Documentation: Interfacing Geant4, Garfield++ and Degrad

Dorothea Pfeiffer[a,b,*], Lennert De Keukeleere[c], Carlos Azevedo[d], Francesca Belloni[e], Stephen Biagi[f], Vladimir Grichine[g], Leendert Hayen[c], Andrei R. Hanu[h], Ivana Hřivnáčová[i], Vladimir Ivanchenko[b,j], Vladyslav Krylov[k,l], Heinrich Schindler[b], Rob Veenhof[b,m]

[a] *European Spallation Source (ESS AB),P.O. Box 176, SE-22100 Lund, Sweden*
[b] *CERN, CH-1211 Geneva 23, Switzerland*
[c] *Instituut voor Kern- en Stralingsfysica, KU Leuven, Belgium*
[d] *I3N - Physics Department, University of Aveiro, 3810-193 Aveiro, Portugal*
[e] *CEA Saclay, 91191 Gif-sur-Yvette, France*
[f] *Department of Physics, University of Liverpool, UK*
[g] *Lebedev Physical Institute of RAS, Moscow, Russia*
[h] *NASA Goddard Space Flight Center, Greenbelt, Maryland 20771, USA*
[i] *Institut de Physique Nucléaire, Université Paris-Sud, CNRS-IN2P3, Orsay, France*
[j] *Tomsk State University, Tomsk, Russia*
[k] *Laboratoire de l'Accélérateur Linéaire (LAL), Université Paris-Sud XI, CNRS/IN2P3, 91898 Orsay, France*
[l] *Taras Shevchenko National University of Kyiv (TSNUK), Kyiv, Ukraine*
[m] *Uludag University, 16059 Nülufer-Bursa, Turkey*

Abstract

This document describes the necessary software steps to interface Geant4, Garfield++ (including its interfaces to Heed and SRIM) and Degrad.

1. Software implementation

To interface Geant4 with an external software package, the physics parametrization feature in Geant4 can be used. The general proposed idea is to create a region, in which the user provides an implementation of the physics and the detector response, instead of the default Geant4 code in this region. This region, defined by the G4Region class, is created during the

*Corresponding author
Email address: dorothea.pfeiffer@cern.ch (Dorothea Pfeiffer)

detector construction, and consists of one or more G4LogicalVolumes, often corresponding to sub detector volumes. The complete syntax is shown in listing 1 of Appendix A. To implement the parametrized physics model, the user has to create a new UserG4FastSimulationModel derived from G4VFastSimulationModel, and attach it to the region. It is possible to attach more than one UserG4FastSimulationModel to the same region. The user physics code is now used instead of Geant4 in this region, whereas for the remainder of the geometry, the Geant4 physics is still valid.

In the physics list of the program, a G4FastSimulationManagerProcess has to be created. The G4FastSimulationManagerProcess is the physics process of the parametrization, and it serves as interface between the Geant4 tracking and the user parametrization. At tracking time, the G4FastSimulationManagerProcess allows the user model to be triggered. In the AddParameterisation() method of the user physics list, the G4FastSimulationManagerProcess must be added as a discrete process to the process list of the particles for which the model shall apply[1]. Listing 2 of Appendix A contains the template for the user physics list class.

The core part of the interface is the UserG4FastSimulationModel derived from G4VFastSimulationModel. The name G4VFastSimulationModel implies normally that the parametrized model is simpler and thus faster than the full Geant4 tracking. In the case of Garfield++ or Degrad, however, the parametrised model is more detailed. The G4VFastSimulationModel has three pure virtual methods, which must be overridden in the UserG4FastSimulationModel. The template of this class is shown in listing 3 of Appendix A. The first method, IsApplicable(), must return true when the parametrization model should be applied to the particle under consideration. If this is not the case, the default Geant4 physics will be applied. The second method, ModelTrigger(), is called in every step along the track and should return true if the user-defined conditions of the track are fulfilled. Finally, the implementation of the parametrised model occurs in the DoIt()-method, depending on the use case. More on the implementation of the G4FastSimulationModel can be found in the Geant4 User guide for application developers[1].

