A Garfield++ interface for CST™

Klaus Zenker, DESY, Hamburg, Germany

August 5, 2013

Abstract

This document introduces an interface developed for Garfield++, which allows to use field data calculated with the finite element based program CST™. In addition, a comparison with a different finite element based program (ANSYS®) is presented.

1 Introduction

A powerful tool for simulations of gaseous and semiconductor detectors developed at CERN is called Garfield++ [1]. In order to perform simulations of complex structures, such as micro pattern gas detectors, up to now Garfield++ depends on external programs capable of simulating the fields arising in these structures. Here most often programs based on finite elements are used. One of such programs is CST™ [2], which solves Maxwell's equations in their integral form [3].

In order to use the results of CST™ in Garfield++ an interface is needed. This interface was written and is presented here. In the Sec. 2 the export of the field data produced by CST™ is explained. Afterwards, in Sec. 3, the interface itself is introduced followed by a comparison with a different finite element program (ANSYS®) in Sec. 4. Finally a summary is given in Sec. 5.

2 Exporting the field data of CST™

CST™ uses the finite element method, which means fields are calculated in finite elements with certain boundary conditions. Such finite elements are shown in Fig. 1. More in detail CST™ uses cubes and the field is calculated at eight nodes. This results in a regular mesh of cubes, which means e.g. that the x-y projection of the nodes is the same for all z positions. Nevertheless the cube size can vary within the considered volume. This is illustrated in Fig. 2.

The interface desires to export the resulting fields of a CST™ simulation in 4 different files (see also Appendix A). These files consist of the following information:

ELIST.lis: Element index e and the corresponding material.

NODELINES.lis: Node positions $n(x_i, y_i, z_i)$.

PRNSOL.lis: Potential of each node $\phi(n)$.

MPLIST.lis: Considered materials and their properties.

The name of the files can be chosen by the user and are here named after the files exported by ANSYS®, because they contain similar information in both cases. In contrast to ANSYS® the node positions in `NODELINES.lis` are saved in a vector like form since CST™ uses a regular grid:

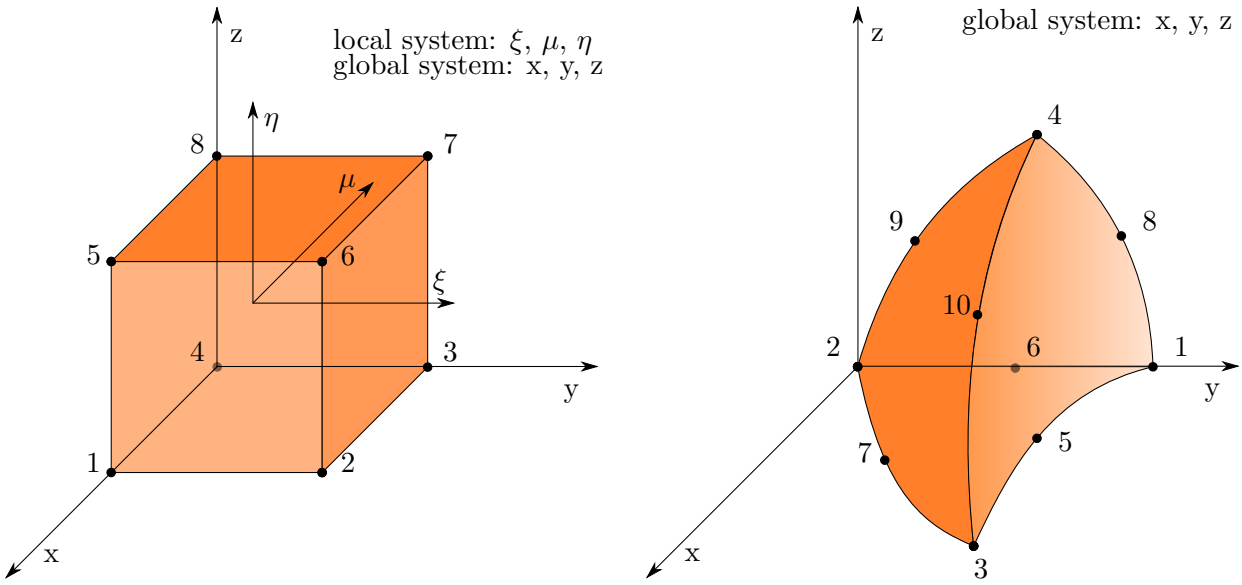


Figure 1: Finite elements used by CST™ (left) and ANSYS® (right).

