


Universidade de São Paulo

Instituto de Física-USP

# Simulações de detectores à gás utilizando os softwares Gmsh, Elmer e Garfield++

**Um guia em português**

# Gaseous detectors simulation using Gmsh, Elmer and Garfield++

**A guide in Portuguese**

Geovane Grossi Araújo de Souza
Contato: geovane.souza@usp.br

# Sumário

## Resumo

Esse guia foi elaborado em português com o intuito de ajudar uma maior quantidade de pessoas na utilização dos softwares GMSH, Elmer e Garfield++, e está voltado ao estudo de detectores de radiação à gás e parâmetros relacionados à difusão de elétrons em gases. Referencias mais completas em inglês podem ser encontradas aqui.

Caso tenham algo a complementar ou corrigir à este guia, por favor, entrem em contato comigo pelo e-mail: geovane.souza@usp.br



## Abstract

This guide was developed in Portuguese in order to help a greater number of people in the use of the GMSH, Elmer and Garfield ++ software, and is aimed at the study of gaseous radiation detectors and parameters related to electron diffusion in gases. More complete references in English can be found here.

If you have anything to add or correct to this guide, please contact me at e-mail: geovane.souza@usp.br

# 1 Repositório no git com exemplos

Antes de começar a ler esse guia, você pode instalar o git no seu computador(acredito que ele já vem por padrão no Linux) e clonar o repositório com alguns exemplos.

Para isso execute o seguinte comando no terminal

```
git clone https://github.com/geovanegrossi/Garfield-tutorial.git
```

Nele você vai encontrar uma célula GEM(Gas Electron Multiplier)[1] padrão que já pode ser utilizada com os dois exemplos do Garfield++. O primeiro (completeavalanche) calcula uma avalanche em um GEM e salva a posição inicial e final dos elétrons e íons em um txt. Com essa saída você pode criar uma TTree no ROOT[10] para análise.

O seguindo exemplo (viewfield) plota um mapa do campo elétrico ao redor de um buraco e o campo elétrico em valor absoluto ao longo de uma linha que passa por dentro de um dos buracos.

# 2 Instalação

# 3 Introdução

O Garfield++ [2] é o programa principal abordado nesse tutorial com enfase em simulações de detectores gasosos. Ele foi desenvolvido no CERN, utilizando as bibliotecas compartilhadas em Fortran do antigo Garfield [3], criado por Rob Veenhof, criado em 1984.

Uma das principais bibliotecas do Garfield é a MAGBOLTZ, escrita por Stephen Biagi [4], responsável por calcular as probabilidade de interação entre os elétrons e moléculas do gás. Os cálculos do MAGBOLTZ contêm as probabilidades de colisão levando em consideração os campos elétricos e efeitos adjacentes que serão explicados mais adiante.

Para trabalhar com o Garfield++ é necessário fornecer dois arquivos de entrada diferentes, a geometria do problema(ou do detector/estrutura de multiplicação) e os campos elétricos. Esses arquivos são obtidos utilizando dois softwares diferentes, o GMSH e o Elmer.

Nesse tutorial você vai encontrar as principais funções desses 3 programas. Um outro guia muito mais completo e interessante que fala sobre a utilização desses 3 softwares pode ser encontrada aqui [5].

# 4 Gmsh

O GMSH [6] é o programa responsável por criar o modelo 3D do detector.

## Criando seu modelo em 3D

## Declarando pontos

No Garfield++ é possível multiplicar as células unitárias, gerando assim o objetivo tridimensional completo. Portanto, devemos criar no GMSH somente a célula que será replicada em todo o espaço.

O arquivo padrão de geometria do GMSH é o **.geo** e podemos monta-lo em um arquivo de texto ou em um bloco de notas.

Um ponto no gmsh é declarado da seguinte maneira:

```
Point(1) = {x1, y1, z1, ref};
```

onde x1, y1, z1 são as coordenadas cartesianas e **ref** é a variável ligada ao refinamento do método de elementos finitos ao redor desse ponto. Vamos trabalhar com as unidades em mícrons.

Variáveis podem ser declaradas da seguinte maneira:

```
ref= 5; // Essa grandeza diz respeito ao refinamento da mesh que será gerada.
 Quando menor o valor de ref maior o refinamento
R_c=35; //Raio do buraco do cobre
```

Para facilitar a escrita do arquivo, você pode até fazer operações dentro do arquivo **.geo**, por exemplo:

```
Point(2) = {x1, y1+R_c, z1/2, ref};
```

A qualquer momentos você pode salvar o seu arquivo **.geo** e abri-lo com o aplicativo do GMSH. Dessa forma, você pode visualizar se os pontos e as linhas criadas fazem sentido e estão corretos. Pode ser mais simples colocar todos os pontos da maneira escrita e utilizar o aplicativo do GMSH somente para ligá-los.