The subsequent implementation steps are quite different for Garfield++ and Degrad, and depending on whether the Geant4 parametrization feature is used. Therefore the implementations are described separately in the following

[1]As of geant4.10.4 it is also possible to implement this via a macro command

sections 1.1, 1.2 and 1.3, respectively.

1.1. Geant4 and Garfield++ without Geant4 parametrization

If just the Geant4 PAI model is used, Garfield++ has to be directly interfaced to Geant4 in the methods UserSteppingAction or SensitiveDetector. In Geant4 the PAI model can either produce all electron-ion pairs directly, or the number and positions can be sampled from the energy deposition in each step. The positions of the electron-ion pairs are subsequently sent to Garfield++, where the detector simulation is continued. If the electrons are to be produced and not just sampled, the lower production cut of the Geant4 PAI model and the lowest electron energy have to be set to the correct values to produce the correct number of electrons. Listing 9 of Appendix A illustrates how to store the last position of the electron track before its tracking is stopped. The sampling of electron-ion pairs from the energy deposition is shown in listing 10 of Appendix A. The mean number of electron-ion pairs can be determined by the MeanNumberOfIonsAlongStep() method of the G4ElectronIonPair class for each step of the primary and secondary particles. Using the mean number of electron-ion pairs together with the user specified Fano factor, a random gamma function calculates the exact number of electron-ion pairs to be produced per step. The positions of these electron-ion pairs are sampled between the pre- and post-step point of the step with the help of the G4UniformRand() distribution. It should be noted that this method is rather crude, especially for low energies.

1.2. Geant4 and Garfield++

In case of the Geant4/Garfield++ interface, the user implementation of the detector response is based on Garfield++. The G4Region or envelope corresponds here to a gas-filled detector volume. For the use cases Geant4 PAI model and Geant4/Heed PAI model interface, the PAI model has to be added as extra EM model to the user physics list (listing 2 of Appendix A) and the lower production cut has to be set to the correct value (roughly between the minimum ionization potential and the W value of the gas mixture.) Further, the lowest electron energy has to be set to an appropriate value.

Depending on the use case, the two virtual methods IsApplicable() and ModelTrigger() of the G4FastSimulationModel should be adapted according to listing 4 of Appendix A. In case the Geant4/Heed PAI model interface is used, the Garfield++ model is valid for ionization electrons produced by the

Geant4 PAI model. The model is triggered when the kinetic energy drops below a user-defined threshold value. If Heed alone is responsible for the creation of all the electron-ion pairs, the Garfield++ model is triggered as soon as the particle enters the gas volume regardless of its energy. The DoIt()-method displayed in listing 5 of Appendix A contains the actual Garfield++ model. The tracking of the relevant particle is first stopped in Geant4 (the particle is *killed*), and subsequently the particle is recreated and propagated in Garfield++. Here, Heed is used to generate the electron-ion pairs: The methods TransportDeltaElectron(), NewTrack() and TransportPhoton() are called, respectively. Since the NewTrack() method produces electrons in ionization clusters, these clusters need to be retrieved first, before extracting the individual electrons. In the other two cases the electrons can be retrieved directly. Subsequently, the electron drift and amplification can be simulated using the classes AvalancheMC and AvalancheMicroscopic in Garfield++ according to the simulation needs. Details concerning these algorithms can be found in the Garfield++ user documentation. If a Garfield++ model is used, the user-defined G4VFastSimulationModel should contain a fourth public method where related objects are initialized before the run is started. Features such as the detector geometry, the electromagnetic fields and preferred model for tracking have to be defined here, as demonstrated in listing 7 of Appendix A. The geometry is added to the field component, and subsequently a sensor is created using the field component. The sensor class links the detector geometry with its properties (material, geometry, fields) to the transport classes such as, TrackHeed. Since the shape and size of the gas volume and the gas composition is available in the Geant4 DetectorConstruction class, the information from there can be used to create the detector geometry in the Garfield++ Model.