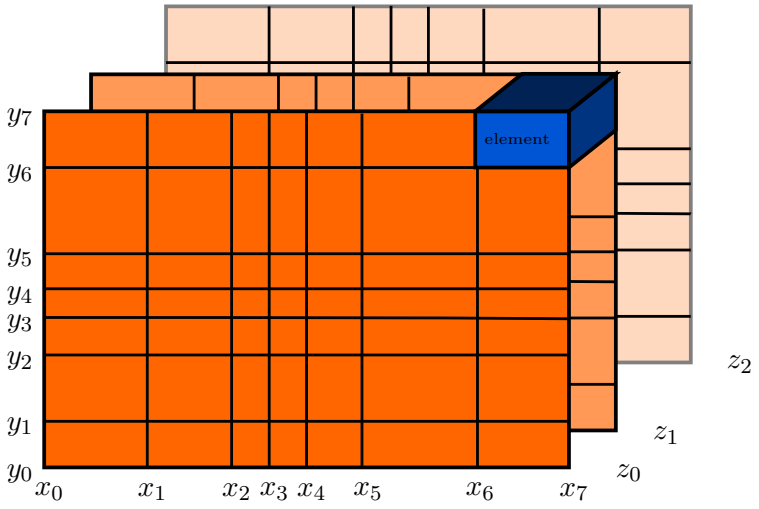


Figure 2: Sketch showing a mesh with variable element size.

x-lines (x_i)**:** $x_0, x_1, \ldots, x_{(N_x-2)}, x_{(N_x-1)}$

y-lines (y_j)**:** $y_0, y_1, \ldots, y_{(N_y-2)}, y_{(N_y-1)}$

z-lines (z_k)**:** $z_0, z_1, \ldots, z_{(N_z-2)}, z_{(N_z-1)}$

where N_x, N_y and N_z are the total number of nodes in x, y and z direction. Thus the index of node n can be calculated if the position in the vectors (i, j, k) is known:

$$n = i + jN_x + kN_xN_y \tag{1}$$

Consequently the highest node index is

$$n_{max} = N_xN_yN_z - 1. \tag{2}$$

The element index (e) can also be reconstructed with the help of the vectors stored in `NODELINES.lis`:

$$e = i + j\,(N_x - 1) + k\,(N_x - 1)\,(j - 1)\,. \tag{3}$$

$$e_{max} = (N_x - 1)\,(N_y - 1)\,(N_z - 1) - 1 \tag{4}$$

Note that in case of nodes, the index i is $N_x - 1$ at maximum and in case of elements i is $N_x - 2$ at maximum. This is because there are two boundaries in x direction for each element.

The assignment of a certain material to each element is not straight forward in case of CST™. During a simulation in CST™ an algorithm called *Perfect Boundary Approximation* (PBA) is used for treating material boundaries which cannot be build with cubes (see [4]). As a results of this algorithm the properties of an element do not correspond to a certain material, but rather a virtual mixed material is used. Since a discrete assignment between element and material is assumed by GARFIELD++ (e.g. to decide if an element belongs to the drift volume) here a cut based approach is used:

$$\text{material}\,(e_i) \quad = \quad \begin{cases} \text{kapton } (\varepsilon_r = 3.5) & \text{for } 3.4 < \varepsilon_r < 3.6 \\ \text{alumina } (\varepsilon_r = 9.4) & \text{for } 9.3 < \varepsilon_r < 9.5 \\ \text{G10 } (\varepsilon_r = 4.8) & \text{for } 4.7 < \varepsilon_r < 4.9 \\ \text{copper} & \text{for } \varepsilon_r = \infty \\ \text{drift medium} & \text{else} \end{cases} \tag{5}$$

where ε_r is the relative permittivity of the element. This assignment has been chosen, according to relative permittivity used in the CST™ simulation. Additional material definitions can be added to `MPLIST.lis` according to the materials used in the simulation with CST™.