## Linhas e círculos

Para ligar dois pontos utilizamos os comandos que criam linhas. Para criar uma linha pelo programa, abra o gmsh, carregue seu arquivo de geometria e vá em **Modules > Geometry > Elementary entities > Add> Straight line** e selecione os dois pontos que quer ligar.

Equivalente, pode-se escrever o comando *line* para se ligar os pontos pelo arquivo de texto:

```
Point(10) = {x1, y1, z1, ref};
Point(11) = {x2, y2, z2, ref};
Line(1) = {10, 11};
```

Um arco de circunferência é gerado pelo mesmo caminho. Selecione a opção **Circle Arc** e escolha 3 pontos. O primeiro é o ponto inicial do arco da circunferência, o segundo o centro do círculo e o terceiro o ponto final do arco. De maneira equivalente, podemos utilizar o comando *Circle*:

```
Circle(39) = {508, 501, 502};
```

## Superfícies

Após conectar todos os pontos o próximo passo é gerar as superfícies. Para isso vá em **Modules > Geometry > Elementary entities > Add> Plane surface**. Um loop contendo 4 linhas vai definir uma superfície. Caso sua superfícies seja curva, você terá que usar a opção **Ruled surface**.

Pela linha de comando podemos utilizar o seguinte:

```
Line Loop(1) = {51, 52, 53, 54}; //Loop entre as linhas 51 a 54
Plane Surface(1) = {1};

Line Loop(2) = {55, 57, 58, 59};
Ruled Surface(2) = {2};
```

## Volumes

Da mesma maneira que linhas são selecionadas para gerar superfícies, podemos selecionar superfícies para criar os volumes. Para isso vá em **Modules > Geometry > Elementary entities > Add> Volume**.

Podemos também utilizar do seguinte comando:

```
Surface Loop(3) = {1, 2, 3, 4, 5, 6, 7, 9};  //seleção do volume definido pelas
superficies de 1 a 9
Volume(2) = {2};
```

## Corpos físicos

Todas as superfícies ou volumes que vão receber algum tipo de condição de contorno ou que serão atribuídos a eles materiais no Elmer, devem ser necessariamente transformadas em superfícies ou volumes físicos. Para isso vá em **Modules > Geometry > Physical groups > Add > Surface/Volume**.

Os seguintes comandos também podem ser utilizados:

```
Physical Volume(668) = {256}; //volume original 256 se torna o volume físico com indice
Physical Surface(740) = {20};
```

**Importante!** O Elmer irá identificar os corpos pelo índice atribuído as suas entidades físicas. Por exemplo, no caso mostrados acima, o Elmer vai encontrar a superfície número 740 e o volume 668 e não 256 e 20.

**Atenção!** Superfícies que vão receber condições de contorno periódicas no Elmer, também precisam ser declaradas como superfícies físicas.

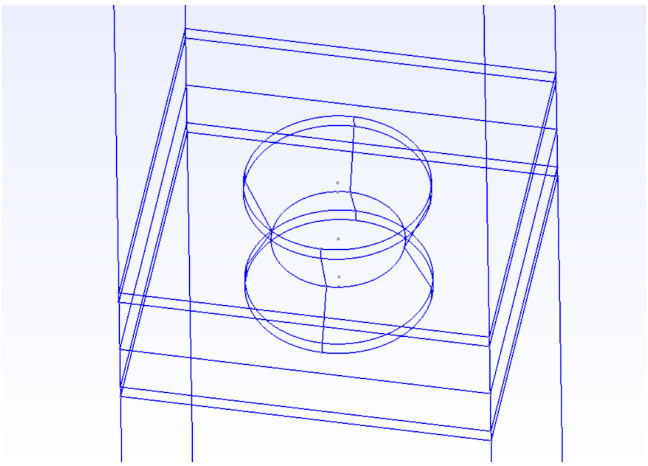


**Figura 1:** A célula unitária do arquivo GEM_example.geo

## Gerando a mesh

Após toda a definição de superfícies e volumes físicos, podemos gerar a mesh no terminal da seguinte maneira:

```
gmsh GEM_example.geo -3 -optimize -order 2
```

Onde -3 é o indicativo de uma mesh em 3 dimensões, optimize otimiza a qualidade dos elementos da mesh(sem essa opção não foi possível carregar o arquivo no Garfield++). Order 2, segundo o menu de ajuda do GMSH, está relacionada com os elementos de segunda ordem da mesh. Após executar esse comando um arquivo .msh é gerado no mesmo diretório. Esse arquivo pode ser lido diretamente pelo Elmer.

