding to the element of line 47 (a triangle). These two elements are also presented in figure 1.

	elt-nbr	*elt-type*	*tags-nbr*	*elt-ref*	*region-ref*	*part-ref*	*node1*	*node2*	*node3*
line 31	5	1	3	1	3	0	3	7	-
line 47	21	2	3	2	5	0	13	4	8

where :

- *elt-nbr* : element number (5 for element described at the 31st line and 21 for element at 47th line)
- *elt-type* : element type : 1 for segments, 2 for triangles with three degrees of freedom, 3 for quadrangles with four degrees of freedom, etc ...
- *tags-nbr* : number of tags describing the element. In our case this index is 3 referring to the three indices of columns 4, 5 et 6 (*elt-ref*, *region-ref* and *part-ref*).

- *elt-ref* : element reference. In our example, 1 for elements belonging to the boundary `Gamma` and 2 for those belonging to `Omega` (see lines 6 and 7 of file `square.msh`)
- *region-ref* : region reference to which the element belongs. In our example 3 and 5 (cf. lines 10 and 15 of the script `square.geo`)
- *part-ref* : partition reference. In our case this index is zero since we have considered only one partition.
- *node1, node2, node3* : these indices refer to nodes that constitute our element. The segment of line 31st is formed by the two nodes number 3 and 7. The triangle of line 47 is formed by vertices number 13, 4 and 8. (See figure 1).

2 Application. Meshing the domain of the project

The aim of this section is to apply the previous method to mesh the domain of our interest in the project. Note that this domain has the geometry given by figure 2

- Recall that we have to create a mesh with different physical parts
- Each sub-domain Ω_1, Ω_2, Ω_3 and Ω_4 must have a different physical name
- The external boundaries Γ_1, Γ_2, Γ_3 and Γ_4 and each part of the internal boundaries $\partial\Omega_1$ and $\partial\Omega_2$ have to be considered as different physical parts too