3 Importing the field data of CST™ into Garfield++

GARFIELD++ provides interfaces to several simulation programs and it also provides classes to implement analytic fields. The major interface class is called `ComponentFieldMap` and consists of all the information about the described volume (here called component), e.g. size, type of material, potential at nodes if a finite element based program was used and more. A major function of this class is called `ComponentFieldMap::ElectricField`, where for a given space point the electric field in the component is calculated. Thus a new interface for CST™ (`ComponentCST`) needs to provide three main functions:

1. `ComponentCST::Initialise`: Read all information from `ELIST.lis`, `MPLIST.lis`, `PRNSOL.lis`, `NODELINES.lis`.
2. `ComponentCST::FindElementCube`: Find the element index for a given space point.
3. `ComponentCST::ElectricField`: Compute the electric field inside an element.

The base class `ComponentFieldMap` provides structures (e.g. element, node) for storing basic information. In addition, also the vectors of the mesh lines used for node and element position are stored in the derived class `ComponentCST` (`m_xlines`, `m_ylines`, `m_zlines`). The import of the information is implemented in `ComponentCST::Initialise` and the desired file format is given in Appendix A.

In order to implement the method `ComponentCST::FindElementCube(double x, double y, double z)` first of all the index in the node vectors is determined: if $x > x_i$ the index is i. With the help of the indexes i, j, k the element number is calculated as given in Eq. 3.

This allows the calculation of the electric field, which is the main part of the interface. After the element search the size of the cube in which the position in question is located can be calculated ($\Delta x = x_{\text{max}}^{\text{element}} - x_{\text{min}}^{\text{element}}, \Delta y, \Delta z$). The first step of the field calculation is a transformation from the global coordinate system (x, y, z) into a local coordinate system of the cube (ξ, μ, η, see Fig.

node q	1	2	3	4	5	6	7	8
ξ_q	-1	1	1	-1	-1	1	1	-1
μ_q	-1	-1	1	1	-1	-1	1	1
η_q	-1	-1	-1	-1	1	1	1	1

Table 1: Node positions in the local coordinate system.

1):

$$\xi = 2 \cdot \frac{x - x_{\min}}{\Delta x} - 1 \tag{6}$$

$$\mu = 2 \cdot \frac{y - y_{\min}}{\Delta y} - 1 \tag{7}$$

$$\eta = 2 \cdot \frac{z - z_{\min}}{\Delta z} - 1, \tag{8}$$

where $\xi, \mu, \eta \in [-1, 1]$. Here $x_{\min}$, $y_{\min}$ and $z_{\min}$ correspond to the boundary of the element. This allows a field calculation independent of the cube size. Referring to Fig. 1 the nodes of the cube can also be represented in the local coordinate system, which is shown in Tab. 1. In the first step the potential $\Phi(\xi, \mu, \eta)$ is calculated using so called *Shaping Functions* N_q and the potential at all nodes:

$$N_q = \frac{1}{8}(1 + \xi_q\xi)(1 + \mu_q\mu)(1 + \eta_q\eta) \tag{9}$$

$$\Phi(\xi, \mu, \eta) = \sum_{q=1}^{8} \Phi_q \cdot N_q, \tag{10}$$

where q stands for one node of the cube and ξ_q, μ_q, η_q needs to be substituted according to Tab. 1. Finally the electric field can be calculated:

$$\vec{E}(x, y, z) = -\vec{\nabla}_{x,y,z}\Phi(x, y, z) \tag{11}$$

$$E_x(x, y, z) = -\left(\frac{\partial\Phi(\xi, \mu, \eta)}{\partial\xi} \cdot \frac{\partial\xi}{\partial x} + \frac{\partial\Phi(\xi, \mu, \eta)}{\partial\mu} \cdot \frac{\partial\mu}{\partial x} + \frac{\partial\Phi(\xi, \mu, \eta)}{\partial\eta} \cdot \frac{\partial\eta}{\partial x}\right) \tag{12}$$

$$= -\sum_{q=1}^{8} \Phi_q \left(\frac{\partial N_q}{\partial\xi} \cdot \frac{\partial\xi}{\partial x} + \frac{\partial N_q}{\partial\mu} \cdot \frac{\partial\mu}{\partial x} + \frac{\partial N_q}{\partial\eta} \cdot \frac{\partial\eta}{\partial x}\right). \tag{13}$$

Since this is a coordinate transformation the partial derivatives can also be expressed by the *Jacobian* $\boldsymbol{J}$ of the transformation

$$\boldsymbol{J} = \begin{pmatrix} \frac{\partial x}{\partial\xi} & \frac{\partial y}{\partial\xi} & \frac{\partial z}{\partial\xi} \\ \frac{\partial x}{\partial\mu} & \frac{\partial y}{\partial\mu} & \frac{\partial z}{\partial\mu} \\ \frac{\partial x}{\partial\eta} & \frac{\partial y}{\partial\eta} & \frac{\partial z}{\partial\eta} \end{pmatrix}, \tag{14}$$

and

$$x = \sum_{q=1}^{8} x_{i,q} N_q \tag{15}$$

$$\frac{\partial x}{\partial\xi} = \sum_{q=1}^{8} x_{i,q} \frac{\partial N_q}{\partial\xi}, \tag{16}$$

where $x_{i,q}$ is the x position of the q^{th} node in the global coordinate system. Using the inverse of $\boldsymbol{J}$, Eq. 13 can be rewritten as follows:

$$\vec{E}(x,y,z) = -\boldsymbol{J}^{-1}\vec{\nabla}_{\xi,\mu,\eta}\Phi(\xi,\mu,\eta) \tag{17}$$

$$= -\sum_{q=1}^{8}\Phi_q^{\text{node}}\begin{pmatrix}\frac{\partial\xi}{\partial x} & \frac{\partial\mu}{\partial x} & \frac{\partial\eta}{\partial x}\\ \frac{\partial\xi}{\partial y} & \frac{\partial\mu}{\partial y} & \frac{\partial\eta}{\partial y}\\ \frac{\partial\xi}{\partial z} & \frac{\partial\mu}{\partial z} & \frac{\partial\eta}{\partial z}\end{pmatrix}\begin{pmatrix}\frac{\partial N_q}{\partial\xi}\\ \frac{\partial N_q}{\partial\mu}\\ \frac{\partial N_q}{\partial\eta}\end{pmatrix}. \tag{18}$$

The calculation of weighting fields, implemented in `ComponentCST::WeightingField`, is equivalent to the calculation of the electric field.
The corresponding function `ComponentCST::SetWeightingField` expects a file similar to `PRNSOL.lis` and works like the corresponding part of `ComponentCST::Initialise`. More information about weighting fields can be found in Ref. [5].

