Detector Simulation in Garfield++ with Open-Source Finite Element Electrostatics

Josh Renner (UC Berkeley/LBNL)
In collaboration with Lawrence Livermore National Laboratory

1 Overview and Installation Notes

All of the simulations described in this document can be performed with a Linux machine capable of running the following free software packages:

1. Garfield++ [1]: A program for constructing detailed simulations of multiple aspects of drift chambers with gaseous media.
2. Elmer [2]: A finite element analysis program. This program can be used to calculate the electrostatic potential throughout space in a detector geometry, and the results can be imported into Garfield++ for use in simulations.
3. Gmsh [3]: A mesher, used to define a geometry and split it into many small elements so that the finite element method can be applied in the calculation of electrostatic fields.

In general, one begins by defining a detector geometry and creating a mesh of that geometry in Gmsh. The mesh is then converted to a format readable by Elmer, and the finite element calculation is performed in Elmer, resulting in a value of the electrostatic potential at each finite element vertex (or node). This list along with the Elmer-format mesh can be imported into Garfield++, which calculates the resulting electrostatic fields and provides functionality for simulation of detector physics in these fields.

The following are notes on the installation of Gmsh and Elmer on a Linux machine. As many varieties of Linux machines exist, these notes do not cover details of every issue that could arise.

1.1 Installing Elmer

Instructions: http://www.csc.fi/english/pages/elmer/sources

Notes:
To begin, obtain the Elmer source code. The code can be downloaded from the Elmer project on Sourceforge. Click "browse code" and there should be a "Download GNU tarball" link at the bottom of the list shown. Then compile each library using the `./configure`, `make`, and `make install` commands as described in the installation instructions.

Note that when compiling some of the libraries, an error may occur after running `./configure` stating: "cannot compile a simple Fortran program". In this case, it is likely that the appropriate environment variables have not been set. Enter the following lines in the command prompt and try again (replacing the compiler names below with the ones available on the current system):

```
export CC=gcc
export CXX=g++
export FC=gfortran
export F77=f77
```

There is also an associated user interface called `ElmerGUI`. This tool requires the use of the Qt and optionally several other libraries. None of the examples in this document require this interface.

1.2 Installing Gmsh

To obtain a pre-compiled version of Gmsh, go to the Gmsh website and download the latest version and unzip/untar. The program can be run from the bin directory that was extracted from the .tar. Instructions for installing from source are not included here.

2 Constructing Finite Element Maps

2.1 Finite Element Maps

For the detailed simulation of electromagnetic fields, Garfield can import electromagnetic field maps created by other programs using finite element analysis (see [4] for a short introduction and [5] for a longer treatment).

After specifying a geometry and creating the mesh with Gmsh, the Elmer finite element analysis program can be used to calculate the electrostatic fields using the mesh. During this process one can assign materials to the different parts of the geometry and specify the properties of the materials (such as the dielectric constant) and boundary conditions. Once imported into Garfield this amounts to the creation of the entire detector geometry to be used in the simulation. Here the steps of this process are described in some detail, and section 2.7 provides a full example of the concepts discussed.

2.2 Creating Field Maps

The following guide summarizes the use of Gmsh and Elmer for the purpose of creating electrostatic field maps for use in Garfield. Further details of the use of Gmsh and Elmer can be found in their user manuals [6, 7, 8, 9].

2.2.1 Creating the Geometry

Gmsh supports both GUI-based and text-based entry of geometry information. It may be useful to work with both entry methods to some extent. The text-based entry has the particular advantage of parameterization (see [6] section 4). Note that a mesh could also be produced without the use of Gmsh, and using only Elmer through a text-based entry method, but this method is not described here.

Geometry specified in Gmsh is saved in a file with extension `.geo`. The general procedure for the creation of geometry is as follows:

1. Create points on a 3D coordinate system that can be connected by lines and curves to form the desired geometry.

   Example:

   ```
   Point(1) = {0,0,0}
   ```

   ... creates a point with numerical ID equal to 1 and coordinates (0,0,0).

   All points must have a distinct numerical ID. Parameterizing input becomes useful because these ID numbers can be kept in numerical order if a point is removed or added during editing. The coordinates can also be made relative to those of another point so that they do not have to be changed if the other point is moved.

   Example:

```
x=0; y=0; z=0; d=1;
p1 = newp; Point(p1) = {x,y,z};
p2 = newp; Point(p2) = {x+d,y,z};
```

... creates two new points separated by 1 unit on the x-axis. Note that the `newp` command provides the point ID one greater than the last one used. There are similar commands for other geometric types (see [6] pgs. 25-26) which will be used below. When assigned to a variable, this ID can then be input as the ID of the point.

2. Connect the points by lines and curves.

   Example:

   ```
   l1 = newl; Line(l1) = {p1, p2};
   ```

   ... connects points `p1` and `p2` to form a line.

3. Create surfaces by specifying lines and curves as surface boundaries. This is a two-step process which involves first the creation of a *line loop* from several lines and next the creation of a *plane surface* or, if the surface is not a plane, for example in the case of the sides of a cylindrical shell, a *ruled surface*. See [6] sections 5.1.2-5.1.3 for more information on these elements. In general a line loop is specified by a list of lines such that the points connected on the surface are listed in the order that they would be encountered when traveling around the edges of the surface. For example, if line 1 connects point 4 to 5 (i.e. `Line(1) = {4,5}`), line 2 connects point 5 to 6, and line 3 connects point 4 to 6, then the triangular surface would be specified by a line loop with list `{1, 2, -3}`. Note the negative sign indicating the order in which the points connected by line 3 are read. Reading the point connections, one has 4, 5, 6, 4. If line 3 connected point 6 to 4 rather than 4 to 6, the line loop list would be `{1, 2, 3}`. Note that Gmsh does keep track of the orientation of the line loop (in which direction the lines are traversed), though the author has not observed the significance of the orientation in any of the examples presented in this document.

   Example:

   ```
   l1 = newl; Line(l1) = {p4, p5}
   l2 = newl; Line(l2) = {p5, p6}
   l3 = newl; Line(l3) = {p4, p6}
   lp = newll; Line Loop(lp) = {l1, l2, -l3};
   s = news; Plane Surface(s) = {lp};
   ```

   ... creates three points with lines, forms a line loop $4 \rightarrow 5 \rightarrow 6 \rightarrow 4$, and creates a surface bounded by this line loop.

4. Create volumes by specifying surfaces as boundaries. This is also a two-step process which involves first the creation of a *surface loop* from several surfaces and next the creation of a volume. The process of creating line and surface loops and then assigning surfaces and volumes may be more intuitive if done initially through the Gmsh GUI, as it will create the loop automatically based on the selected lines or surfaces and then create the surface or volume. However, it will save the actual numerical identifiers in the `.geo` file, so if variables are to be used they must then be entered manually using text entry and the `Tools->Visibility` menu option to match variables with ID numbers.

   Example:

```
s_loop = newsl; Surface Loop(s_loop) = {s1, s2, s3, s4, s5, s6};
vol = newv; Volume(vol) = {s_loop};
```

... creates a surface loop from 6 surfaces enclosing a volume and then creates the enclosed volume. Note that the order in which surfaces are listed in a surface loop does not affect the volume defined, but a negative sign before the ID can be used to change the orientation of a surface in the loop from that originally defined in the line loop. It is suggested in the Gmsh manual that the orientation of all surfaces in a surface loop be consistent.