Os arquivos contendo os elementos da mesh que são entrada tanto para resolução dos potenciais do Elmer quanto para calculo do Garfield++ podem ser gerados utilizando o **ElmerGrid**, um programa adjacente do Elmer que já vem na instalação.

Para gerar a lista de arquivos deve-se executar o seguinte comando no terminal:

```
ElmerGrid 14 2 GEM_example.msh
```

14 indica que o arquivo de entrada foi fornecido pelo GMSH e 2 é para a saída ser dada em arquivos compatíveis com ElmerSolver.

# 5 Elmer

O Elmer [7] é um programa capaz de resolver equações diferenciais para diferentes problemas de física, como por exemplo mecânica estrutural, eletro magnetismo, dinâmica de fluidos e transferência de calor. É no Elmer que o usuário assinala quais materiais compõe o detector.

O chamado *Weighting Field*($\vec{\epsilon}_{\mathrm{w}}$) é um campo criado onde todos os eletrodos do problema estão aterrados, menos um, onde um potencial de 1 V é aplicado. Segundo o teorema de Shockley-Ramo [8], a corrente induzida em um determinado eletrodo devido ao movimento de uma carga é dado por:

$$i_k(t) = q\vec{v}(\mathrm{t})\vec{\epsilon}_{\mathrm{w}} \tag{1}$$

onde $q$ é a carga, $\vec{v}(\mathrm{t})$ é a velocidade de deriva, que pode ser escrita da seguinte maneira:

$$\vec{v}(\mathrm{t}) = \mu\vec{\epsilon}_{\mathrm{d}} \tag{2}$$

sendo $\mu$ a mobilidade do portador de carga e $\vec{\epsilon}_{\mathrm{d}}$ o campo de deriva.

## Gerando seu arquivo .sif

Vamos precisar de dois arquivos **.sif**, o primeiro dele contém os potenciais aplicados a cada um dos eletrodos do problema e o segundo é o arquivo com o *weighting field*. O arquivo **.sif** carrega todas as atribuições de materiais, condições de contorno, potenciais e tipo de problema ser resolvido. A primeira vez que for gerar esse arquivo, a melhor maneira é utilizando o programa ElmerGUI (durante a instalação do Elmer, pode-se alterar uma opção para instala-lo junto).

Abrindo o ElmerGUI, carregue a pasta completa de arquivos gerada após fazer o comando do ElmerGrid no terminal(utilizando o comando **Load Mesh**). Em seguida vá até *Model* e adicione uma equação(no caso desse exemplo, *Electrostatics*). Selecione todos corpos onde se quer aplicar essa equação e clique em *OK*.

Novamente em *Model* vá até *Material* e adicione os materiais do problema(alguns já estão disponíveis na biblioteca interna do Elmer). Na aba *Electrostatics* coloque a permissividade relativa do corpo. Para finalizar dê um nome ao material criado e selecione os corpos que serão formados por esse material.

Mais uma vez em *Model*, vá até *Boundary Conditions*. Ali selecione a aba *Electrostatics*, preencha o valor do potencial, configure outras opções caso deseje, dê um nome a essa configuração e selecione as superfícies que vão receber esse potencial. Nessa mesma parte é possível

selecionar as condições periódicas de contorno em corpos simétricos, impondo translações e rotações. Essas condições são ativadas na aba *General*. Consulte o guia do Elmer para mais informações.

Após completar todas essas etapas, vá na aba *Sif*, selecione *Edit*. Confira todas as informações do problema e na opção *Post file* você tem duas saídas possíveis. A primeira delas e provavelmente padrão do Elmer é a **file.vtu**. Esse é um arquivo de resultados padrão para visualização dos campos pelo software *Paraview*[9]. O arquivo que queremos gerar para trabalhar no Garfield++ tem a extensão **.result**. Faça essa alteração, salve o arquivo e vá na opção *Save Project*.

