stim_pixe_tomography

Geant4 advanced example for tomography imaging

User's Guide

Please cite the following paper:

Michelet C, Li Z, Jalenques H, Incerti S, Barberet P, Devès G, et al. A Geant4 simulation of X-ray emission for three-dimensional proton imaging of microscopic samples. Phys Med. 2022;94:85-93. https://doi.org/10.1016/j.ejmp.2021.12.002.

Table of contents

1. Getting Started

The ***stim_pixe_tomography*** example is based on the Monte Carlo simulation code ***Geant4***. It has been developed at Laboratoire de Physique des 2 Infinis Bordeaux *(LP2I - CENBG)*. This user guide is dedicated to:

- introduce some basic knowledge of Geant4 to beginners
- guide the user to model a 3D tomography experiment: STIM-T or PIXE-T using the ***stim_pixe_tomography*** example.

1.1. Introduction to STIM and PIXE tomography

Proton microbeams of a few MeV are widely used for the imaging and quantitative analysis of microscopic samples of a few ten or hundred micrometers in size, with a wide field of applications. Scanning Transmission Ion Microscopy tomography (STIM-T) and Particle-Induced X-ray Emission tomography (PIXE-T) are techniques to determine the three-dimensional content of microscopic samples [1]. STIM-T aims to determine the density of analyzed sample, PIXE-T to reveal the chemical content (in g/cm^3). In the experiment, the incident proton beam scans over the area of interest at a certain angle/projection, and then the sample is rotated to perform the scan for the next projection. The difference between STIM-T and PIXE-T consists of three aspects:

- STIM-T aims to detect transmitted protons, while PIXE-T deals with emitted X-rays
- The detector for STIM-T is always placed in front of the proton source (at 0°), which is not the case for PIXE-T. For example, the detector can be placed at 135° relative to the direction of incident protons) (Figure 1b).
- The number of protons required for STIM-T is small, usually 20 or 100 protons at each position of beam [2]. For PIXE-T, a higher number of protons (about 10^8 to 10^9) is needed in experiments. While in simulation, 10^6 protons can be used for each position of the beam [3].

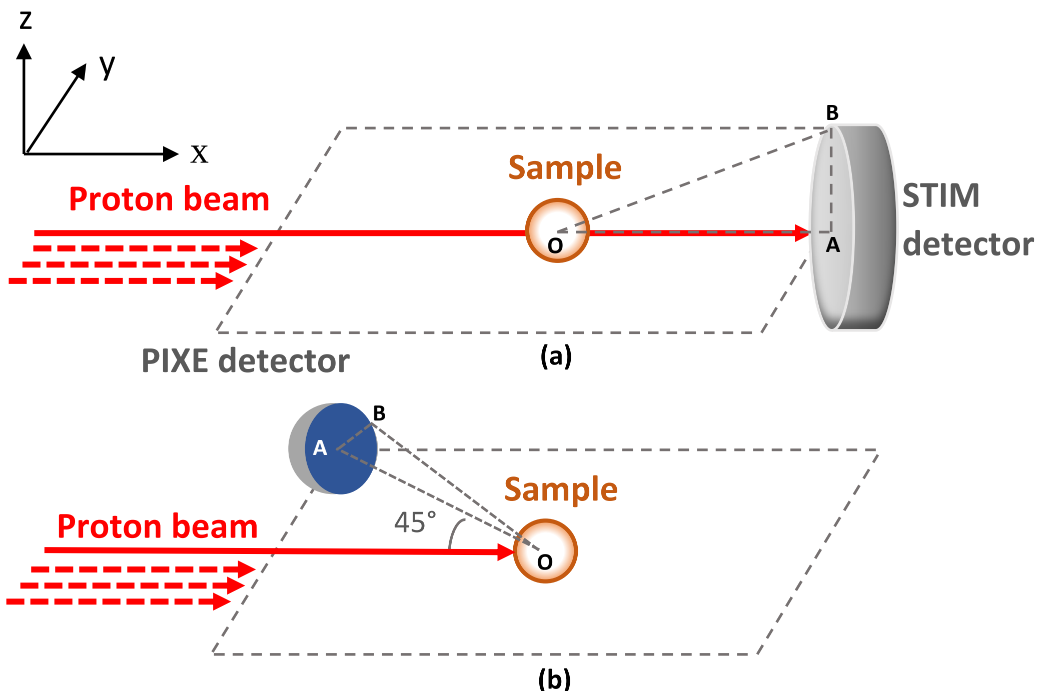


Figure 1. Experimental set-up for STIM-T (a) and PIXE-T (b).