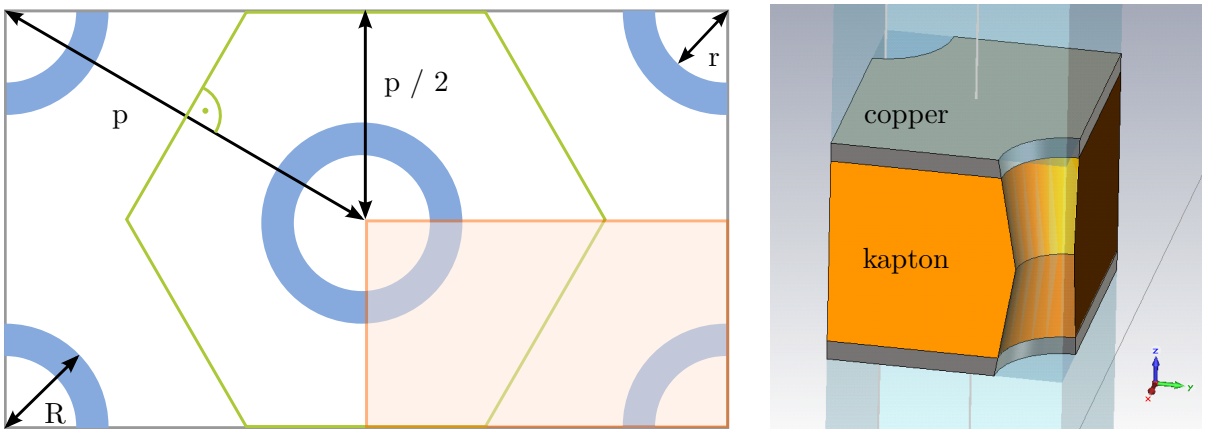
Finally, it is also possible to visualise the mesh used by CST™ in Garfield++ using the class `ViewFEMesh`. Here the function `ViewFEMesh::DrawCST` was added to allow the drawing of a $x-y$, $y-z$ or $x-z$ projection of the mesh.

4 Example: GEM simulation

In order to demonstrate the performance of the interface, here a simulation of a *Gas Electron Multiplier* (GEM) is presented. Figure 3 shows the geometry of a GEM and the considered basic cell used in the field simulation. Applying mirror symmetries in Garfield++ the GEM structure can be realised using this basic cell, which is depicted by the orange box in Fig.3(a). The modelled GEM corresponds to GEMs produced at CERN with the following specifications:

- Double conical holes
- $p = 140\,\mu\text{m}$
- $R = 35\,\mu\text{m}$, $r = 25\,\mu\text{m}$
- $h_{\text{copper}} = 5\,\mu\text{m}$, $h_{\text{kapton}} = 50\,\mu\text{m}$

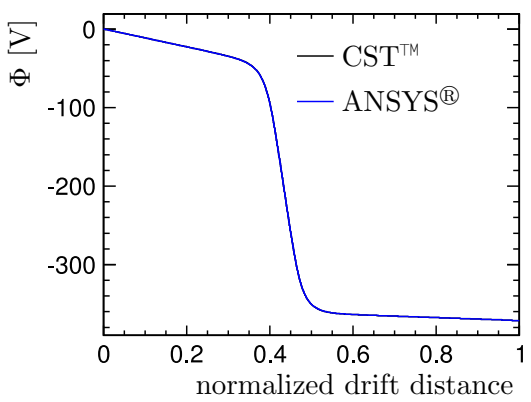
The drift space in the simulation was set to $d_{\text{drift}} = 400\,\mu\text{m}$ and the induction space was set to $d_{\text{induction}} = 300\,\mu\text{m}$.



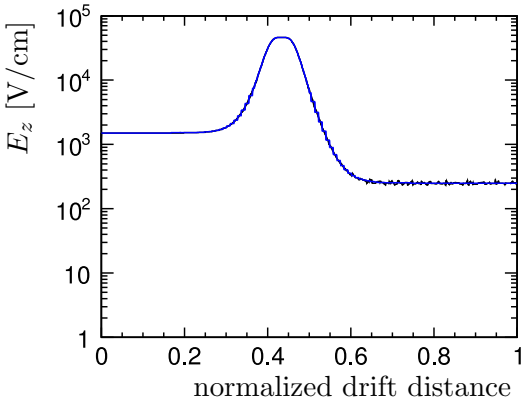
(a) Sketch of a GEM showing also the basic cell considered in the simulation (orange box).