Quando salvar o projeto verifique se o arquivo ELMERSOLVER_STARTINFO foi gerado junto com o **.sif**. Caso ele não tenha sido gerado, crie um arquivo sem extensão com esse nome (ELMERSOLVER_STARTINFO) e coloque o nome do arquivo **.sif** a ser resolvido nele.

```
file.sif
```

Para gerar os arquivos com o resultado do Elmer, execute o seguinte comando no terminal:

```
ElmerSolver Field.sif
```

Lembre-se que o **.sif** e os **mesh.** gerados pelo gmsh precisam estar no mesmo diretório.

Caso encontre alguma dificuldade neste ponto, consulte os arquivos do diretório *git*. Nele é possível encontrar exemplos dos **.sif** necessários.

# 6 Outros arquivos necessários para o Garfield++

Outro dois arquivos são necessários para rodar o Garfield++. O primeiro deles é o arquivo chamado **dielectrics.dat**. A montagem desse arquivo é simples. Na primeira linha se coloca quantos volumes físicos foram utilizados na geometria e nas linhas seguintes, o índice do material seguido da constante dielétrica do mesmo, por exemplo, um geometria com duas chapas de cobre sobre um material dielétrico com constante dielétrica 3.5 ficaria:

```
3
1 1e10
2 1e10
3 3.5
```

Os arquivos de mobilidade de Ions também precisam ser fornecidos. A maioria deles pode ser encontrada na pasta **$GARFIELD_HOME/Data** e tem o nome parecido com **IonMobility_Ar+_Ar.txt**.

# 7 Garfield++

Antes de começar a falar sobre o Garfield++, vale lembrar que ele foi escrito de forma a atuar com o ROOT[10], e ele também é orientado à objeto. Assim, vale a pena tomar algum tempo para aprender a utilizar o ROOT para aproveitar de suas funcionalidades.

## As principais classes do Garfield++

Para calcular o gás no Garfield++, a classe responsável é a **MediumMagboltz**.

```
double rp=0.45; //valor para de transferencia do efeito penning
double ArgonConcent = 90.0; //Concentração de Ar
double CO2Concent = 10.0; //Concentração de CO2
MediumMagboltz* gas = new MediumMagboltz();
gas -> SetTemperature(298.15); // Temperature [K]
gas -> SetPressure(740.);      // Pressure [Torr]
gas -> EnableDrift();
gas -> SetComposition("Ar", ArgonConcent, "CO2", CO2Concent);
gas -> SetMaxElectronEnergy(200.);   // Energy [eV]
gas -> EnableDebugging();
gas -> Initialise();
gas -> DisableDebugging();
gas -> LoadIonMobility("IonMobility_Ar+_Ar.txt");
gas -> EnablePenningTransfer(rp, 0.0, "ar");
```

Como podemos ver é possível adicionar ou não a contribuição do efeito Penning [11] no cálculo das interações e ionizações do gás. Resumidamente, o efeito Penning está ligado com ionizações indiretas das moléculas do gás auxiliar(nesse caso o $CO_2$), causado por desexcitações do Argon. Os valores de $rp$, ou seja, das probabilidade de ionização indiretas variam com diversos fatores, entre eles pressão e concentração, dessa forma é necessário consultar na literatura o valor de $rp$ para o seu caso em específico [12]

A próxima classe do Garfield++ é a **ComponentElmer**.

```
ComponentElmer * elm = new ComponentElmer("/files/mesh.header",
"/files/mesh.elements","/files/mesh.nodes", "/files/dielectrics.dat",
"/files/GEM.result","micron");
elm -> EnablePeriodicityX();
elm -> EnableMirrorPeriodicityY();
elm -> SetWeightingField(wtlel,"wtlel");
```

```
elm -> SetMedium(0, gas);
elm -> PrintMaterials();
elm -> PrintRange();
```

Olhando elemento à elemento:

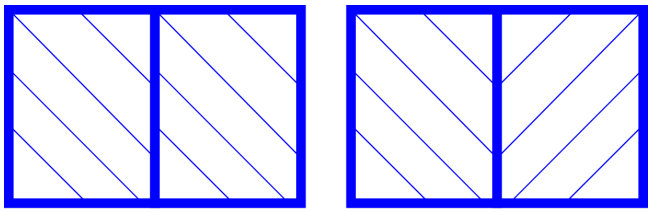
```
ComponentElmer * elm = new ComponentElmer("/files/mesh.header",
"/files/mesh.elements", "/files/mesh.nodes", "/files/dielectrics.dat",
"/files/GEM.result","micron");
```

São os arquivos de geometria finais do GMSH e o arquivo **.dat** citado anteriormente junto com o *.result*, arquivo de saída do Elmer. Micron é a escala de tamanho que essas geometrias estão, definido no GMSH.