1.2. General introduction to *stim_pixe_tomography* example

The ***stim_pixe_tomography*** advanced example is an open source application, provided as an advanced example in the ***GEANT4*** toolkit. It is based on the ***TestEm5*** example, dedicated to study the interaction of particles going through a simple material. The ***stim_pixe_tomography*** is developed for 3D imaging purpose, more precisely for STIM-T and PIXE-T simulation. STIM-T and PIXE-T simulation will generate data file(s) containing all the information, *i.e.* the energy and direction (momentum) of interested particles. The particles (protons or X-rays) are collected in 4π solid angle, and will be sorted by the user after the simulation in order to model a specific detection setup. The sorted events are ultimately used for tomographic reconstruction. At LP2IB (CENBG), a data reduction software package suited to STIM-T and PIXE-T is developed, ***TomoRebuild*** [4]. The simulated data generated by the ***stim_pixe_tomography*** code are processed in the same way as "real" experimental data, in order to obtain reconstructed images. The final reconstructed image for STIM-T is the distribution of mass density (expressed in g/cm^3). For PIXE-T, it is the distribution of the mass density of each considered chemical element.

Note that ***stim_pixe_tomography*** can be also used for other applications, such as classical (2D) imaging. Moreover, the type of incident particle can be changed, like X-rays or ion beam. It will be discussed in section 8.

1.3. Installation of *stim_pixe_tomography* example

To use the ***stim_pixe_tomography*** example, the user has to install Geant4 first. The Geant4 code is available from Geant4 website: https://geant4.web.cern.ch/. A complete virtual machine is provided by LP2I Bordeax (CENBG): https://geant4.cenbg.in2p3.fr/.

Once ***Geant4*** is installed, the ***stim_pixe_tomography*** example will be found under: ***/your-G4-install/share/your-G4-version/examples/advanced/stim_pixe_tomography***. The compilation of the ***stim_pixe_tomography*** example requires Geant4 libraries and headers. For this reason, the user needs to configure it by the following commands, in order to be able to use Geant4 tools (from the directory where Geant4 is installed). We assume that Geant4 is installed under ***/your-G4-install***, and ***stim_pixe_tomography*** example under ***/home/you/stim_pixe_tomography***

```
cd stim_pixe_tomography        // enter in stim_pixe_tomography example
mkdir build                    // create a build directory in stim_pixe_tomography example
cd build                       // enter in build directory
cmake -DGeant4_DIR=/your-G4-install/lib64/your-G4-version /home/you/stim_pixe_tomography
                               // run CMake to generate the Makefile needed to build the application
make                           // build the executable
./stim_pixe_tomography -s pixe3d.mac    // run a simulation for STIM-T, the argument “-s” is
                                        //necessary, otherwise an error will occur
```

Similarly, for PIXE-T, run as follows:

```
./stim_pixe_tomography -p pixe3d.mac    // run a simulation for PIXE-T, the argument “-p” is
                                        //necessary, otherwise an error will occur.
// After correct configuration, the following lines will appear on the screen:
-- Configuring done
-- Generating done
-- Build files have been written to: …/stim_pixe_tomography/build
```

Note that if you use the virtual machine provided by LP2I (CENBG), it is recommended to create an alias ***cmk*** for replacing the CMake command above. The alias ***cmk*** is created in ***.ucshrc*** file by following command:

```
alias cmk "cmake -DGeant4_DIR=$G4COMP .."
```

So, from the ***build*** directory, to generate the makefile, you just need to type ***cmk*** before ***make***.

2. Execution of a simulation

2.1. Description of a simulation

2.1.1. Positions of the source at first projection

First of all, the execution of ***stim_pixe_tomography*** example for STIM-T or PIXE-T is composed of a series of runs. A run corresponds to the simulation in which the proton beam is place at a certain position. Figure 2 shows the layout of a simulation of STIM-T or PIXE-Tat the first projection (*i.e.* projection index = 0, projection angle = 0° relative to the source direction). The object is here represented by a sphere in the middle. The scan is shown as the green cube surrounding the sample. The red points represent the positions of the source when scanning the sample, which are indexed relative to the YZ coordinate system. At the first projection, the source is directed along the positive x-axis. The tomographic slices are horizontal (in x-y plane). The Y index indicates the position of the source within the horizontal slice. The Z index indicates the position of the slice vertically.