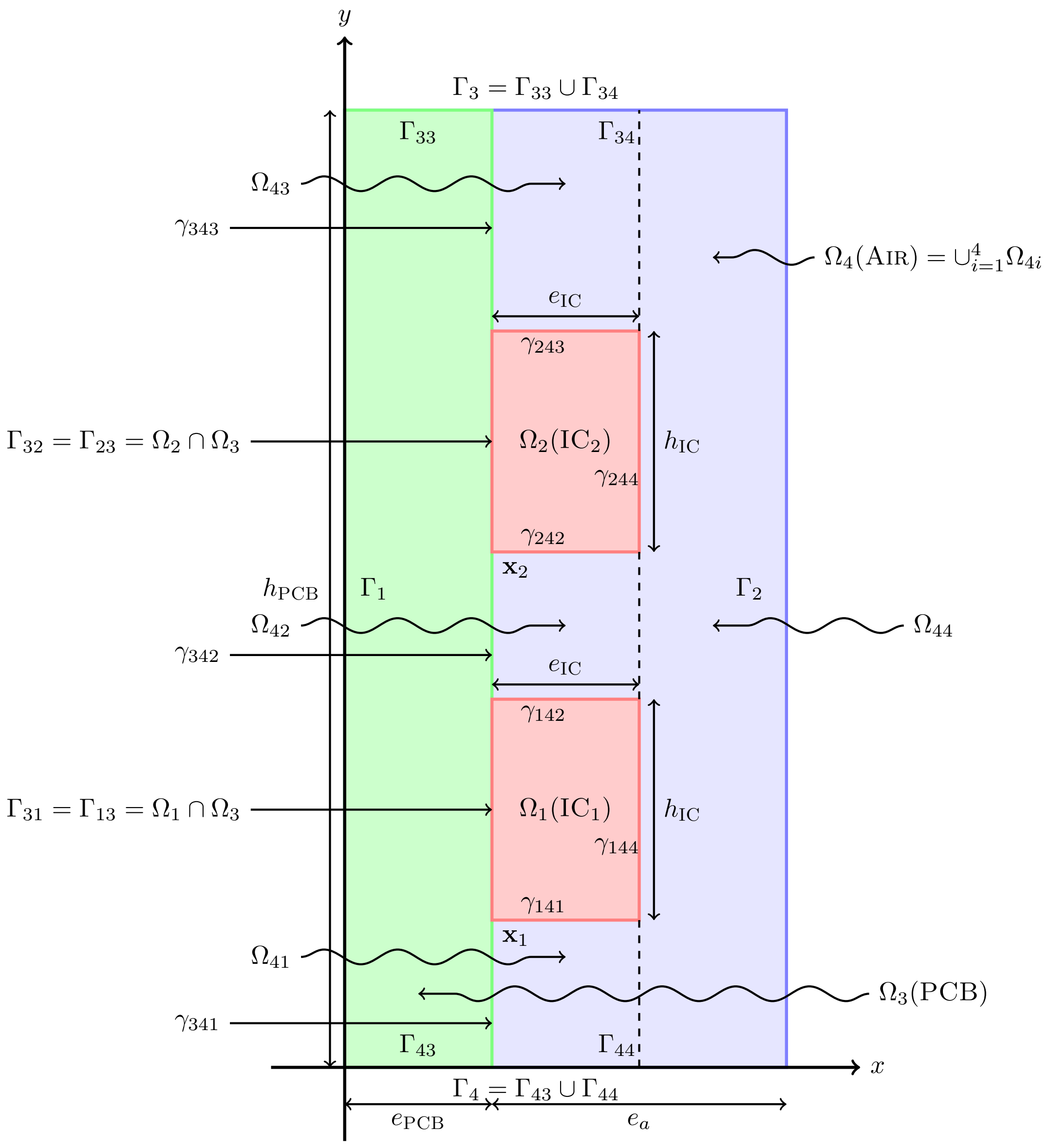

Figure 2: Geometry of $\Omega = \cup_{i=1}^4 \Omega_i$ with $\partial\Omega = \cup_{i=1}^4 \Gamma_i$

3 Handle your mesh with a simple C++ code

Write a simple C++ code that reads your `.geo` file and calls automatically Gmsh to generate the corresponding `.msh` file. You can skip this section and go to the next one if you want to use the FEEL++ library[3]. To do this, you will have to install this library under Linux (Debian or Ubuntu distribution) or Macos (unsing brew[4]). Before doing this, be sure that your computer has at least 8Go of memory.

4 Handle your mesh with a Feel++ application (optional)

The example files `myapp.cpp` and `mymesh.cpp` could help you to create your first FEEL++ application and handle interfacing with GMSH. These files are located at `/usr/local/share/doc/feel/examples/`.

More precisely you can use the function `createGMSHMesh`[5] with at least two arguments as follow :

- assume that your project class has a member `projectMesh` of type `mesh_ptrtype` :

```
    typedef Feel::Simplex<Dim, 1,Dim> entity_type;
    typedef Feel::Mesh<entity_type> mesh_type;
    typedef boost::shared_ptr<mesh_type> mesh_ptrtype;
```

- This member can be initialized by one of your project constructors using the FEEL++ BOOST function `createGMSHMesh` [6]. For example :

```
    projectMesh =  createGMSHMesh( Feel::_mesh=new mesh_type,
                                   Feel::_desc=myCreateGeoFunction(<list
                                   of arguments>) );
```

  Where `myCreateGeoFunction` is a function that you created to generate your GMSH `.geo` file. An example of this function to create the `square.geo` file of section 1.1 is given by :

```
Feel::gmsh_ptrtype
myCreateGeoFunction( double h,
                     double xmin, double xmax,
                     double ymin, double ymax )
{
  std::ostringstream ostr;
  std::ostringstream nameStr;
  ostr << "h=" << h << ";\n"
       << "xmin=" << xmin << ";\n"
       << "xmax=" << xmax << ";\n"
       << "ymin=" << ymin << ";\n"
       << "ymax=" << ymax << ";\n"
       << "\n"
       << "Point(1) = {xmin, ymin, 0, h};\n"
       << "Point(2) = {xmax, ymin, 0, h};\n"
       << "Point(3) = {xmax, ymax, 0, h};\n"
       << "Point(4) = {xmin, ymax, 0, h};\n"
       << "\n"
       << "Line(1) = {1,2};\n"
       << "Line(2) = {2,3};\n"
       << "Line(3) = {3,4};\n"
       << "Line(4) = {4,1};\n"
       << "\n"
       << "Line Loop(5) = {1,2,3,4};\n"
       << "Plane Surface(5) = {5};\n"
       << "\n"
```

[3] `http://www.feelpp.org/`
[4] `http://brew.sh`
[5] don't forget to include the header files `feel/feelfilters/gmsh.hpp` and `feel/feelfilters/exporter.hpp`
[6] see `/usr/local/include/feel/feelfilters/gmsh.hpp`

```
        << "Physical Line(\"Gamma\") = {1,2,3,4};\n"
        << "Physical Surface(\"Omega\") = {5};\n";
  nameStr << "square";
  return Feel::gmsh_ptrtype(new Feel::gmsh_type);
}
```

This function has to be adapted to the geometry of your project domain. Its arguments could be : the mesh size parameters, e_{PCB}, e_a, h_{PCB}, h_{IC}, e_{IC}, h_1 and h_2. Recall that h_1 and h_2 are the second coordinates of points $\mathbf{x}_1 = (e_{\mathrm{PCB}}, h_1)$ and $\mathbf{x}_2 = (e_{\mathrm{PCB}}, h_2)$.