---

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

As duas formas de periodicidade existentes podem ser vistas abaixo:




---

```
elm -> SetWeightingField(wtlel,"wtlel");
```

O chamado *Weighting Field*($\vec{\epsilon}_{\mathrm{w}}$), explicado anteriormente, é necessário para coleta de sinais em um eletrodo específico.

---

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

Designa o meio ionizante.

---

```
elm -> PrintMaterials();
elm -> PrintRange();
```

É interessante colocar esse comando para verificar o potencial aplicado, assim como das dimensões da geometria carregada e os parâmetros dos materiais.

---

A classe *sensor* é muito importante e vai ser chamada pelas classes responsáveis pro avalanches. A função *SetArea* vai dar os limites de calculo da simulação(área do detector).

```
Sensor* sensor = new Sensor();
sensor->AddComponent(elm);
sensor->SetArea(-x_dim,-y_dim,-z_dim,x_dim,y_dim,z_dim);
sensor->AddElectrode(elm,"wtlel");
```

---

A classe **AvalancheMicroscopic** calcula a avalanche e a deriva dos elétrons. **EnableSignalCalculation** é necessário caso se queira coletar gerado pela carga induzido em um dos eletrodos. **AvalancheElectron** inicia o calculo da avalanche onde xe1,ye1,ze1 e te1 são as coordenadas iniciais, e1 é a energia do elétron e dxe,dye e dze formam um vetor com a direção inicial.

```
AvalancheMicroscopic* aval = new AvalancheMicroscopic();
aval->SetSensor(sensor);
aval->SetCollisionSteps(100);
aval->EnableSignalCalculation();
aval -> AvalancheElectron(xe1, ye1, ze1, te1, e1, dxe, dye, dze);
```

---

A classe **AvalancheMC** calcula a deriva dos íons. **DriftIon** inicia a deriva para os ions com as coordenadas inciais xe1, ye1, ze1 e te1.

```
AvalancheMC* drift = new AvalancheMC();
drift->SetSensor(sensor);
drift->SetDistanceSteps(5.e-4); // Integrate in constant (2 um) distance intervals.
drift->EnableSignalCalculation();
drift->DriftIon(xe1, ye1, ze1, te1);
```

---

Algumas outras funções importantes ligadas aos objetos anteriores são:

```
aval->GetElectronEndpoint(l, xe1, ye1, ze1, te1, ee1,
xe2, ye2, ze2, te2, ee2, e_status);
drift->GetIonEndpoint(l, xi1, yi1, zi1, ti1, xi2, yi2, zi2, ti2, i_status);
```

Essas duas funções recuperam as coordenadas iniciais e finais de uma trajetória. ***l*** é o índice do elétron/íon dentro de uma avalanche e ***status*** fornece informação sobre a condição final dos portadores de carga. Os códigos mais comuns que apareceram nas simulações são:

| Processo | Descrição |
|---|---|
| -1 | Partícula saiu da região delimitada pelo sensor |
| -3 | Cálculo abandonado (Error) |
| -5 | Partícula saiu do meio de deriva (Chegou em um eletrodo por exemplo) |
| -7 | attachment (elétron quebrou uma molécula e pode ser absorvido) |

A lista completa de status pode ser obtida no manual do Garfield++[2].

---

## Ferramentas para visualização de linhas de deriva

Para habilitar a visualização da deriva tanto dos elétrons quantos dos íons, deve-se chamar a classe **ViewDrift**.

```
ViewDrift* viewDrift = new ViewDrift();
viewDrift->SetArea(-1*axis_x,-1*axis_y,-1*axis_z,1*axis_x,1*axis_y,1*axis_z);
aval -> EnablePlotting(viewDrift);
drift -> EnablePlotting(viewDrift);
```

Nesse caso, a função **SetArea** vai dar a região onde será guardada a informação das linhas de deriva.

