Appendix C

FreeCAD; Gmsh; pyGIMLi workflow

This is a short electrical resistivity tomography (ERT) modeling and inversion example using a FreeCAD → Gmsh → pyGIMLi workflow. Gmsh provides a build-in CAD engine, but defining a geometry in a parametric CAD program such as FreeCAD is much more intuitive and flexible. After defining the geometry in FreeCAD, a mesh is created with Gmsh. In Gmsh the mesh is locally refined in the inversion region and around the electrodes and region, boundary and electrode markers are assigned. The mesh is then loaded into pyGIMLi and subsequently ERT modeling and inversion are done in BERT. In the 1.1 release of pyGIMLi, BERT will be integrated into pyGIMLi and this example will be included on pygimli.org, but for now ERT modeling and inversion are still done in pyBERT. Where pyBERT is the python implementation of BERT. Note that this is an ERT modeling and inversion example, but that this workflow can easily be translated to other geophysical methods as well.

This is an ERT modeling and inversion example on a small dike. The geometry and acquisition design come from the IDEA League master thesis of Joost Gevaert. The target in this example is to find the geometry of a sand channel underneath the dike. To keep the example simple, a homogeneous resistivity is assumed except for the sand channel, which has a higher resistivity. One mesh for ERT modeling is created, consisting of 3 regions; the outer region; the inner region (same as inversion region in this example) and the sand channel. A second mesh for ERT inversion, consisting of 2 regions; the outer region and the inversion region. When the same meshes are used for modeling and inversion, the geometry of the sand channel is already included in the structure of the mesh. Therefore the mesh itself would act as prior information to the inversion.

November 28, 2019

The files used for this example are included, so that you can do the example yourself. The following file formats can be found in the attached archive:

Table C-1: FreeCAD; Gmsh; pyGIMLi workflow file types

Extension	File type	Extension	File type
.FCStd	FreeCAD file	.geo	Gmsh script
.brep	Geometry file to be imported in Gmsh	.msh	Gmsh mesh file
.dat	BERT unified data format file	.bms	BERT mesh file
.vtk	Visualization Toolkit file	.py	Python script
.xlsx	Excel with electrode locations and ERT scheme		