A paradigmatic application for the Geant4/Garfield++ interface is the simulation of secondary scintillation (also referred as electroluminescence) in gaseous detectors as shown in the bottom part of listing 5 of Appendix A. The AvalancheMicroscopic class in Garfield++ is used to *microscopically* track the electron-ion pairs released after the primary particle interaction, storing a list of the Xe excited states. The SetUserHandleInelastic() method accepts a user written callback function that has access to the time, position and excitation levels in each simulation step. This information is stored in a data structure. In the case of pure noble gases, if one assumes that each excitation will produce a secondary scintillation photon, the production of light can be computed in a straightforward way. For each element of the

data structure filled in the callback function, a secondary optical photon is produced in Geant4 as shown in listing 6 of Appendix A. The optical photon tracking will then be carried out by Geant4. For gas mixtures, a more detailed model should be adopted in Garfield++ in order to include the quenching of possible excimers by the admixture.

Listing 8 of Appendix A shows how to compile and link the Geant4/Garfield++ program. A CMakeLists.txt has to be created, in which the Garfield++ include and library directories are defined. The library flags -lGarfield -lgfortran have to be placed behind the library flags for the Geant4 libraries.

1.3. Geant4 and Degrad

The implementation of the Geant4/Degrad interface is based on a user-defined G4VFastSimulationModel, analogous to the Geant4/Garfield++ interface. During detector construction, a G4Region is created (listing 1 of Appendix A). In the user physics list the AddParameterisation() function should be modified to include only the particles for which the model shall apply (listing 2 of Appendix A). For the Degrad model, it corresponds to X-rays producing electrons via the photo-electric effect or Compton scattering in Geant4. Directly after their production, the tracking of these electrons is stopped in Geant4. Degrad then re-simulates the X-ray interaction at the time and location of the stopped electrons. The parametrization in Geant4 is therefore not valid for X-rays, but for the electrons. In the IsApplicable() and ModelTrigger() methods of the G4VFastSimulationModel (listing 11 of Appendix A), the model is enabled for electrons that have an energy higher than the thermalization energy. The thermalization energy can be set in Degrad, and should be slighly lower than the minimum ionization potential of the gas mixture. Finally, the DoIt() method contains the actual Degrad model and interfacing details. All parameters needed to run the simulation (gas composition, electric and magnetic fields, particle type and kinetic energy) are stored in a parameter text file, which is read in by the Degrad binary. Prior to its call, the Degrad binary executable has to be compiled from the source code with the command "f95 degrad2.14.f -o degrad".

In the example under listing 11 of Appendix A, a 5.9 keV Fe X-ray is simulated in pure Xenon. As Degrad is a Fortran software and writes its output to a result file, the simplest way to transfer information back to Geant4 is based on a file I/O chain: Geant4 runs Degrad with a parameter file and Degrad writes the simulation results to a text file. From this result file, Geant4 reads back the positions and production times of the ionization

electrons. Degrad always assumes $(x0, y0, z0) = (0, 0, 0)$ as the interaction position of the photon, and simulates an infinite volume. Therefore, the positions of the ionization electrons have to be corrected, using the stored interaction position of the X-ray from Geant4. If the corrected positions are within the gas volume of the detector, the CreateSecondaryTrack() method of the G4FastTrack class is called to create new 7 eV (the thermalization energy set in Degrad) secondary electrons in Geant4. For a simulation of the light production, now the Garfield++ parametrization of the electroluminescence example could be used to produce optical photons.