Com **ViewGEMesh** é possível visualizar as formas da geometria carregada.

```
ViewFEMesh * vFE = new ViewFEMesh();
vFE->SetCanvas(cGeom);
// É necessário declarar um Canvas onde será desenhado  a imagem
vFE->SetComponent(elm);
vFE->SetPlane(0,-1,0,0,0,0); //vetor que dá a direção do plano de visualização
vFE->SetFillMesh(true);
vFE->SetColor(1,kGray+3); //cor dos corpos do problema
vFE->SetColor(2,kGray+3);
vFE->SetColor(3,kGray+3);
vFE->SetColor(4,kAzure+2);
vFE->EnableAxes();
vFE->SetXaxisTitle("x (cm)");
vFE->SetYaxisTitle("z (cm)");
vFE->SetArea(-0.015,-0.015,0,0.015,0.015,0);
// Area que será visualizada no Canvas do root
```

Após realizar a avalanche, utilize os seguintes comandos para visualização:

```
vFE->SetViewDrift(viewDrift);
vFE->Plot();
```

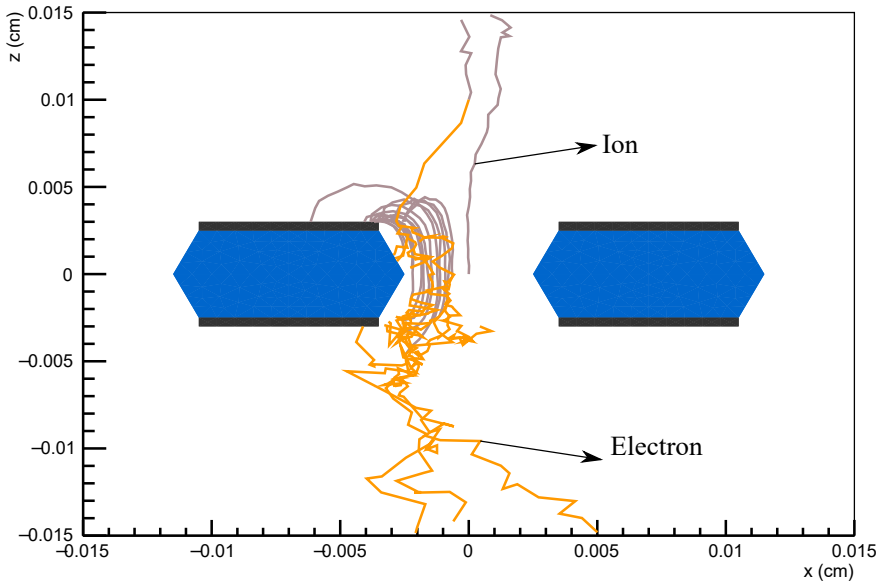



**Figura 2:** Exemplo de uma avalanche dentro de um GEM.

## Ferramentas para visualização do campo

É possível visualizar pelo Garfield++ o campo elétrico absoluto ou suas componentes, assim como o potencial de uma simulação. Para isso a classe **ViewField** é utilizada.

```
TCanvas* eField = new TCanvas("eField", "Electric Field");
ViewField * vf = new ViewField();
vf->SetComponent(elm);
vf->SetArea(-1*axis_x,-1*axis_y,axis_x,axis_y);
vf->SetNumberOfSamples2d(300,300); //Número de pontos para gerar imagem
vf->SetPlane(0,-1,0,0,0,0); //vetor diretor do plano de visualização
vf->SetCanvas(eField);
vf->PlotContour("e");
```

A função **PlotContour** oferece uma lista de opções para visualização:

Caso selecionado a opção “p”, “v”ou “potential”, o campo a imagem vai gerar o potencial. A string “e”fornece o campo elétrico e as opções “ex”, “ey”e “ez”fornecem as componentes de campo elétrico em cada uma das coordenadas cartesianas.