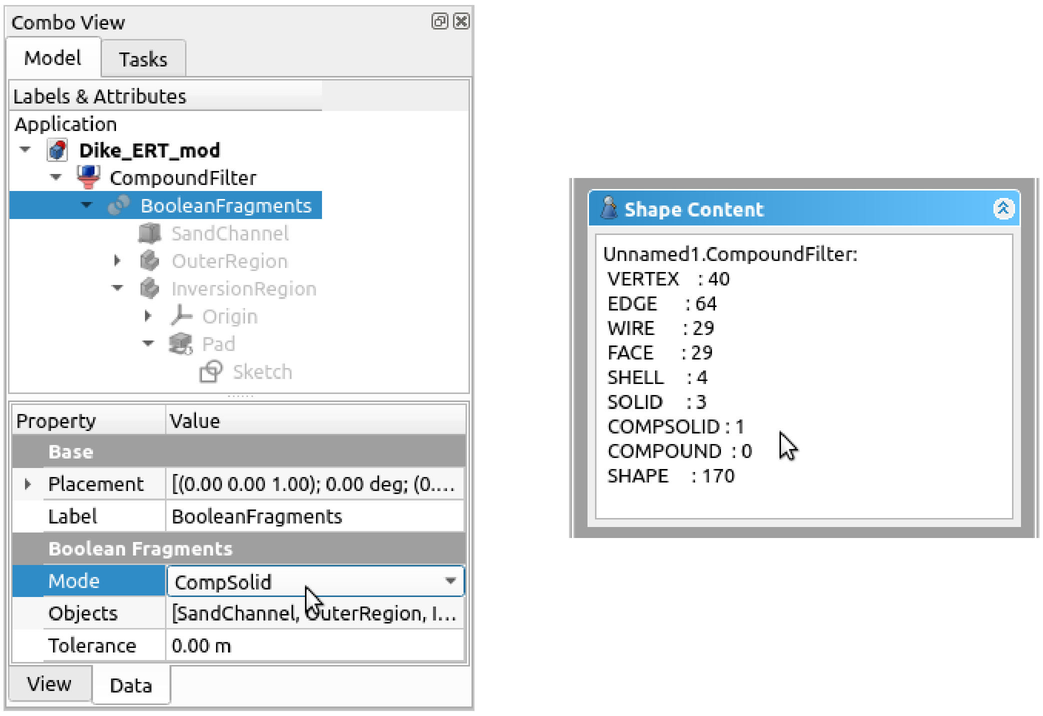
C-1 FreeCAD

The first step is to design each region of the geometry separately in the part workbench, or in the part design workbench for more complicated geometries. To get familiar with the part design workbench, this FreeCAD-tutorial with some videos is great. When all objects, i.e. regions are defined, they have to be merged into one single ”compsolid”, i.e.composite solid. Meaning one object that consists of multiple solids that share the interfaces between the solids. Once this is done, the geometry can be exported. The geometry should be exported in .brep or .step format. .brep is preferred, as that is the native format of the OpenCascade CAD engine on which both FreeCAD and Gmsh run. .step is the standardized CAD exchange format and also works well. Gmsh can also read .iges files, but this format should be avoided. Development for the .iges format stopped after 1996 and geometries are not always imported correctly.

The outer region and inversion region of this dike example were created in the Part Design workbench, by making a sketch and then extruding it with the Pad option, see the Inversion-Region in the object tree in Figure C-1. The sand channel is a simple cube, created in the Part workbench. The trick then lies in merging these shapes into a single compsolid. This is done in the following steps:

1. Open a new project and merge all objects, i.e. regions (File → Merge project...) into this project
2. In the Part workbench, select all objects and create Boolean Fragments (Part → Split → Boolean Fragments)
3. Select the newly created BooleanFragments in the object tree and change its Mode property to CompSolid, see Figure C-1.
4. Keep BooleanFragments selected and then apply a Compound Filter to it (Part → Compound → Compound Filter)
5. Quality check the obtained geometry. Select the newly created CompoundFilter from the object tree and click Check Geometry (Part → Check Geometry). The SOLID: number, in the Shape Content, should match the number of objects merged when creating the Boolean Fragments, 3 in this example. COMPSOLID: should be 1 and the COMPOUND: should be 0, also for other geometries. COMPSOLID: 1 and COMPOUND: 0 indicates that the objects were indeed merged correctly to one single compsolid, see Figure C-1.

6. Select the CompounSolid from the object tree and export (File → Export...) as .brep.



Combo View

Model | Tasks

Labels & Attributes

```
Application
▾ Dike_ERT_mod
  ▾ CompoundFilter
    ▾ BooleanFragments
        SandChannel
      ▸ OuterRegion
      ▾ InversionRegion
        ▸ Origin
        ▾ Pad
            Sketch
```

Property	Value
Base	
Placement	[(0.00 0.00 1.00); 0.00 deg; (0....
Label	BooleanFragments
Boolean Fragments	
Mode	CompSolid
Objects	[SandChannel, OuterRegion, I...
Tolerance	0.00 m

View | Data

Shape Content

```
Unnamed1.CompoundFilter:
VERTEX    : 40
EDGE      : 64
WIRE      : 29
FACE      : 29
SHELL     : 4
SOLID     : 3
COMPSOLID : 1
COMPOUND  : 0
SHAPE     : 170
```

Figure C-1: Left: object tree of compsolid ERT modeling geometry. The compsolid was created by creating Boolean Fragments, in CompSolid Mode, from the SandChannel, OuterRegion and InversionRegion and then applying a Compound Filter. Right: Check Geometry of the compsolid. Important that the following match: SOLID: number of objects merged; COMPSOLID: 1; COMPOUND: 0.