Appendix A. Code listings

```
1 G4VPhysicalVolume* UserDetectorConstruction::
     Construct() {
2   /* Geant4 in sequential mode*/
3   G4Region* region = new G4Region("gasRegion");
4   region->AddRootLogicalVolume(fLogicalVolumeGas);
5
6   fModel = new GarfieldG4FastSimulationModel("
     UserModel", region);
7 }
```

Listing 1: G4Region implementation in the G4VUserDetectorConstruction class in Geant4 sequential mode.

```
void UserPhysicsList::AddParameterisation() {
   /* Geant4 in sequential mode:
   Create G4FastSimulationManagerProcess*/
  G4FastSimulationManagerProcess* process =
      new G4FastSimulationManagerProcess("G4FSMP");
  auto it=GetParticleIterator();
  it->reset();
  while ((*it)()) {
    G4ParticleDefinition* p = it->value();
    G4ProcessManager* manager = p->GetProcessManager
   ();
    G4EmConfigurator* config = G4LossTableManager::
   Instance()-> EmConfigurator();
    G4String name = "e-";
    if (p->GetParticleName() == name)   {
      /*Add G4FastSimulationManagerProcess*/
      manager->AddDiscreteProcess(fastSimProcess);
      /*Add PAI or PAI photon model*/
      G4PAIModel* pai = new G4PAIModel(p, "
   G4PAIModel");
      config->SetExtraEmModel(name,   "eIoni", pai,
   "gasRegion", 0.*eV, 1.*TeV, pai);
    }
  }
}

void UserPhysicsList::SetCuts() {
  /* Geant4 in sequential mode*/
  G4ProductionCutsTable::GetProductionCutsTable()->
   SetEnergyRange(cut*eV, 1.*TeV);
  G4EmParameters* emParams = G4EmParameters::
   Instance();
  emParams->SetLowestElectronEnergy(energy*eV);
}
```

Listing 2: Instantiation of the G4FastSimulationManagerProcess in a physics list in sequential Geant4. The code shows how to enable the PAI model and the Geant4/Garfield++ interface. From Geant4 10.2 on, the PAI model may be enabled via UI command instead, however this code will still work.

```
1  class UserFastSimulationModel : public
       G4VFastSimulationModel {
2
3    public:
4        /*Constructor and Destructor*/
5        UserFastSimulationModel();
6        ~UserFastSimulationModel();
7
8        /*virtual methods*/
9
10       /*return true if the G4FastSimulationModel has
       to be applied for a particle type*/
11       virtual G4bool IsApplicable(const
       G4ParticleDefinition&);
12
13       /*return true if conditions like kinetic
       energy of G4FastTrack are fulfilled*/
14       virtual G4bool ModelTrigger(const G4FastTrack
       &);
15
16       /*The parametrization, i.e. Garfield or Degrad
       related code, should be implemented here*/
17       virtual void DoIt(const G4FastTrack&,
       G4FastStep&);
18 };
```

Listing 3: Class definition of the user-defined class derived from the G4VFastSimulationModel.

```
#include "GarfieldG4FastSimulationModel.hh"

G4bool GarfieldG4FastSimulationModel::IsApplicable(
const G4ParticleDefinition& particleType) {
  /*Geant4/Heed PAI model interface*/
  return &def == G4Electron::ElectronDefinition();

  /*Heed for relativistic charged particles*/
  return &def == G4Proton::ProtonDefinition();

  /*Heed for photons*/
  return &def == G4Gamma::GammaDefinition();

  /*Electroluminescence example*/
  return &def == G4Electron::ElectronDefinition();
}

G4bool GarfieldG4FastSimulationModel::ModelTrigger(
    const G4FastTrack& fastTrack) {
  double ekin = fastTrack.GetPrimaryTrack()->
   GetKineticEnergy() / keV;

  /*Geant4/Heed PAI model interface*/
  if(ekin < 2. * keV)   {return true;}

  /*Heed*/
  return true;

  /*Electroluminescence example*/
  if(ekin < 10. * eV) {return true;}

  return false;
}
```