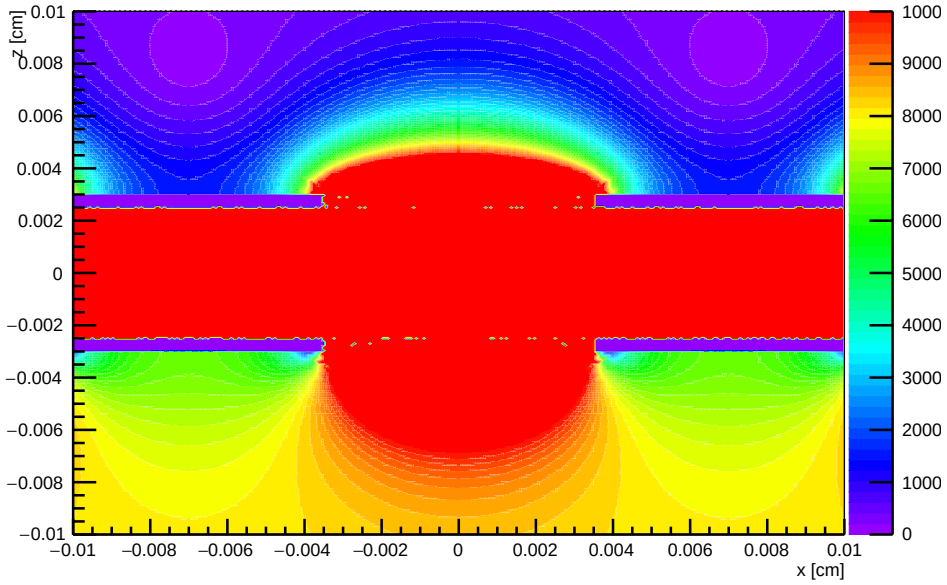



**Figura 3:** Campo elétrico dentro de um GEM.

Da mesma maneira, podemos fazer um perfil do campo elétrico utilizando a função **PlotProfile**.

```
TCanvas* eProfileElectricField = new TCanvas("Profile", "Profile");
ViewField * pf = new ViewField();
pf->SetComponent(elm);
pf->SetArea(-1*axis_x,-1*axis_y,1*axis_x,1*axis_y);
pf->SetNumberOfContours(40);
pf->SetPlane(0,1,0,0,0,0);
pf->SetCanvas(eProfileElectricField);
pf->PlotProfile(0,0,-1*axis_z,0,0,1*axis_z,"e");
```

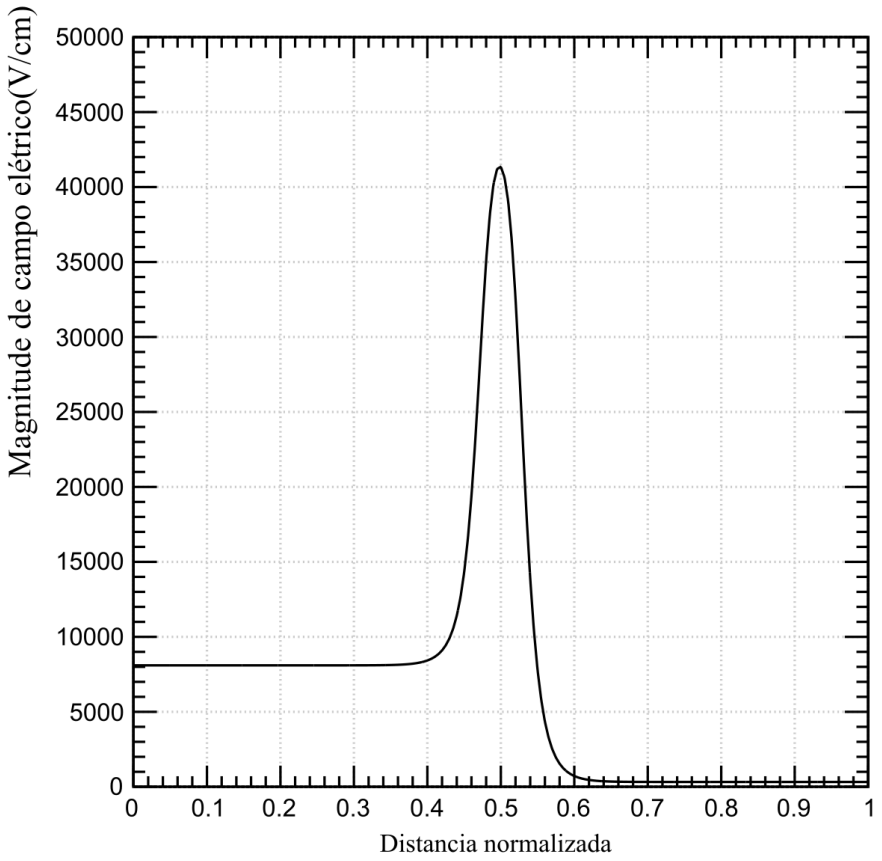



**Figura 4:** Magnitude de campo elétrico em V/cm por distancia normalizada

# HEED

O HEED[13] é um programa capaz de fazer a interação da radiação com o gás, gerando assim as partículas produtos dessa interação. É possível utiliza-lo para fazer o lançamentos de fótons e algumas partículas. Para lançar um fóton, pode-se utilizar do seguinte comando:

```
TrackHeed* track = new TrackHeed();
track -> SetSensor(sensor);
track -> EnableElectricField();
track -> TransportPhoton(x0, y0, z0, t0, energy, dx0, dy0, dz0, nsum);
```

Onde x0, y0, z0, e t0 são as coordenadas de posição e tempo do lançamento do fóton, dx0, dy0 e dz0 e o vetor que dá a direção desse fóton e nsum é o número de elétrons que foi gerado na interação.