C-2 Gmsh

Gmsh is incredibly versatile, but also has a very steep learning curve. Gmsh has several tutorials and demos, but lacks a clear overview on its website. This website does give a nice Gmsh tutorial overview, with pictures of the results and some additional examples.

It is easiest to work in Gmsh with both the Gmsh script and graphical user interface (GUI) open side by side. After making changes to the script, update the geometry in the GUI by pressing the Reload script button in the Geometry module. After changing something in the GUI, make sure to update the script.

Before diving into local mesh refinement, putting the electrodes in the mesh and assigning markers, the .brep geometry file should be checked. Import the geometry, change the units to [m], have a look at the geometry in the GUI and then mesh it in the GUI:

```
SetFactory("OpenCASCADE");
Merge "Dike_ERT_mod.brep";        // Import FreeCAD geometry
Geometry.OCCScaling = 0.001;      // Scale geometry from [mm] to [m]
```

Mesh the geometry by subsequently clicking 1D, 2D and 3D in the Mesh module. Check if the geometry was meshed without errors and that the mesh makes sense. Especially check whether the meshes of two adjacent regions share nodes on their interfaces. Tips for viewing the mesh:

1. Double left clicking opens a menu in where you can set geometry and mesh visibility.
2. Tools → Visibility opens a window in which you can select parts of the mesh and geometry. Also handy later to QC whether physical groups were set correctly.
3. Clip the mesh and geometry with Tools → Clipping.
4. The number of elements ect. can be found in the Tools → Statistics window.

If importing and meshing the .brep geometry went correctly, great! Next define the Characteristic Length (CL) for each region and the electrodes:

```
4  // Define characteristic lengths
5  cl_elec = 0.1;    cl_dike = 0.6;    cl_outer = 30;
6  // Volume 1: sand channel, Volume 2: outer region, Volume 3: inner region
7  Characteristic Length { PointsOf{ Volume{2}; } } = cl_outer;
8  Characteristic Length { PointsOf{ Volume{3}; } } = cl_dike;
9  Characteristic Length { PointsOf{ Volume{1}; } } = cl_dike;
```

The CL is defined at each point and dictates the mesh size at that point. When setting the CL, make sure to think about the order (lines 7-9 set CL from big to small), because the CL is overwritten in shared points. Now reload the script, mesh the geometry again and have a look how the mesh changed.

The next step is adding the electrodes to the mesh. The few lines of python code below read the electrode positions from an Excel data sheet and write them into a Gmsh .geo file. The grid on the dike has 152 electrodes. These points are added in Gmsh as points 201-352, to prevent clashing with points already defined in the geometry.

```python
import pandas as pd
filename = 'ERT_pos_and_scheme.xlsx'
elec_depth = 0.02       # elec depth [m]
# Load electrode positions from Excel
pos = pd.read_excel(filename, sheet_name='elec_pos')
pos['z'] = pos['z'] - elec_depth
# Write to Gmsh .geo file
id = int(200)            # Gmsh point ID, high number to prevent clash with other points
gmsh_grid = open('gmsh_elec_pos.geo', 'w')
for i, xyz in pos.iterrows():
    gmsh_grid.write('Point(%d) = {%.3f, %.3f, %.3f, cl_elec};\n'
                    % ((xyz['elec #'] + id), xyz['x'], xyz['y'], xyz['z']))
gmsh_grid.close()
```

Copy and paste the electrode positions from `gmsh_elec_pos.geo` to the Gmsh script with the geometry and include the newly added points in the volume of the dike body:

```
Point(201) = {8.500, 0.600, -0.020, cl_elec};
...
Point(352) = {14.500, 10.954, 0.186, cl_elec};
// Include electrodes in the correct geometric volume
Point{201:352} In Volume{3};
```