Listing 4: G4FastSimulationModel.

```
void GarfieldG4FastSimulationModel::DoIt(const
    G4FastTrack& fastTrack, G4FastStep& fastStep) {
  /*Get world and local position from fast track,
    get momentum vector from fast track, get global
    time in ns, and x,y,z position in cm, get kinetic
     energy of fast track in eV*/
  double ekin_eV = fastTrack.GetPrimaryTrack()->
    GetKineticEnergy() / eV;

  /*Kill track in Geant4 - Garfield takes over*/
  fastStep.KillPrimaryTrack();

  /*Number of electrons produced in a collision*/
  int nc = 0;

  Garfield::TrackHeed* trackHeed = new Garfield::
    TrackHeed();
  trackHeed->EnableDeltaElectronTransport();

  /*Geant4/Heed PAI model interface*/
  trackHeed->TransportDeltaElectron(x_cm, y_cm, z_cm
    , globaltime, eKin_eV, dx, dy, dz, nc);
  for (int cl = 0; cl < nc; cl++) {
    double xe, ye, ze, te;
    double ee, dxe, dye, dze;
    trackHeed->GetElectron(cl, xe, ye, ze, te, ee,
    dxe, dye, dze);
    /*Garfield++ simulation according to needs*/
  }

  /*Heed for relativistic charged particles*/
  trackHeed->SetParticle("proton");
  trackHeed->SetKineticEnergy(ekin_eV);
  trackHeed->NewTrack(x_cm, y_cm, z_cm, globaltime,
    dx, dy, dz);
  double xcl, ycl, zcl, tcl, ecl, extra;
  while (trackHeed->GetCluster(xcl, ycl, zcl, tcl,
    nc, ecl, extra)) {
```

```
29      for (int cl = 0; cl < nc; cl++) {
30        double xe, ye, ze, te;
31        double ee, dxe, dye, dze;
32        trackHeed->GetElectron(cl, xe, ye, ze, te, ee,
       dxe, dye, dze);
33        /*Garfield++ simulation according to needs*/
34      }
35    }
36
37    /*Heed for photons*/
38    trackHeed->TransportPhoton(x_cm, y_cm, z_cm,
       globaltime, eKin_eV, dx, dy, dz, nc);
39    for (int cl = 0; cl < nc; cl++) {
40      double xe, ye, ze, te;
41      double ee, dxe, dye, dze;
42      trackHeed->GetElectron(cl, xe, ye, ze, te, ee,
       dxe, dye, dze);
43      /*Garfield++ simulation according to needs*/
44    }
45
46    /*Electroluminescence example: Create the
       microscopic avalanche class*/
47    Garfield::AvalancheMicroscopic * aval = new
       Garfield::AvalancheMicroscopic();
48    /*Connect avalanche microscopic to sensor*/
49    aval->SetSensor(sensor);
50    /*Set the callback function*/
51    aval->SetUserHandleInelastic(
       accessExcitationLevels);
52    aval->AvalancheElectron(x0,y0,z0,t0,e0,0.,0.,0.);
53    /*Loop over data structure filled in callback
       function, produce optical photons in Geant4
54  }
55
56  /*Define a global callback function*/
57  void accessExcitationLevels(double x, double y,
       double z, double t, int type, int level,Garfield
       ::Medium * m) {
```

```
58   /*Access the excitation levels, fill data
      structure with time and position of excitations*/
59 }
```

Listing 5: G4FastSimulationModel DoIt() method for Garfield++.

```
1 void GarfieldG4FastSimulationModel::DoIt(const
    G4FastTrack& fastTrack, G4FastStep& fastStep) {
2     /*Previously described steps*/
3
4     /*Position of the Garfield++ particle*/
5     G4ThreeVector position(x, y, z);
6
7     /*Create new electron or optical photon*/
8     G4DynamicParticle particle(G4Electron::
    ElectronDefinition(), G4RandomDirection(), eKin);
9     G4DynamicParticle particle(G4OpticalPhoton::
    OpticalPhotonDefinition(),G4RandomDirection(),
    eKin);
10
11    /*Set a number that is larger than the maximum
    number of expected secondaries*/
12    fastStep.SetNumberOfSecondaryTracks(1000000);
13
14    /*Create secondary electron*/
15    fastStep.CreateSecondaryTrack(particle, position
    , time, true);
16
17    /*Create secondary optical photon with random
    polarization*/
18    fastStep.CreateSecondaryTrack(particle,
    G4RandomDirection(), position, time, true);
19 }
```