(b) Model of the GEM basic cell created with CST™.

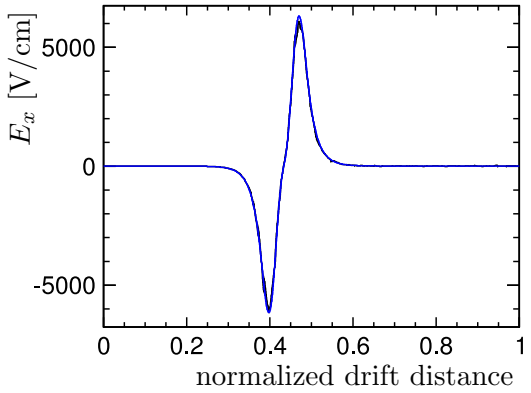
Figure 3: GEM characteristics.



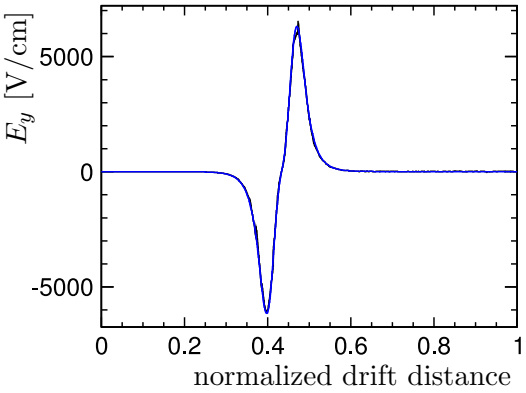
(a) Potential.



(b) Electric field in drift direction.



(c) Electric field transverse to the drift direction.



(d) Electric field transverse to the drift direction.

Figure 4: Potential and fields resulting from a scan in drift direction through the centre of a GEM hole in Garfield++. The drift starts in the drift volume at a position corresponding to the normalised drift distance of 1.

Furthermore, the drift gas in the Garfield++ simulation was chosen to be P5 (95 % Ar, 5 % CH_4). Corresponding to this gas the following voltages were applied:

- drift boundary: −371.57 V
- induction boundary: 0 V
- GEM cathode: −368.38 V
- GEM anode: −38.38 V

After exporting the field simulation data and importing them in Garfield++ the resulting fields and the potential are shown in Fig. 4. The same simulation was repeated in ANSYS® and the results are also shown in Fig. 4. Good agreement between the results of CST™ and ANSYS® can be seen. Finally 100 electrons were released in the drift field and when these electrons reach the GEM an avalanche develops. The mean number of produced electrons in case of CST™ is $\overline{n_e} = 56$ and $\overline{n_e} = 59$ in case of ANSYS®. This is shown in Fig. 5. Both results agree well taking the small number of events into account. This agreement could be expected from the similar fields shown in Fig. 4. The important difference is the computing time for the simulation with Garfield++: $t_{\mathrm{CST}} = 10\,\mathrm{min}$ and $t_{\mathrm{ANSYS}} = 157\,\mathrm{min}$. This shows the advantage of the regular grid used in CST™, which speeds up the element search compared to ANSYS®. Especially for a simulation of a GEM stack this fact is relevant.