Caso seja de interesse recuperar a posição, tempo e energia cinética dos elétrons gerados após interação, o HEED conta com uma função **GetElectron** que busca esses valor.

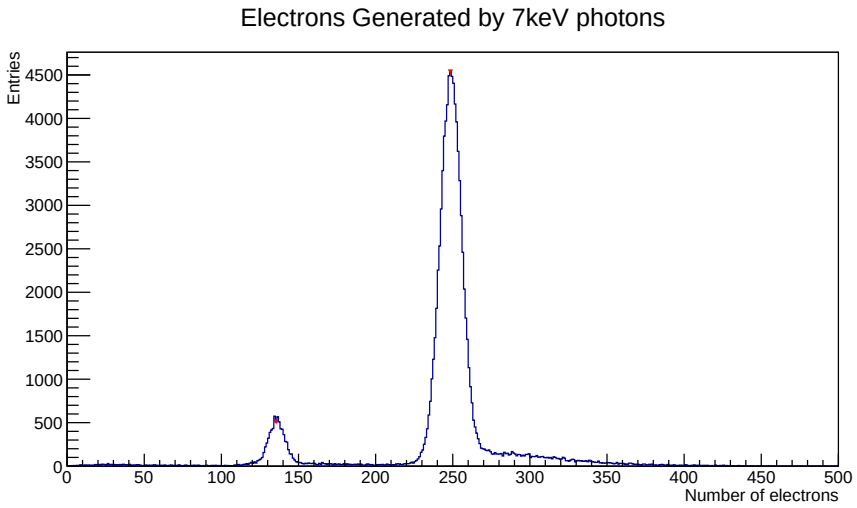



**Figura 5:** Histogramas de elétrons primários gerados por fótons com 7keV de energia. Note que no caso específico, o pico de escape já é automaticamente calculado

# Outras funções

O Garfield++ conta com mais uma grande quantidade de funções que podem ou não ser encontradas nos guias fornecidos pelos desenvolvedores. Dessa maneira, é sempre interessante visitar os arquivos fonte do Garfield++ e estudar um pouco dos códigos em cada uma das classes definidas.

Para isso vá em:

```
cd $GARFIELD_HOME/Source
```

# Referências


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

[2] H. Schindler, "Garfield++ simulation of tracking detectors — user guide." `https://garfieldpp.web.cern.ch/garfieldpp/`, 2017.2. Access in: 14-09-2018.

[3] R. Veenhof, "Garfield — simulation of gaseous detectors." `http://cern.ch/garfield`. Access in: 14-09-2018.

[4] S. F. Biagi, "Magboltz - transport of electrons in gas mixtures." `http://magboltz.web.cern.ch/magboltz/`, 1995. Access in: 05-10-2018.

[5] J. Renner, "Detector simulation in garfield++ with open-source finite element electrostatics." `http://garfieldpp.web.cern.ch/garfieldpp/examples/elmer/garfield_elmer_doc.pdf`. Access in: 16-07-2019.

[6] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[7] J. Ruokolainen, M. Malinen, P. Raback, T. Zwinger, A. Pursula, and M. Byckling, "Elmersolver manual." `http://www.nic.funet.fi/index/elmer/doc/ElmerSolverManual.pdf`, 2018. Access in: 05-10-2018.

[8] A. Rivetti, *CMOS: front-end electronics for radiation sensors*. Devices, circuits, and systems, Boca Raton, FL: CRC Press, 2015.

[9] "Paraview." `https://www.paraview.org/`. Access in: 15-07-2019.

[10] R. Brun and F. Rademakers, "ROOT - an object oriented data analysis framework," *Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. Meth. A 389 81-86*, 1997.

[11] M. J. Druyvesteyn and F. M. Penning, "The mechanism of electrical discharges in gases of low pressure," *Rev. Mod. Phys.*, vol. 12, pp. 87–174, Apr 1940.

[12] Ö. Şahin, I. Tapan, E. N. Özmutlu, and R. Veenhof, "Penning transfer in argon-based gas mixtures," *Journal of Instrumentation*, vol. 5, no. 05, p. P05002, 2010.

[13] I. Smirnov, "Modeling of ionization produced by fast charged particles in gases," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 554, no. 1, pp. 474 – 493, 2005.