Listing 6: Production of secondary particles.

```
1  void GarfieldG4FastSimulationModel::Initialize() {
2    /*Define the gas*/
3    MediumMagboltz* gas = new MediumMagboltz();
4    gas->SetComposition("ar", percentageAr, "co2",
5    percentageCO2);
6    gas->SetTemperature(temperature);
7    gas->SetPressure(pressure);
8    gas->LoadGasFile("ar_70_co2_30_1000mbar.gas");
9    /*Set W value and Fano factor to literature or
      measured values*/
10   gas->SetW(27.3090);
11   gas->SetFanoFactor(0.1866);
12
13   /*Create the geometry*/
14   SolidBox* box = new SolidBox(0, 0., 0., dx, dy, dz
      );
15   GeometrySimple* geo = new GeometrySimple();
16   geo->AddSolid(box, gas);
17
18   /*Make a component*/
19   ComponentConstant* comp = new ComponentConstant();
20   comp->SetGeometry(geo);
21   comp->SetElectricField(-1000., 0., 0.);
22
23   /*Make a sensor*/
24   Sensor* sensor = new Sensor();
25   sensor->AddComponent(comp);
26
27   /*Create the track class*/
28   TrackHeed* trackHeed = new TrackHeed();
29   trackHeed->SetSensor(sensor);
30   trackHeed->EnableDeltaElectronTransport();
31
32   /*Create the drift algorithm*/
33   AvalancheMC * drift = new Garfield::AvalancheMC();
34   drift->EnableDiffusion();
35   drift->DisableMagneticField();
36   drift->SetSensor(sensor);
```

```
37
38   /*Create the avalanche algorithm*/
39   AvalancheMicroscopic * avalanche = new
      AvalancheMicroscopic();
40   avalanche->SetSensor(sensor);
41 }
```

Listing 7: Setup of Garfield++ simulation.

```
1 include(${Geant4_USE_FILE})
2 include_directories(${PROJECT_SOURCE_DIR}/include)
3 include_directories($ENV{GARFIELD_HOME}/Include)
4
5 link_directories($ENV{GARFIELD_HOME}/Library)
6 target_link_libraries(IonisationChamber ${
     Geant4_LIBRARIES} -lGarfield -lgfortran )
```

Listing 8: CMakeList.txt.

```
1 void GarfieldSeppingAction::UserSteppingAction(const
      G4Step* theStep) {
2   G4Track* track = theStep->GetTrack();
3   G4double time = track->GetGlobalTime();
4   G4ThreeVector wp = theStep->GetPostStepPoint()->
     GetPosition();
5   G4ThreeVector p = theTouchable->GetHistory()->
     GetTopTransform().TransformPoint(wp);
6   G4String name = track->GetDefinition()->
     GetParticleName();
7
8   if(name == "e-" && track->GetStatus() ==
     fStopAndKill) {
9     /*Send position and time to Garfied++ to
     continue simulation*/
10    SendElectronsToGarfield(p.getX(), p.getY(), p.
     getZ(), time);
11  }
12 }
```

Listing 9: Storing of last positions of electron tracks in UserSteppingAction to generate electron-ion pairs for Garfield++.

```
1  void GarfieldSeppingAction::UserSteppingAction(const
       G4Step* theStep) {
2    double invFanoFactor = 1/0.19;
3    G4double meanIon = G4LossTableManager::Instance()
     ->ElectronIonPair()->MeanNumberOfIonsAlongStep(
     theStep);
4    /*Determine number of electron-ion pairs with the
     help of W and Fano*/
5    G4int nIon =  G4lrint(G4RandGamma::shoot(meanion*
     invFanoFactor,invFanoFactor));
6
7    /*Sample ionisation along step*/
8    if(nIon > 0) {
9      G4Track* fTrack = theStep->GetTrack();
10     G4double t = fTrack->GetGlobalTime();
11     G4ThreeVector p = theStep->GetPreStepPoint()->
     GetPosition();
12     G4ThreeVector dp = theStep->GetPostStepPoint()->
     GetPosition() - p;
13     for(G4int i=0; i<nIon; ++i) {
14       G4ThreeVector nposw = p + dp*G4UniformRand();
15       G4ThreeVector np = theTouchable->GetHistory()
     ->GetTopTransform().TransformPoint(nposw);
16
17       /*Send position and time to Garfied++ to
     continue simulation*/
18       SendElectronsToGarfield(np.getX(), np.getY(),
     np.getZ(), t);
19     }
20   }
21 }
```