Reload the Gmsh script and mesh it again to see the result. Further mesh refinement is then possible with so-called background fields. Taking a quick look at Gmsh tutorial `t10.geo` is highly recommended. It shows a wide range of possible background fields. In this example a Distance field is defined from the electrodes and then a Threshold field is applied as the background field:

```
// Define a background field that further refines mesh size.
// LcMax -                         /------------------
//                               /
//                             /
// LcMin -o----------------/
//        |                |       |
//      Point           DistMin DistMax
Field[1] = Distance;         Field[1].NodesList = {201:352};
Field[2] = Threshold;        Field[2].IField = 1;
Field[2].LcMin = cl_elec;    Field[2].LcMax = cl_dike;
Field[2].DistMin = 0.2;      Field[2].DistMax = 1;
Field[2].Sigmoid = 0;        Field[2].StopAtDistMax = 1;
Background Field = 2;
```

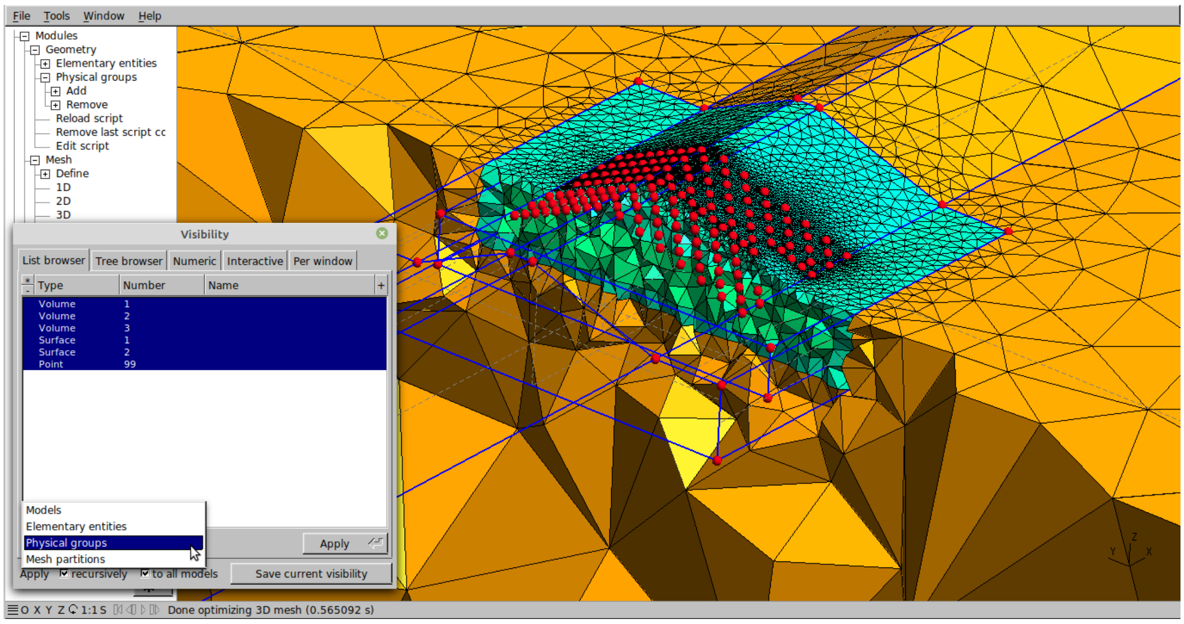
Again reload the Gmsh script and mesh it, to see the result. As the last step in creating the mesh, the physical groups have to be defined, such pyGIMLi and pyBERT recognize regions, boundaries and the electrodes. Defining can be done in the GUI or in the script. In the GUI in menu on the left: Modules → Geometry → Physical groups → Add. In either case it can be handy to use the Tools → Visibility window, to show and hide parts of the geometry. Make sure to follow the same naming conventions for marking the regions, surfaces and points. This to make sure that pyBERT correctly recognizes them. In the script it should look similar to:

```
// Boundaries: (1) Free Surface
Physical Surface(1) = {13, 12, 7, 9, 11, 21, 23, 24, 27, 25, 26, 28, 29};
// (2) Mixed Boundary Conditions: V = 0
Physical Surface(2) = {14, 20, 15, 8, 16};
// Regions
Physical Volume(2) = {1}; // Outer region
Physical Volume(1) = {2}; // Outer region
// Electrodes
Physical Point(99) = {201:352}; // Setting electrode marker
```