5. Create *physical surfaces* from one or more surfaces and *physical volumes* from one or more volumes. These physical surfaces and volumes will be used later to identify the different parts of the geometry, for example to tell Elmer which surfaces are assigned which boundary conditions and to tell Garfield which part of the geometry is the gaseous drift medium.

   Example:

   ```
   psurf = newreg; Physical Surface(psurf) = {s1, s2};
   ```

   ... creates a physical surface composed of surfaces `s1` and `s2`.

Once the geometry has been specified, the mesh can be created from the command line.

Example:

```
gmsh <geometry_file>.geo -3 -order 2
```

... meshes all elements described in `<geometry_ file>.geo`, creating a mesh with tetrahedral elements suitable for second-order interpolation.

The mesh will be saved with the same name as the geometry file but with a `.msh` extension. Note that adding the `-optimize` flag may help improve the mesh (see [6] pg. 13). If the `-order 2` option is removed, the mesh will be created with first-order tetrahedral elements. Note that only second-order (quadratic) Gmsh/Elmer-based meshes are currently supported in Garfield++. See [8] appendix D for more information on first and second-order tetrahedral elements.

2.2.2 Additional Information on Gmsh

Below are a few miscellaneous Gmsh tips that may help in getting started:

a) Different elements of the geometry can be shown/hidden by clicking `Tools->Visibility` on the main menu. If the elements have been given IDs with a variable name, those names will also be shown.

b) *Characteristic Lengths:* Points can be created with a fourth argument called a characteristic length, which defines the size of the finite element in the mesh near that point (see [6] section 6.3.1). This can be used to make the mesh finer at some points and coarser at other points.

2.3 Calculating the Fields

There are two main components to Elmer that will be used to calculate the electric fields once a mesh has been created:

1. ElmerGrid can be used to convert the mesh output by Gmsh to a form that Elmer can then use to perform calculations.
2. ElmerSolver takes the output of ElmerGrid and performs the finite element calculation to obtain the electric potential and fields.

2.3.1 Converting the Mesh

First ElmerGrid must be run on the mesh produced by Gmsh.

Example:

```
ElmerGrid 14 2 <mesh_file>.msh -autoclean
```

... converts the mesh output by Gmsh and located in the current directory into the mesh format readable by ElmerSolver.

The option 14 indicates that the input file will be a Gmsh .msh, the option 2 indicates that the output will be in the format readable by ElmerSolver, and the -autoclean option indicates that the physical elements defined in the geometry file will have their IDs re-numbered starting at 1 (see [7] section 1.2). This command will work without the -autoclean option, but later assigning boundary conditions and identifying materials may not be possible if this option is not specified. Note that when the re-numbering is performed, the physical surfaces, which will later correspond to boundary conditions, are re-numbered separately from the physical volumes, which will later correspond to "bodies" in ElmerSolver. In each case, the lowest ID will be re-numbered as 1, and the second-lowest as 2, etc. That means that if one uses the newreg keyword to specify the ID numbers, the re-numbered physical elements will be in the same order as they are defined in the .geo file.

ElmerGrid should output 4 files, mesh.boundary, mesh.elements, mesh.header, and mesh.nodes. The contents of these files are described in [7]. These files will be read by ElmerSolver and later by Garfield.

2.3.2 Solving for the Fields

Next, ElmerSolver must be run with an input file describing the boundary conditions of the problem and specifying how Elmer is to solve for the fields. This information is placed in several different blocks in a file with extension .sif. Full details and explanation on creating .sif files can be found in [8], an example is given in section 2.7, and relevant notes are summarized in table 1. See also [9] pgs. 53-56 for more information on the Electrostatics solver in specific.

With the .sif file located in the same directory as the directory created by ElmerGrid containing the converted mesh files, ElmerSolver can be run to perform the calculation.

Example:

```
ElmerSolver <solver_input_file>.sif
```

... performs the calculation described by the given solver input file.

ElmerSolver should output 2 files in the mesh directory created by ElmerGrid: <mesh_name>.result and <mesh_name>.ep. Only the .result file will be relevant for simulations with Garfield, as the .ep file is meant to be read into the visualization program ElmerPost (see [8] Appendix C). The results of the calculation must next be imported into Garfield so that detector simulations can be performed with them.

Block	Notes
`Header`	- Use the name and directory generated by `ElmerGrid`
`Simulation`	- Use `<mesh_name>.result` (same name as given to mesh) as the "output file" for import into Garfield
`Constants`	- Only ϵ_0 is necessary for electrostatics
`Body N`	- $N =$ the number of the physical volume (starting with 1); Be sure to specify both equation and material
`Solver/ Equation`	- Use `Stat Elec Solver` for electrostatics
`Material N`	- N is not related to the physical elements except as assigned in the `Body` statement; The same material can be assigned to more than 1 body; specify dielectric constants here
`Boundary Condition N`	- N is **not** the number of the physical volume; this is set with the `Target Boundary` keyword. Periodic boundary conditions are set using the `Periodic BC` keyword

Table 1: Notes on `ElmerSolver` Input Blocks

2.4 Importing the Field Maps into Garfield++

In Garfield++, the class `ComponentElmer` is used to import an Elmer-based field map. The following items must be present to do so:

1. A separate directory containing the mesh and result files produced by `ElmerGrid` and `ElmerSolver` must be located in the directory containing the executable Garfield++ program that will import the field map.
2. The mesh files must be left with their original names (`mesh.boundary`, `mesh.elements`, `mesh.header`, and `mesh.nodes`), and the `.result` file must have the same name as the directory.
3. A file containing the dielectric constants called `dielectrics.dat` must be in this same directory. This file contains a list of dielectric constants indexed by their ElmerSolver "body" numbers. These numbers are the IDs of the physical volumes created in Gmsh after re-numbering (starting at 1 due to the `-autoclean` flag of `ElmerGrid`). The body number is listed, followed by a single space, followed by the dielectric constant. The first line of this file must contain a single number that specifies the number of dielectrics that will be listed. For example, for a geometry in which we have two volumes, one for the drift medium and one for the dielectric, the dielectrics file will contain three lines:

```
2
1 1
2 <epsilon_for_dielectric>
```

For example, for a field map we will call "sample_map," the directory structure after the field map has been created and the script is ready to be run may be as follows:

```
./
  (a Garfield++ executable)
  sample_map.geo
  sample_map.msh
  sample_map.sif
  /sample_map
```

```
    mesh.boundary
    mesh.header
    mesh.elements
    mesh.nodes
    sample_map.result
    dielectrics.dat
```

In the source file compiled into the executable with the Garfield++ classes, the `ComponentElmer` object can be used as follows.

```
ComponentElmer * elm = new ComponentElmer("sample_map/mesh.header",
  "sample_map/mesh.elements", "sample_map/mesh.nodes","sample_map/dielectrics.dat",
  "sample_map/sample_map.result","cm");
```

The creation of this object imports the field map with the specified files into the Garfield simulation. One can then include periodicity, for example:

```
elm->EnablePeriodicityX();
elm->EnableMirrorPeriodicityY();
```

adds X-periodicity and mirror Y-periodicity, dictating how Garfield repeats the field map in x and y. One can also assign a medium object, such as a gas described by Magboltz [10], to any medium that was assigned during the creation of the Elmer field map.

```
MediumMagboltz * gasArCO2 = new MediumMagboltz();
gasArCO2->SetComposition("ar",70.,"co2",30.);

... // further define properties of medium and construct ComponentElmer * elm

elm->SetMedium(0,gasArCO2);
```