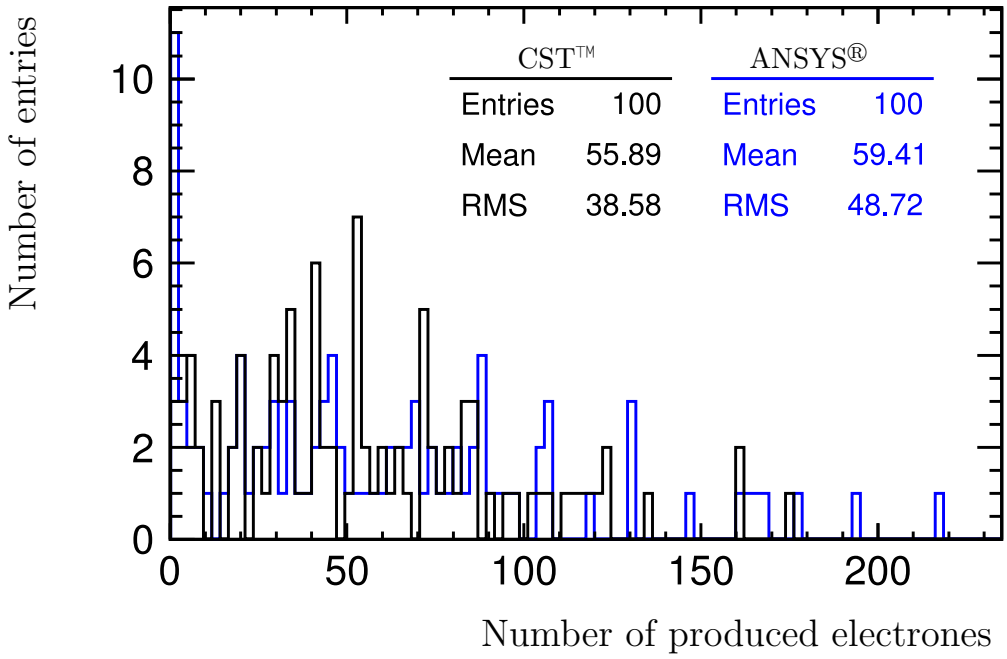


Figure 5: Number of produced electrons in P5 gas by a single GEM avalanche for CST™ and ANSYS®.

5 Summary

A detailed description of a developed interface for CST™ in Garfield++ was presented. This includes the export of field data with CST™, the import of these data in Garfield++ and the calculation of the field in Garfield++ for any given point. This required the implementation of an element search and a field interpolation inside of the finite elements used by CST™. Here the advantages of the regular grid used by CST™ were taken into account, resulting in a very fast and efficient element search and reducing the computing time in Garfield++ a lot. This was proven in comparison to another commonly used finite element program – ANSYS®. The fields resulting from the CST™ field calculation were similar to those of ANSYS®. Thus, this interface provides the opportunity of using CST™ as an fast but not less precise alternative field simulation program to be used with Garfield++.

A File formats

`NODELINES.lis` (mesh coordinates):

```
xmax 136 ymax 79 zmax 425
x-lines
0
8.92857e-07
1.78571e-06
...
y-lines
0
8.92857e-07
1.78571e-06
...
z-lines
0.0027
0.00270674
0.00271348
...
```

`ELIST.lis` (element nb.; material):

```
 0      3
 1      3
 2      1
 3      1
 4      2
 5      1
...
```

`PRNSOL.lis` (node nb.; potential):

```
  0   0
  1   12
  2   15
  3   17
  4   18
  5   21
...
```

`MPLIST.lis` (material def. and index):

```
 Materials 4

 Material 1  PERX 1.000000

 Material 2  RSVX 0.000000 PERX 0.1000000E+11

 Material 3  PERX 3.500000

 Material 4  PERX 4.800000
```

Figure 6: These are excerpts of the output files of a CST™ simulation formatted as desired by the CST™ interface in Garfield++.

References

[1] Garfield++ project web page. `http://garfieldpp.web.cern.ch/garfieldpp/`.

[2] CST homepage. `www.cst.com`.

[3] M. Clemens and T. Weiland. Discrete electromagnetism with the finite integration technique. *Progress In Electromagnetics Research*, 32:65–87, 2001.

[4] B. Krietenstein, R. Schuhmann, P. Thoma, and T. Weiland. The perfect boundary approximation technique facing the big challenge of high precision field computation. In *Proceedings of the XIX International Linear Accelerator Conference (LINAC 98), Chicago, USA*, pages 860–862, 1998.

[5] Garfield++ user guide, March 2012. `http://garfieldpp.web.cern.ch/garfieldpp/documentation/`.