The final mesh should look something like Figure C-2. Check whether the Physical groups are defined correctly using the Visibility window as shown in the figure. Then save the mesh in .msh format: File → Save Mesh. Finally make the inversion mesh in the same way as the modeling mesh. The differences being that (a) there should be no sand channel in the geometry of the inversion mesh. Meaning that there are also only 2 volumes, the outer region and the inner region, i.e. inversion region. And (b) that the mesh does not have to be as fine. `cl_elec` = 0.25 and `cl_dike` = 1.2 were used for the inversion mesh in the attached files. Besides changing the mesh size by playing around with the CL, the general mesh size can also be adapted in Gmsh by changing the Global mesh size factor (double left click).

C-3 pyGIMLi and pyBERT

When the mesh has the desired refinement and is saved in .msh format, it is time to set up ERT modeling and inversion. First an ERT file in unified data format, which is the file format which tells BERT the electrode positions and quadruples, i.e. measurement scheme.



Type	Number	Name
Volume	1	
Volume	2	
Volume	3	
Surface	1	
Surface	2	
Point	99	

```
Done optimizing 3D mesh (0.565092 s)
```

Figure C-2: ERT modeling mesh. Red dots are geometry points, as well as the electrode grid in the middle of the dike.

The quadruples used in this tutorial are dipole-dipole measurements in lines across the dike. The BERT file can be created by running:

```python
# Write to BERT/pyGIMLi unified data format .dat file
ne = len(pos)                           # number of electrodes
scheme = pd.read_excel(filename, sheet_name='ERT_scheme')
BERT_dat = open('BERT.dat', 'w')
BERT_dat.write(f'{ne}\n' + '# x y z\n')
for i, xyz in pos.iterrows():
    BERT_dat.write('%.3f %.3f %.3f\n' % (xyz['x'], xyz['y'], xyz['z']))
BERT_dat.write(f'{scheme.shape[0]}\n')
BERT_dat.write('# a b m n\n')
for i, abmn in scheme.iterrows():
    BERT_dat.write('%d %d %d %d\n' % (abmn['A'], abmn['B'], abmn['M'], abmn['N']))
BERT_dat.write('0')
BERT_dat.close()
```

Subsequently running the lines of code below, creates another BERT file in unified data format. This time with apparent resistivities, that resulted from ERT modeling.

```python
import pygimli.meshtools as mt
import pybert as pb
#Read Gmsh .msh model mesh and model ERT
msh_nm = 'Dike_ERT_mod'
mesh_mod = mt.readGmsh(msh_nm + ".msh", verbose=True)
# Read electrode positions and ERT scheme from BERT file
BERT = pb.importData('BERT.dat')
# Initiate pyBERT ERT object
ert = pb.ERTManager()
# Model ERT
rhomap = [[1, 10.], [2, 10.], [3, 300.]]
```

```
38 data_mod = ert.simulate(mesh_mod, res=rhomap, scheme=BERT)
39 data_mod.save('BERT_mod.dat')
```