Once set up, a field map can be assigned to a `Sensor` object which is passed to other objects to perform different functions such as particle drift and visualization of the fields.

```
  Sensor* sensor = new Sensor();
  sensor->AddComponent(elm);
```

2.5 Weighting Potentials and Signal Readout

In addition to simulating particle drift, one may be interested in simulating the resulting charge induced on an electrode in a detector geometry. This can also be done using Garfield++ given that it is provided with the proper weighting potential maps. A thorough explanation of how the induced charge on an electrode can be calculated is given in [11] (beginning on pg. 60), which we will summarize here. The current induced at time t on a given electrode j due to a particle of charge q drifting along the path $\vec{x}(t)$ with velocity $\vec{v}(\vec{x}(t))$ can be found as

$$i_j(t) = -q\vec{v}(\vec{x}(t)) \cdot \vec{F}_j(\vec{x}(t)) \tag{1}$$

where $\vec{F}_j$ is the weighting field corresponding to electrode j. For a given electrode, its weighting potential is the electrostatic potential V computed for the entire geometry given the following boundary conditions: the potential on the electrode $V_j = 1$, and the potential on all other conductors i is set to $V_i = 0$. The weighting field is then computed by $\vec{F}_j = -\vec{\nabla}V$. Note that in equation 1, $\vec{F}_j$ is assumed to have dimensions of length^{-1}, as the unit potential has been divided out of both sides of the equation.

From the above discussion, we see that the weighting potential can be calculated using Elmer on the same mesh used to calculate the drift fields by running `ElmerSolver` again, this time setting the boundary conditions in the `.sif` file such that the readout electrode is assigned `Potential = 1` and all other conductors are assigned `Potential = 0`. Note that multiple field maps can be produced in this way, and each can be assigned to its electrode within Garfield++. Assuming a detector geometry called `sample_map` imported into Garfield++ in a `ComponentElmer` object called `elm` (as in section 2.4), and the solution corresponding to a weighting field for an electrode in the geometry output to `sample_map/sample_map_W1.result`, we can import the weighting field with

```
elm->SetWeightingField("sample_map/sample_map_W1.result","wt1");
```

Note that Garfield++ keeps track of weighting potential maps with a label, and we have chosen `wt1` as the label for this map. Once the weighting potential has been imported, it can be assigned to the `Sensor` object to which the primary field map was assigned.

```
Sensor* sensor = new Sensor();
sensor->AddComponent(elm);
sensor->AddElectrode(elm,"wt1");
```

The developed signal is binned according to a specified start time, bin size, and number of bins. For example:

```
sensor->SetTimeWindow(tStart,(tEnd-tStart)/nsBins,nsBins);
```

To develop a signal, the `Sensor` object to which the weighting potential map was assigned can be passed to another object, for example an `AvalancheMicroscopic`, so that it can be updated during particle drift. Note that the calculation of the signal must be enabled for this to occur.

```
 AvalancheMicroscopic* aval = new AvalancheMicroscopic();
 aval->SetSensor(sensor);
 aval->EnableSignalCalculation();
```

A summary of the procedure for producing weighting field maps is as follows:

1. Produce and import the detector field map as discussed in previous sections.
2. Create a new `.sif` file which is an exact copy of the previous one, except:
   a.) In the `Simulation` group, the `Output File` and `Post File` fields are modified with different names so that the original output files will not be overwritten.
   b.) The boundary conditions are modified so that the electrode corresponding to this weighting potential map has `Potential = 1` and all other conductors have `Potential = 0`.
3. Solve using the newly created `.sif` file.
4. Import the weighting potential into Garfield++, add an electrode, and turn on signal generation for the avalanche/drift mechanism used. The developed signal can be accessed once drifting has completed using methods of the `Sensor` class or the `ViewSignal` class.

2.6 Modular Creation of Field Maps

Gmsh allows the user to define functions and loops in creating a geometry. This allows for the creation of field maps via construction from individual objects defined in functions. The following organizational scheme can be used to build a complex geometry out of several smaller components. There is a single main `.geo` file organized as described below, and one or more smaller `.geo` files that describe individual components of the larger geometry. Note that this is only one possibility and will be used later (section 2.7) in a short example, however it may not be useful depending on the geometry being simulated and is not used in the LEM example of section 3. Either way, it highlights some key points of geometry creation in Gmsh. The procedure is as follows (compare to `parallel_plate.geo` in section 2.7.2):

1. Define three variables:

   a. `n_physv`: Keeps a running count of the number of physical volumes in the geometry. This should be incremented every time a new physical volume is defined.

   b. `n_physs`: Keeps a running count of the number of physical surfaces in the geometry, similar to `n_physv`.

   c. `n_bdry`: Incremented each time a new boundary surface loop is created. These are the surface loops that will define the boundaries between the bulk (drift) medium and the other components. Each surface loop on the boundary is added to the array `bounds` so that it can be later used to create the main volume.

2. Include the `.geo` files that will be part of the design.

3. For each component specify the parameters, call the appropriate function, and retrieve/manipulate the return values. The parameters and return values for each component are described in the comments in their `.geo` files.

   a. Assign values to the parameters of the function that will create the component. These parameters are not “passed” to the functions upon calling the function but are placed in variables with the appropriate names before calling the function. They are then used in the function to describe the geometry created.

   b. (Optional) Assign some default value to each of the return values. This step is not necessary but helps in keeping track of what variables will be given values after calling the function.

   c. Call the function defined in the `.geo` file using for example `call gf_object`.

   d. Use the return values, for example, to collect the identifiers for each surface to be later included as a physical surface.

More objects can be constructed if the available objects are not sufficient to specify the detector geometry.

2.6.1 Functions in Gmsh

To create a function in Gmsh, the syntax is

```
Function function_name
 // Gmsh commands here
Return
```

where the word `Function` must be capitalized. The Gmsh commands entered between `Function` and `Return` can be executed at another location in the script by calling

```
Call function_name
```

ensuring the word `Call` is capitalized.

2.6.2 Arrays and Loops in Gmsh

Gmsh arrays do not need to be initialized. Their values can be assigned with a syntax similar to that of C++, for example,

```
arr_name[index] = value;
```

All of the values assigned in an array can be placed in a list using the `[]` operator. For example, a volume can be defined from an array called `bounds` containing IDs of surface loops as

```
vol_bound = newv; Volume(vol_bound) = {bounds[]};
```

The “for” loop in Gmsh has the following syntax

```
For n In {1:3}
  // code to be looped over here
EndFor
```

2.7 An Example: Parallel Plate Capacitor

The following example shows how to create the geometry, mesh, and finite element field map of a parallel plate capacitor using the modular design method described in this section. All code in this example should accompany this document.