Listing 10: Use of MeanNumberOfIonsAlongStep() in UserSteppingAction to generate electron-ion pairs for Garfield++.

```
1  void DegradG4FastSimulationModel::DoIt(const
       G4FastTrack& fastTrack, G4FastStep& fastStep) {
2    /*Start by killing the primary track*/
3    fastStep.KillPrimaryTrack();
4
5    /*Store the interaction position and time*/
6    G4ThreeVector p = fastTrack.GetPrimaryTrack()->
       GetVertexPosition();
7    G4double t = fastTrack.GetPrimaryTrack()->
       GetGlobalTime();
8
9    /*Generate a random seed to Degrad*/
10   G4int n = 54217137*G4UniformRand();
11   G4String seed = G4UIcommand::ConvertToString(n);
12
13   /* Write the Degrad input cards. Example for
       photon of 5.9 keV in pure Xenon (thermalization
       energy of 7.0 eV), in a 3000.0 V/cm electric
       field anti-parallel to the photon direction. The
       gas was considered to be at 20 degrees C and 760
       Torr. For details see Degrad source file */
14   G4String degradCards="printf \"1,1,3,-1,"+seed+",
15     5900.0,7.0,0.0\n
16     7,0,0,0,0,0\n
17     100.0,0.0,0.0,0.0,0.0,0.0,20.0,760.0\n
18     3000.0,0.0,0.0,1,0\n
19     100.0,0.5,1,1,1,1,1,1,1\n
20     0,0,0,0,0,0\" > conditions.txt";
21   G4int stdout=system(degradCards.data());
22
23   /*Call Degrad with the input conditions file*/
24   stdout=system("./degrad < conditions.txt");
25
26   /*Here the user should write a simple code in C++
       just to read the Degrad ASCII file and get the
       electrons position and thermalization time.
       Remember that Degrad makes all interactions in (
       x0,y0,z0)=(0,0,0) and t0=0: the returned
```

```
    positions should be shifted according to
    interaction position and interaction time.
    Moreover, the Y and Z axes are swapped in Degrad
    relatively to Geant4. Distance units in Degrad
    are um and time is ps, whereas Geant4 uses mm and
     ns.*/
27   G4ThreeVector ep;
28   for (int electron=0;electron<
    NumberOfElectronsProduced; electron++) {
29     ep.setX(posXDegrad*0.001+ip.getX());
30     ep.setY(posZDegrad*0.001+ip.getY());
31     ep.setZ(posYDegrad*0.001+ip.getZ());
32     time=timeDegrad*0.001+t;
33
34     /*Check if the electron position is inside the
    gas volume. Degrad does not use geometrical
    constraints*/
35     G4Navigator* theNavigator =
    G4TransportationManager::GetTransportationManager
    ()->GetNavigatorForTracking();
36     G4VPhysicalVolume* myVolume = theNavigator->
    LocateGlobalPointAndSetup(myPoint);
37     G4String name=myVolume->GetName();
38     /*gasName should be the name of the gas volume*/
39     if (name.contains("gasName")) {
40       G4DynamicParticle electron(G4Electron::
    ElectronDefinition(),G4RandomDirection(),
    thermalizationEnergy*eV);
41       /*Create secondary electron*/
42       G4Track *newTrack=fastStep.
43       CreateSecondaryTrack(electron, ep,time,false);
44       newTrack->SetTrackID(fastStep.
    GetNumberOfSecondaryTracks());
45     }
46   }
47 }
```

Listing 11: DoIt() method for Degrad.

[1] Geant4 Collaboration, Book For Application Developers.
URL http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/fo/BookForAppliDev.pdf