The modeled data can then be inverted by running:

```
40 #Read Gmsh .msh
41 msh_nm = 'Dike_ERT_inv'
42 mesh_inv = mt.readGmsh(msh_nm + ".msh", verbose=True)
43 # Initiate pyBERT ERT object
44 ert = pb.ERTManager()
45 # Import the modeled data
46 data = pb.importData('BERT_mod.dat')
47 # Invert modeled data,
48 ert.invert(data, mesh=mesh_inv, lam=15)
49 # Write inversion outcome to .vtk to visualize in paraview
50 paradomain = ert.paraDomain
51 paradomain.addData('Resistivity [Ohm.m]', ert.resistivity)
52 paradomain.addData('log10(sensitivity)', ert.coverageDC())
53 paradomain.exportVTK('invResult.vtk')
```

The inversion optimized the resistivity inside the inversion region. In BERT the inversion region is called the parameter domain. Lines 50-53 export the inversion result, so the resistivity model and sensitivity distribution inside the parameter domain, to a .vtk file. A .vtk file is a Visualization Toolkit file, which can be opened ParaView, which is a flexible visualization toolkit. The resulting resistivity model, visualized in ParaView, looks like this:

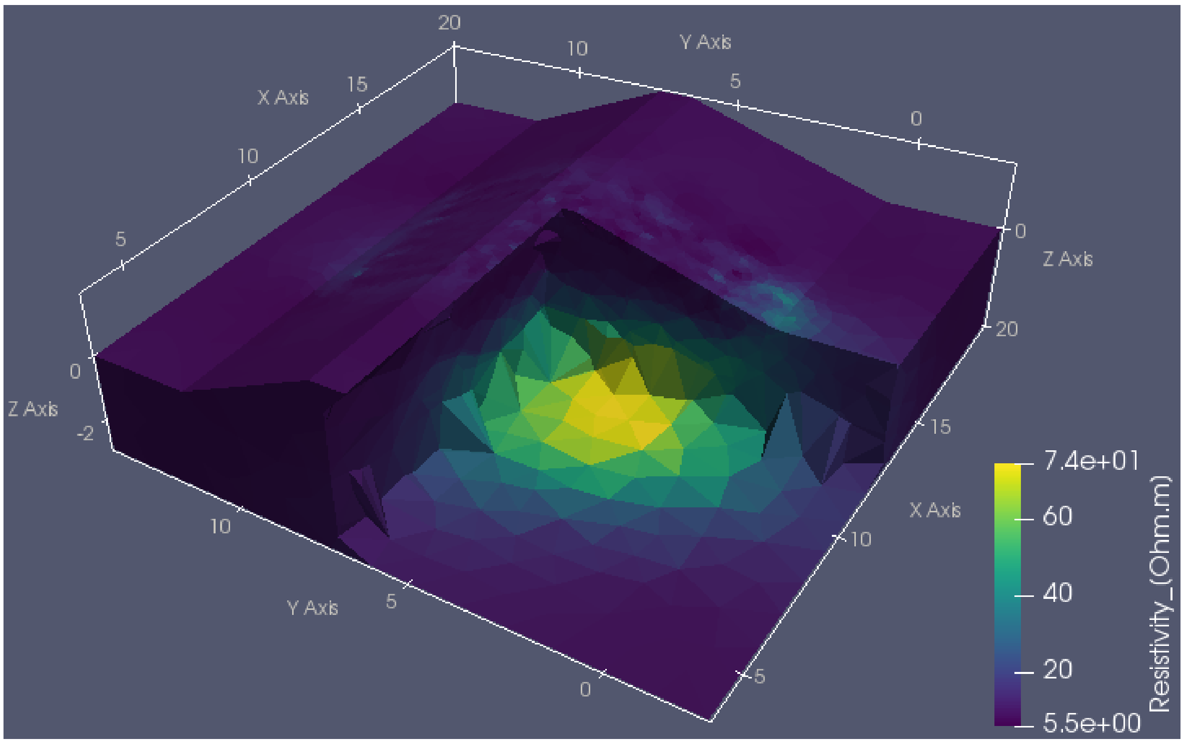


Figure C-3: Paraview visualization of the resistivity model that results from the inversion of the modeled ERT data.