2.7.1 3D Rectangle Module

We begin by creating a 3D cube in Gmsh. The following code is placed in `gf_rectangle.geo`.

```
// *********************************************************************
// gf_rectangle.geo
// -----------------
// A rectangle.
//
// ---------------------------------------------------------------------
// Parameters
// ---------------------------------------------------------------------
// x0: x position of center
// y0: y position of center
// z0: z position of center
// lc: characteristic length
// l: length
// w: width
// h: height
//
// ---------------------------------------------------------------------
// Return Values
// ----------------------------------------------------------------------
// vol: the ID of the inner volume
// s1: ID of surface 1: normal -x
// s2: ID of surface 2: normal +y
// s3: ID of surface 3: normal +x
// s4: ID of surface 4: normal -y
// s5: ID of surface 5: normal +z
// s6: ID of surface 6: normal -z
//
// **********************************************************************

// Default parameters
x0 = 0;
y0 = 0;
z0 = 0;
l = 1;
w = 1;
h = 1;
lc = 1;

Function gf_rectangle

  // ********************************
  // Points
  // ********************************

  p1 = newp; Point(p1) = {x0-l/2, y0-w/2, z0-h/2, lc};
  p2 = newp; Point(p2) = {x0-l/2, y0-w/2, z0+h/2, lc};
  p3 = newp; Point(p3) = {x0-l/2, y0+w/2, z0-h/2, lc};
  p4 = newp; Point(p4) = {x0-l/2, y0+w/2, z0+h/2, lc};
  p5 = newp; Point(p5) = {x0+l/2, y0-w/2, z0-h/2, lc};
  p6 = newp; Point(p6) = {x0+l/2, y0-w/2, z0+h/2, lc};
  p7 = newp; Point(p7) = {x0+l/2, y0+w/2, z0-h/2, lc};
  p8 = newp; Point(p8) = {x0+l/2, y0+w/2, z0+h/2, lc};

  // ********************************
  // Lines
  // ********************************

  l1 = newl; Line(l1) = {p1,p2};
  l2 = newl; Line(l2) = {p2,p4};
  l3 = newl; Line(l3) = {p4,p3};
  l4 = newl; Line(l4) = {p3,p1};
  l5 = newl; Line(l5) = {p5,p6};
  l6 = newl; Line(l6) = {p6,p8};
  l7 = newl; Line(l7) = {p8,p7};
  l8 = newl; Line(l8) = {p7,p5};
  l9 = newl; Line(l9) = {p1,p5};
  l10 = newl; Line(l10) = {p2,p6};
  l11 = newl; Line(l11) = {p3,p7};
```

```
69:   l12 = newl; Line(l12) = {p4,p8};
70:
71:   // ******************************
72:   // Surfaces
73:   // ******************************
74:
75:   lp1 = newll; Line Loop(lp1) = {l1, l2, l3, l4};
76:   s1 = news; Plane Surface(s1) = {lp1};
77:   lp2 = newll; Line Loop(lp2) = {l12, l7, -l11, -l3};
78:   s2 = news; Plane Surface(s2) = {lp2};
79:   lp3 = newll; Line Loop(lp3) = {-l5, -l6, -l7, -l8};
80:   s3 = news; Plane Surface(s3) = {lp3};
81:   lp4 = newll; Line Loop(lp4) = {-l10, l5, l9, -l1};
82:   s4 = news; Plane Surface(s4) = {lp4};
83:   lp5 = newll; Line Loop(lp5) = {-l2, -l12, l6, l10};
84:   s5 = news; Plane Surface(s5) = {lp5};
85:   lp6 = newll; Line Loop(lp6) = {-l9, l8, l11, -l4};
86:   s6 = news; Plane Surface(s6) = {lp6};
87:
88:   // ******************************
89:   // Volumes
90:   // ******************************
91:
92:   sl = newsl; Surface Loop(sl) = {s5, s1, s4, s3, s2, s6};
93:   vol = newv; Volume(vol) = {sl};
94:   // ---------------------------------------
95:
96:   // Update the boundaries array.
97:   bounds[n_bdry] = sl; n_bdry += 1;
98:
99: Return
```

Lines 1-28 provide documentation describing the content of the file and the parameters and return values. Note that in this context "parameter" refers to a variable that can be set before the function is called and whose value is used within the function, and "return value" means a variable that is set during the function that will remain set after the function has ended.

Lines 30-37 assign default values to the parameters. These values will be in effect if the user has not assigned values to the parameters before calling the function `gf_rectangle`.

Lines 39-99 describe the geometry of the rectangle according to the parameters. Note at the end of the file that the surface loop containing all 6 surfaces is added to the array of boundaries. It is important to keep the convention of using `n_bdry` as the counter for the `bounds` array, as each component will update the array and increment the counter. In the end the array will contain all of the surfaces that will serve as boundaries between the internal volumes of each component and the bulk drift volume of the detector.

2.7.2 Parallel Plate Geometry

We now describe the code necessary to use this module to create a parallel plate capacitor. To do this we create two cubes, one inside the other, and create physical surfaces from the top and bottom surfaces of the inner cube to which we will later assign potentials.

```
1:   //---------------------------------------------------------------------
2:   // parallel_plate.geo
3:   // A parallel plate capacitor built from two gf_rectangle objects.
4:   // --------------------------------------------------------------------
5:
6:   // Create the counters.
7:   n_physv = 1; // physical volumes
8:   n_physs = 1; // physical surfaces
9:   n_bdry = 0; // number of boundary surfaces
10:
11:  // Include the components.
12:  Include "gf_rectangle.geo";
13:
14:  // Create outer bounding box; first set parameters and then return values.
15:  x0 = 0; y0 = 0; z0 = 0; lc = 1; l = 2; w = 2; h = 2;
16:  vol = -1; s1 = -1; s2 = -1; s3 = -1; s4 = -1; s5 = -1; s6 = -1;
17:  Call gf_rectangle;
18:
19:  // Create the inner box, the top and bottom of which will be assigned voltages.
20:  x0 = 0; y0 = 0; z0 = 0; lc = 1; l = 1; w = 1; h = 1;
21:  vol = -1; s1 = -1; s2 = -1; s3 = -1; s4 = -1; s5 = -1; s6 = -1;
22:  Call gf_rectangle;
23:
24:  // Save the surface IDs for the top and bottom of the inner box and volume.
25:  s_top_plate = s5;
26:  s_bottom_plate = s6;
27:  vol_inner = vol;
28:
29:  // Create physical surfaces for the top and bottom plates.
30:  physs_top_plate = n_physs; Physical Surface(physs_top_plate) = {s_top_plate};
-->    n_physs += 1;
31:  physs_bottom_plate = n_physs; Physical Surface(physs_bottom_plate) =
-->    {s_bottom_plate}; n_physs += 1;
32:
33:  // Create the bounding volume.
34:  vol_bound = newv; Volume(vol_bound) = {bounds[]};
35:
36:  // Create the physical volumes.
37:  physv_gas = n_physv; Physical Volume(physv_gas) =
-->    {vol_inner, vol_bound}; n_physv += 1;
38:  physv_dielectric = n_physv; Physical Volume(physv_dielectric) =
-->    {vol_inner}; n_physv += 1;
```

In lines 7-10 we define the counters necessary to keep track of the boundaries, physical volumes, and physical surfaces. In line 12 we include the rectangle geometry file. In lines 15-17 we create the outer bounding box, a cube with 2 `cm` sides, and in lines 20-22 we create the inner cube with 1 `cm` sides. Lines 25-27 store the upper and lower surface IDs and the volume ID from the inner box. Note this is not necessary in this example, as we could just use `s5`, `s6`, and `vol`, but it makes the physical surface and volume assignments more clear later on, and would be necessary in cases for which the return variables are later re-used such that their values are changed. Lines

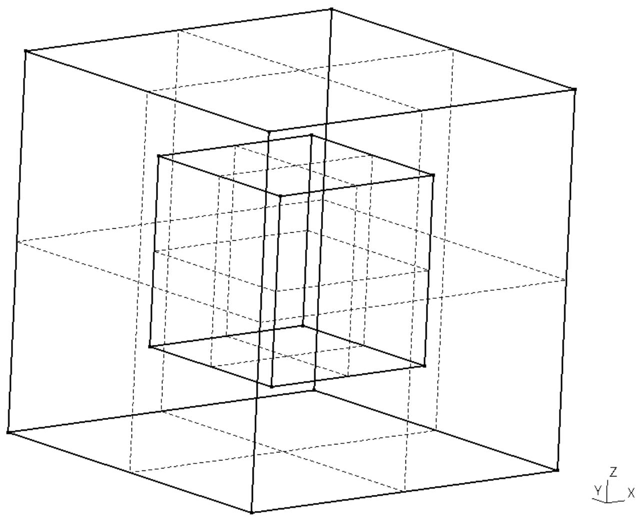

Figure 1: Parallel plate geometry before meshing, from Gmsh. The outer box defines the boundaries of the geometry. The inner box provides the parallel plate geometry, as its top and bottom surfaces will be assigned as separate physical surfaces to which potentials are applied. Note that the remaining surfaces on the inner box will not be assigned as physical surfaces but are present to define an inner volume and assign it a dielectric constant different from that of the rest of the total volume.

30-31 assign physical surface IDs to these surfaces that correspond to the top and bottom parallel plate in the capacitor. Line 34 creates the exterior gas volume using the boundaries defined in the `gf_rectangle` function. Lines 37-38 create the physical volumes for both the gas and the dielectric.

We note the order in which the physical surfaces and volumes are defined and assign dielectric constants and boundary conditions accordingly. Upon running `ElmerGrid` with the `-autoclean` option these will be reordered starting at 1 and progressing in the order in which the physical surfaces and volumes were defined. Note that the list kept for physical surfaces is separate from the list kept for physical volumes. Therefore from the above geometry, boundary condition 1, corresponding to the first defined physical surface, applies to the top plate of the capacitor. Boundary condition 2 then applies to the bottom plate. Likewise material 1 describes the gas and material 2 describes the dielectric.

The parallel plate geometry as seen in Gmsh before meshing is shown in figure 1.

2.7.3 Parallel Plate Electrostatics

We describe the boundary conditions and dielectric constants in the `.sif` file to be read by Elmer to perform the electrostatics calculation.

```
! parallel_plate.sif
!
! ElmerSolver input file for parallel plate geometry.
!
! This is a modified version of "elstatics.sif" from the Elmer tutorials.
! (see ElmerTutorials manual, Tutorial 14)
```

```
!
Check Keywords Warn

! Specify output folder and location.
Header
  Mesh DB "." "parallel_plate"
End

! Details of the calculation and output files.
Simulation
  Coordinate System = Cartesian 3D
  Simulation Type = Steady State
  Steady State Max Iterations = 1
  Output File = "parallel_plate.result"
  Post File = "parallel_plate.ep"
End

! Define constants.
Constants
  Permittivity Of Vacuum = 8.8542e-12
End

! Specify equation and material for gas.
Body 1
  Equation = 1
  Material = 1
End

! Specify equation and material for the dielectric.
Body 2
  Equation = 1
  Material = 2
End

! Define the gas material.
Material 1
  Relative Permittivity = 1
End

! Define the dielectric material.
Material 2
  Relative Permittivity = 2
End

! Upper plate
Boundary Condition 1
  Target Boundaries = 1
  Potential = 0
End

! Lower plate
```

```
58:  Boundary Condition 2
59:    Target Boundaries = 2
60:    Potential = 1000
61:  End
62:
63:  ! Details of the calculation procedure: all that must
64:  !  be calculated is the potential.
65:  Equation 1
66:    Active Solvers(1) = 1
67:    Calculate Electric Energy = True
68:  End
69:  Solver 1
70:    Equation = Stat Elec Solver
71:    Variable = Potential
72:    Variable DOFs = 1
73:    Procedure = "StatElecSolve" "StatElecSolver"
74:    Calculate Electric Field = False
75:    Calculate Electric Flux = False
76:    Linear System Solver = Iterative
77:    Linear System Iterative Method = BiCGStab
78:    Linear System Max Iterations = 1000
79:    Linear System Abort Not Converged = True
80:    Linear System Convergence Tolerance = 1.0e-10
81:    Linear System Preconditioning = ILU1
82:    Steady State Convergence Tolerance = 5.0e-7
83:  End
```

See the ElmerSolver manual [8] for more detailed information about the above commands. We note that in lines 30-39, the "body" numbers correspond to those of the defined physical volumes, and in lines 52-61, the numbers specified as `Target Boundaries` correspond to those of the defined physical surfaces.

The field maps can be imported into Garfield++ using the methods of section 2.4. A contour plot of the potential is shown in figure 2.

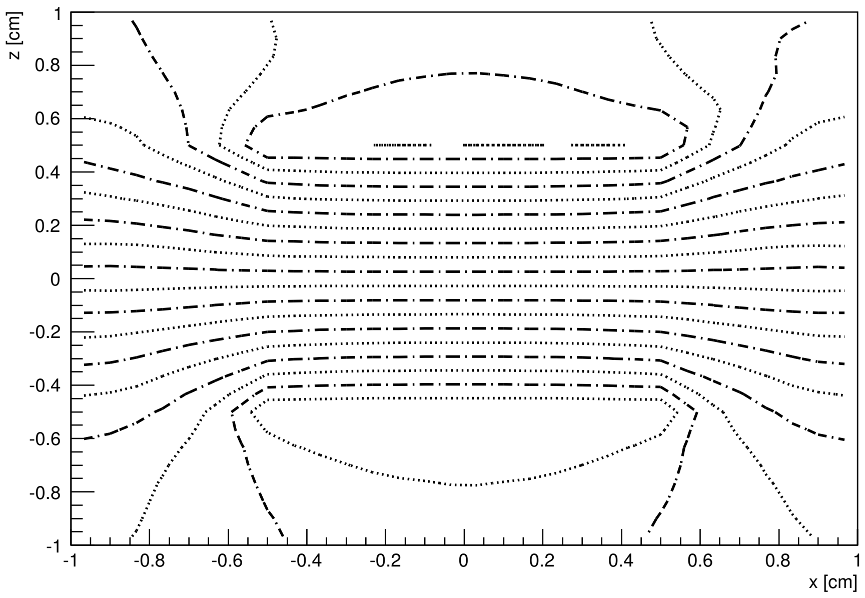

Figure 2: Parallel plate equipotential lines, as plotted in Garfield++.

3 Simulation of LEM/GEM Devices

The gas electron multiplier was introduced in [12] and consists of a three layers of material, two metallic surfaces with dielectric between them, with a pattern of holes drilled through these layers. Since then this concept has been modified to create similar devices such as the thick GEM (THGEM) [13]. Simulations of such devices have been performed in the past, for example in [14, 15, 16], and were consulted in the preparation of this document. Here we present an example of thick GEM (or as we will call, large electron multiplier - LEM) simulation. The code used in constructing the LEM, calculating the fields, importing the fields into Garfield++, and avalanching an electron should accompany this document.

3.1 Constructing a Thick GEM / LEM

The techniques developed in section 2 can be used to create and simulate the LEM. The geometry can be constructed using the periodicity features of Garfield to repeat a single "cell" as done in [14, 15, 17, 18]. Here we use the cell layout from [17].

In this example, many of the geometrical parameters used in the construction of the LEM are similar to those of THGEM #9 in [16]. These parameters are listed in table 2.

Parameter	**Value**	**Description**
$\varepsilon_{\texttt{Kapton}}$	3.23	Dielectric constant of Kapton [19] (60 Hz value, $T = 20^oC$)
$\varepsilon_{\texttt{Ar}}$	1	Dielectric constant of Argon gas (approximate)
r_D	0.015 cm	Radius of hole in dielectric
r_C	0.01 cm	Radius of copper etching (etching extends $r_G \to r_G + r_C$)
t_D	0.04 cm	Thickness of dielectric
t_C	0.0035 cm	Thickness of copper plate
l_E	0.4 cm	Distance between LEM and external electrodes
l_E	0.07 cm	THGEM Pitch
ΔV	1300 V	Voltage across LEM
V_{drift}	100 V/cm	Drift field
V_{trans}	3000 V/cm	Transfer field

Table 2: LEM Parameters

In addition to the LEM itself one must also be concerned with the fields in the regions above and below it, as these are the fields electrons will encounter in drifting towards and away from the LEM. Following the naming conventions of [16], the *drift field*, E_{drift}, is the electric field in region traversed by incoming electrons, and the *transfer field*, E_{transfer}, is the electric field in the region into which electrons are expelled after passing through the LEM. In the geometry, we create these regions by creating a long box encompassing the LEM. The inner volume of this box will be part of the same volume as the center of the LEM hole, and this volume will contain gas. The boundaries of this box will be assigned potentials, meaning they will have to be made physical surfaces. These potentials, assigned relative to the top and bottom plates of the LEM, dictate the drift and transfer fields.

The fields of the LEM can be imported into Garfield as described in section 2. For example, the following lines of code will create a gas medium, an Elmer component for the LEM with a weighting field for the lower electrode, and a sensor containing this component set up to calculate the induced signal on the electrode.

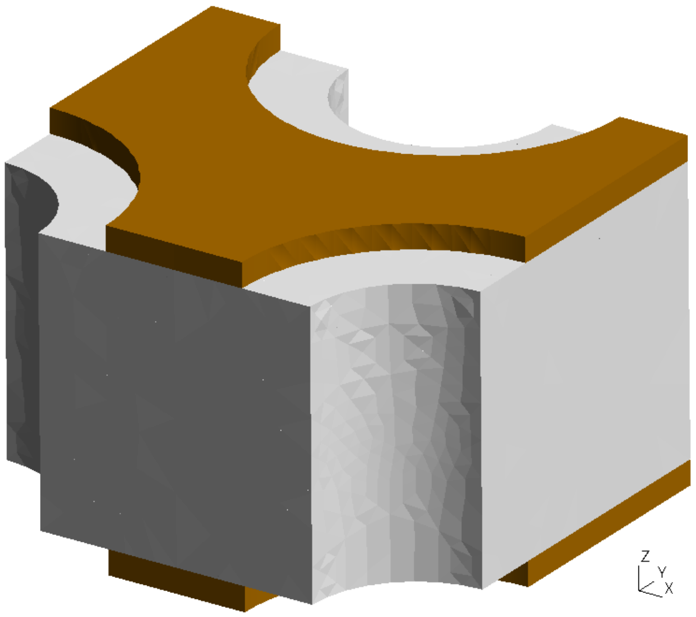

Figure 3: LEM/THGEM cell geometry from Gmsh. The electrodes used to produce the drift and transfer fields are not shown. Note that in the field calculation, the two ends of the cell perpendicular to the x-axis are matched with periodic boundary conditions. The ends perpendicular to the y-axis are divided in half by a plane parallel to the yz plane; the half on the front end located at lesser (greater) x is matched with the half on the back end located at greater (lesser) x.

```
// Define the medium.
MediumMagboltz* gas = new MediumMagboltz();
gas->SetTemperature(293.15);      // Set the temperature [K]
gas->SetPressure(190.);      // Set the pressure [Torr]
gas->EnableDrift();      // Allow for drifting in this medium
gas->SetComposition("ar", 70., "co2", 30.);  // Specify the gas mixture (Ar/CO2 70:30)

// Import an Elmer-created LEM.
ComponentElmer * elm = new ComponentElmer("gemcell/mesh.header",
  "gemcell/mesh.elements", "gemcell/mesh.nodes","gemcell/dielectrics.dat",
  "gemcell/gemcell.result","cm");
elm->EnablePeriodicityX();
elm->EnableMirrorPeriodicityY();
elm->SetMedium(0,gas);
elm->SetWeightingField("gemcell/gemcell_WTlel.result","wtlel");

// Set up a sensor object.
Sensor* sensor = new Sensor();
sensor->AddComponent(elm);
sensor->SetArea(-1*axis_x,-1*axis_y,-1*zi,axis_x,axis_y,zi+0.02);
sensor->AddElectrode(elm,"wtlel");
sensor->SetTimeWindow(0.,tEnd/nsBins,nsBins);
```

Note that to repeat this cell properly in Garfield, one must request x-perodicity and y mirror periodicity as done above. This causes the cell to be tiled in the x-y plane by repeating the entire cell down the x-axis with no rotation or flipping, moving one cell x-width each time. This pattern is repeated every y-length along the y-axis, except the cell is also flipped upside-down once along the y-axis for each increment. Periodic boundary conditions must be assigned so that the fields are solved assuming a complete LEM hole (see figure 3). We have also set the gas medium we have created to be present in all elements with dielectric index 0. Note that once imported into Garfield++, the dielectrics are numbered starting from 0 rather than starting from 1. We can then perform simulations using the imported field map using the sensor to provide information about the fields to other objects. For example, we can visualize the fields and then draw the finite element map on top of the field plot:

```
// Set up the object for field visualization.
ViewField * vf = new ViewField();
vf->SetSensor(sensor);
vf->SetCanvas(c1);
vf->SetArea(-1*axis_x,-1*axis_y,axis_x,axis_y);
vf->SetNumberOfContours(40);
vf->SetNumberOfSamples2d(30,30);
vf->SetPlane(0,-1,0,0,0,0);

// Set up the object for FE mesh visualization.
ViewFEMesh * vFE = new ViewFEMesh();
vFE->SetCanvas(c1);
vFE->SetComponent(elm);
vFE->SetPlane(0,-1,0,0,0,0);
vFE->SetFillMesh(true);
vFE->SetColor(1,kGray);
vFE->SetColor(2,kYellow+3);
vFE->SetColor(3,kYellow+3);
vFE->SetViewDrift(viewDrift);

// Set up the object for signal visualization.
ViewSignal * vSignal = new ViewSignal();
vSignal->SetSensor(sensor);
vSignal->SetCanvas(cSignal);

// Create plots.
vFE->SetArea(-1*axis_x,-1*zi,-1*zi,axis_x,zi+0.02,zi);
vf->PlotContour("v");
vFE->Plot();
vSignal->PlotSignal("wtlel");
```

This produces a plot similar to that shown in figure 4. Note that the format of the plot will depend on the `Draw` options set in the `PlotContour` method of `ViewField.cc`.

3.2 Simulation of an Electron Avalanche

As an example of the kinds of simulations that can be performed upon importing a finite element map, we consider the development of an electron avalanche. Using the LEM described in section 3.1 and the Garfield++ `AvalancheMicroscopic` class, we drift a single electron towards a LEM and track the electron avalanche developed as it passes through a LEM hole.

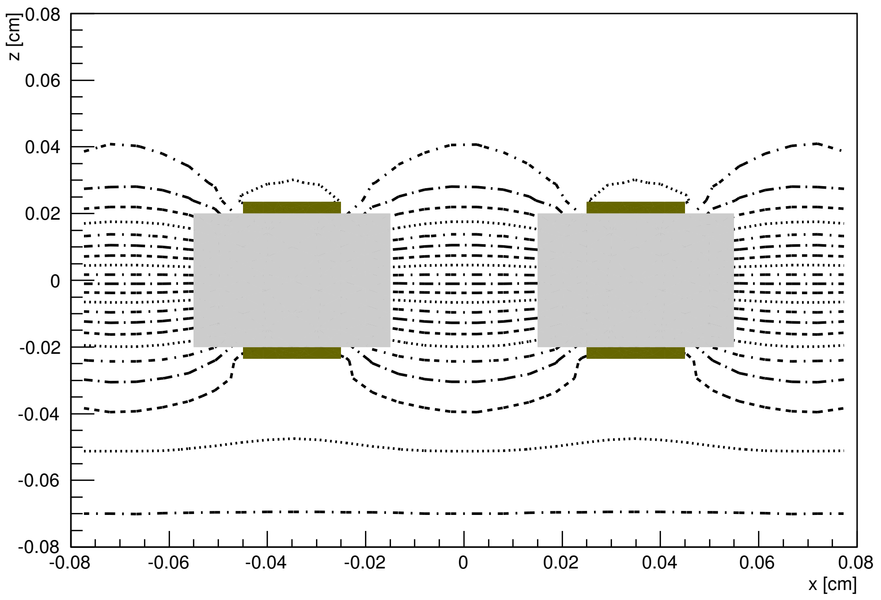

Figure 4: Contours of the potential for a LEM, plotted in Garfield++ using a field map created with Gmsh/Elmer.

First we construct the avalanche:

```
AvalancheMicroscopic* aval = new AvalancheMicroscopic();
aval->SetSensor(sensor);
aval->SetCollisionSteps(100);
aval->EnableSignalCalculation();
```

The first line above creates a new instance of the object that will be used to initiate and track the avalanche. The second line passes the sensor object we created previously to the avalanche object for use in determining the fields at different locations. The third line sets the maximum number of Monte Carlo steps for which a single electron can be tracked before updating the other electrons in the avalanche, and the fourth line enables the calculation of signals on the electrodes that have been registered with `sensor` object induced by the drifting charges in the avalanche. We can use additional Garfield classes to visualize the results of the avalanche by adding an instance of `ViewDrift` to the `ViewFEMesh` object we constructed in section 3.1 and creating a `ViewSignal` class to view the induced signals on an electrode. Calling `EnableAxes` enables a default set of axes to be drawn, as in this case we are not creating a contour plot that would establish a set of axes.

```
// Set up the object for drift line visualization.
ViewDrift* viewDrift = new ViewDrift();
viewDrift->SetArea(-1*axis_x,-1*axis_y,-1*zi,axis_x,axis_y,zi+0.02);
aval->EnablePlotting(viewDrift);

// ... (construct the ViewFEMesh object)

vFE->EnableAxes();
```

```
vFE->SetXaxisTitle("x (cm)");
vFE->SetYaxisTitle("z (cm)");
vFE->SetViewDrift(viewDrift);
```

Finally, we initiate the avalanche with

```
aval->AvalancheElectron(xi, yi, zi, 0., 0., 0., 0., 0.);
```

A plot of the avalanche similar to that of figure 5 can be produced by calling `vFE->Plot();` and a plot of the induced signal similar to figure 6 can be produced with `vSignal->PlotSignal("wtlel");`

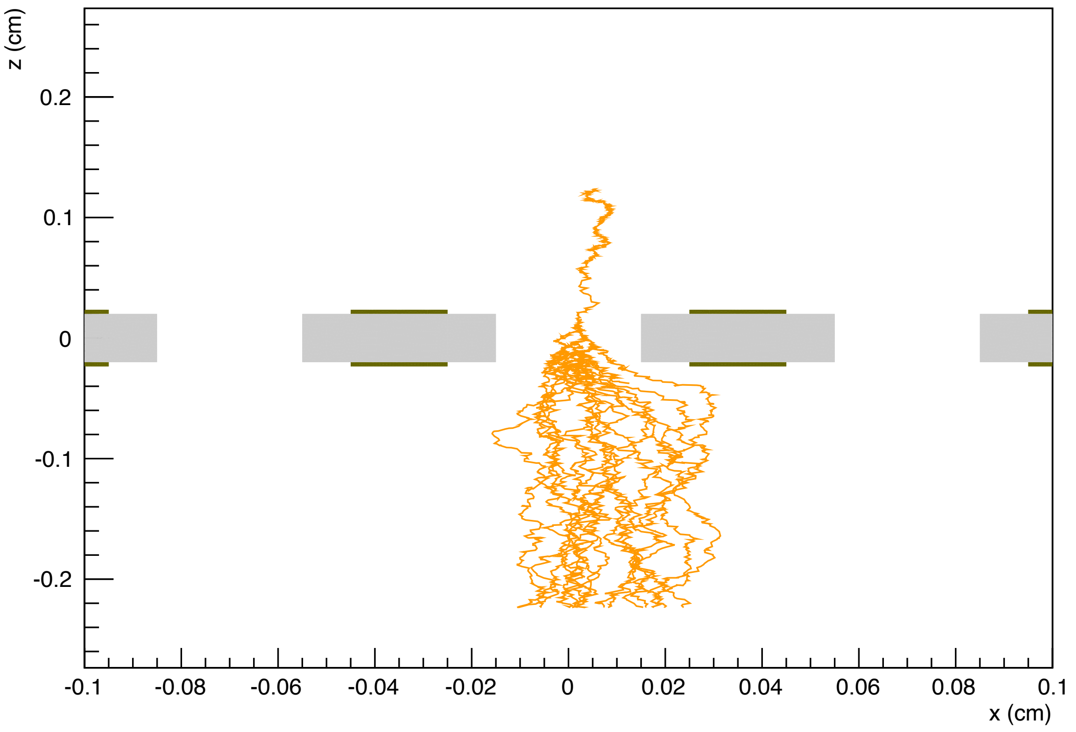

Figure 5: Electron avalanche in a LEM, simulated with Garfield++. The lower readout electrode is invisible in the rendered geometry, but it is located in the horizontal plane at which the drift lines of all of the electrons escaping the LEM terminate.

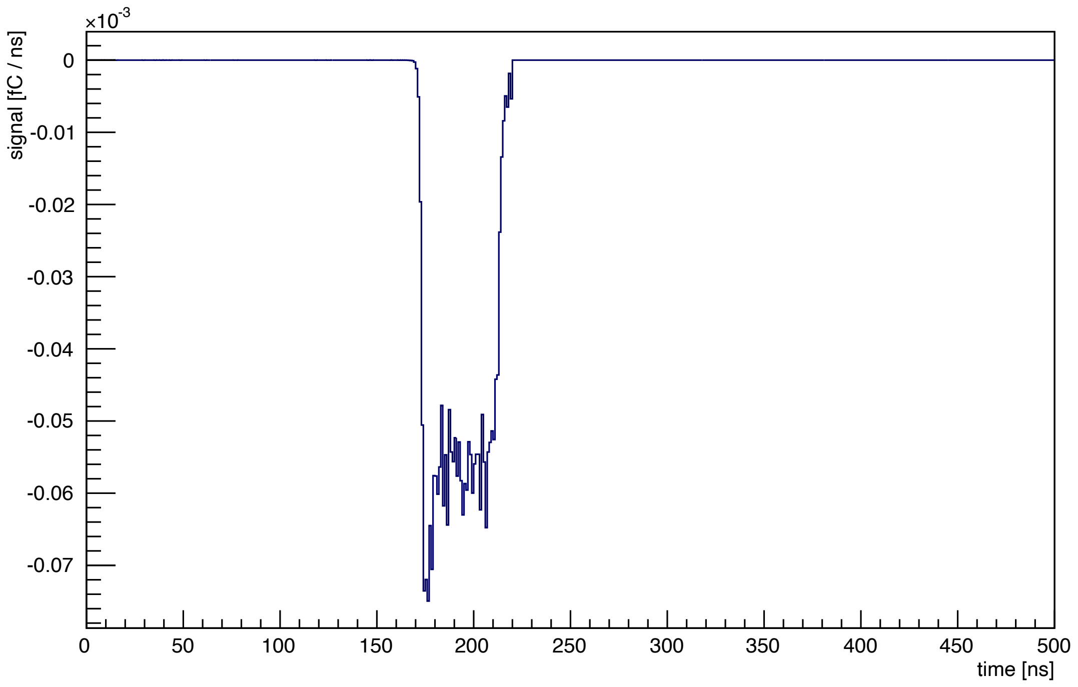

Figure 6: Electron induced signal produced on the lower readout electrode by the avalanche shown in figure 5.

4 Potential Issues/Errors

- *ElmerSolver ends with a segmentation fault shortly after execution.*
  Be sure you have the most updated version of Elmer, specifically if the failure occurs in the OptimizeBandwidth phase and you are using a large mesh ($>$ 150000 elements). Ensure that in your .geo file, the physical volumes and surfaces are numbered starting at 1, or the .msh file may not be created properly and Elmer will not find the correct number of element types.

- *ElmerSolver does not converge.*
  You may have to increase the `Linear System Max Iterations` parameter in the `Solver` section of your .sif file, but if each iteration does not seem to make significant progress towards convergence, this is unlikely to be the solution. It is more likely that a boundary condition corresponding to a physical surface was not set or labeled properly.

- *Upon mesh creation, Gmsh gives warning messages such as:*
  `Warning : Converged for i=0 j=1 (err=5.31079e-11 iter=5) BUT xyz error = ...`
  These messages will not cause the electrostatics calculation to fail. Reducing the size of the characteristic lengths, yielding a finer mesh, will help eliminate these warnings which seem to involve precision of the surface mesh.

- *At seemingly random locations in the finite element map, the potential returned is 0.*
  These mesh "holes" are a result of Garfield++ failing to find a finite element at a specific point in the map due to floating point error. One can fix this problem in specific cases by slightly relaxing the constraints on the tetrahedral coordinates returned when the element corresponding to a point in 3D space is identified. These coordinates must be in the range $t \in [0, 1]$, but a calculation may return coordinates very slightly less than 0 or greater than 1 if the point of interest is just on the boundary of two elements. This is not a clean, general solution to the problem, but in short one would change the lines:

  ```
  if (rc == 0 &&
      t1 >= 0 && t1 <= +1 &&
      t2 >= 0 && t2 <= +1 &&
      t3 >= 0 && t3 <= +1 &&
      t4 >= 0 && t4 <= +1) {
  ```

  in the element loop of `FindElement13` in `ComponentFieldMap.cc` to something like:

  ```
  if (rc == 0 &&
      t1 >= -1e-10 && t1 <= +1+1e-10 &&
      t2 >= -1e-10 && t2 <= +1+1e-10 &&
      t3 >= -1e-10 && t3 <= +1+1e-10 &&
      t4 >= -1e-10 && t4 <= +1+1e-10) {
  ```

Acknowledgements

The author of this document acknowledges support from a U.S. Department of Energy National Nuclear Security Administration Stewardship Science Graduate Fellowship, grant number DE-FC52-08NA28752.

The author would like to thank Adam Rubin (Weizmann Institute of Science) for fixing an error in the LEM avalanche example.

References

[1] H. Schindler, R. Veenhof, *et al.*, *Garfield++*. http://garfieldpp.web.cern.ch/garfieldpp.

[2] C. I. C. for Science, *Elmer: Open Source Finite Element Software for Multiphysical Problems, http://www.csc.fi/english/pages/elmer*.

[3] C. Geuzaine and J.-F. Remacle, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, pp. 1309–1331, 2009.

[4] J. D. Jackson, *Classical Electrodynamics*. Wiley, 1999.

[5] N. Ida and J. Bastos, *Electromagnetic Calculation of Fields*. Springer, 1997.

[6] C. Geuzaine and J.-F. Remacle, *Gmsh Manual, http://geuz.org/gmsh*.

[7] C. I. C. for Science, *ElmerGrid Manual (Jan. 20, 2010)*.

[8] C. I. C. for Science, *ElmerSolver Manual (Jan. 20, 2010)*.

[9] C. I. C. for Science, *ElmerModels Manual (Jan. 20, 2010)*.

[10] S. F. Biagi, "Monte Carlo simulation of electron drift and diffusion in counting gases under the influence of electric and magnetic fields," *Nucl. Instr. Meth. A*, vol. 421, pp. 234–240, 1999.

[11] H. Spieler, "Radiation Detectors and Signal Processing," *VII. Heidelberger Graduate Lectures in Physics*, 2001. http://www-physics.lbl.gov/ spieler/Heidelberg_Notes_2001/index.html.

[12] F. Sauli, "GEM: A new concept for electron amplification in gas detectors," *Nucl. Instr. Meth. A*, vol. 386, pp. 531–534, 1997.

[13] R. Chechik, A. Breskin, C. Shalem, and D. Mōrmann, "Thick GEM-like hole multipliers: properties and possible applications," *Nucl. Instr. Meth. A*, vol. 535, pp. 303–308, 2004.

[14] A. Sharma, "3D simulation of charge transfer in a Gas Electron Multiplier, (GEM) and comparison to experiment," *Nucl. Instr. Meth. A*, vol. 454, pp. 267–271, 2000.

[15] V. Tikhonov and R. Veenhof, "GEM simulation methods development," *Nucl. Instr. Meth. A*, vol. 478, pp. 452–459, 2002.

[16] C. Shalem, R. Chechik, A. Breskin, and K. Michaeli, "Advances in Thick GEM-like gaseous electron multipliers - Part I: atmospheric pressure operation," *Nucl. Instr. Meth. A*, vol. 558, pp. 475–489, 2006.

[17] A. Sharma, "A How-to Approach for a 3d Simulation of Charge Transfer, Characteristics in a Gas Electron Multiplier (GEM)," *(CERN Preprint)*.

[18] O. Bouianov, M. Bouianov, R. Orava, P. Semenov, and V. Tikhonov, “Progress in GEM simulation,” *Nucl. Instr. Meth. A*, vol. 450, pp. 277–287, 2000.

[19] A. Hammoud, E. Baumann, E. Overton, J. Suthar, and W. Khachen, “High Temperature Dielectric Properties of Apical, Kapton, Peek, Teflon AF, and Upilex Polymers,” *IEEE Conference on Electrical Insulation and Dielectric Phenomena*, pp. 549–554, 1992.