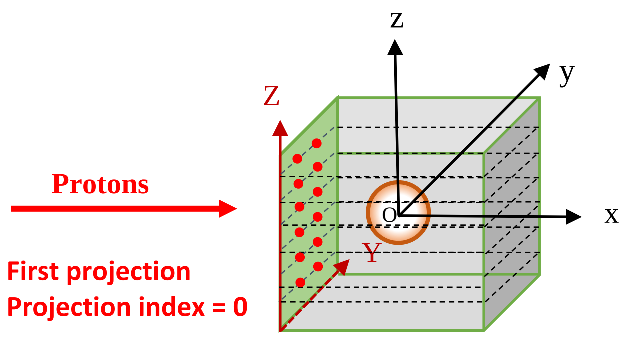


Figure 2. Layout of a simulation of STIM-T or PIXE-T at first projection (projection index = 0).

Figure 3 shows an example of the positions of the source. This example considers: number of pixels = 10 (horizontal scan), number of slices = 10 (vertical scan). The scan starts from the position indices (Y=0, Z=0). The beam source horizontally moves from Y = 0 to 9. Then it switches to the next slice, until the last position (Y=9, Z=9). The number of pixels in the Y direction defines the final number voxels in the reconstructed tomographic slice. Here it is 10 × 10 voxels for each reconstructed slice.

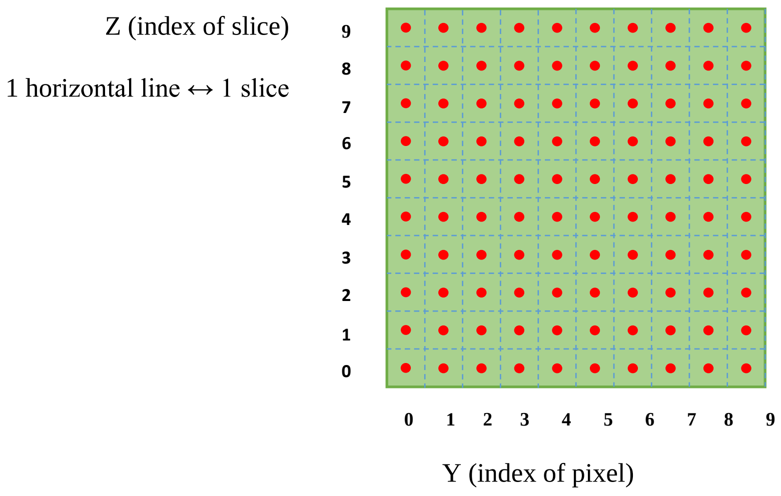


Figure 3. Positions of the beam at a given projection for a scan composed of 10 slices of 10 pixels.

2.1.2. Rotation of the source

When the source finishes the scan at the first projection, the source rotates through a certain angle (step angle) counterclockwise as Figure 4 shows. The scan described above is then repeated at this projection. Then the source rotates again and the scan is performed again, and so on until the last projection.

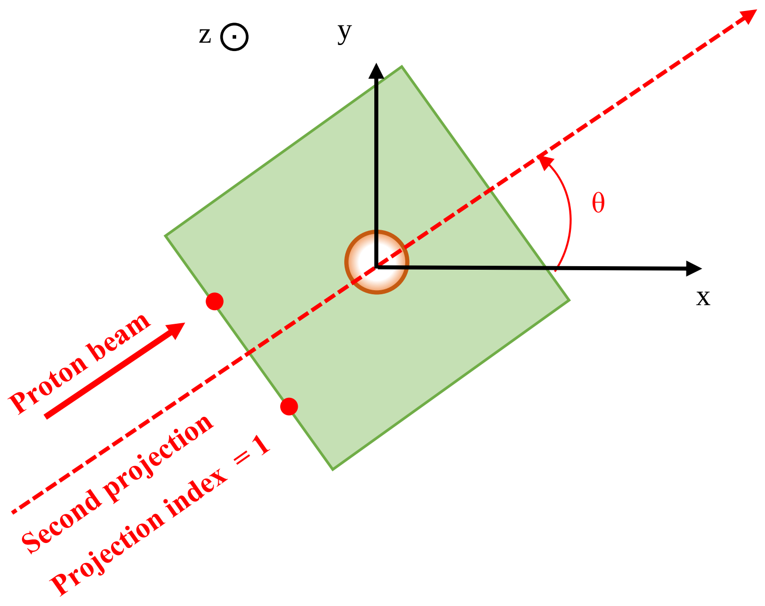


Figure 4. Rotation of the source at projection index = 1.

2.1.3. Output file(s) of a STIM-T or PIXE-T simulation

At the end of a simulation, output file(s) containing information of particles of interest are generated.

For STIM-T, a file is generated containing the residual energy and momentum of transmitted protons getting out of the object.

For PIXE-T, twos files are generated:

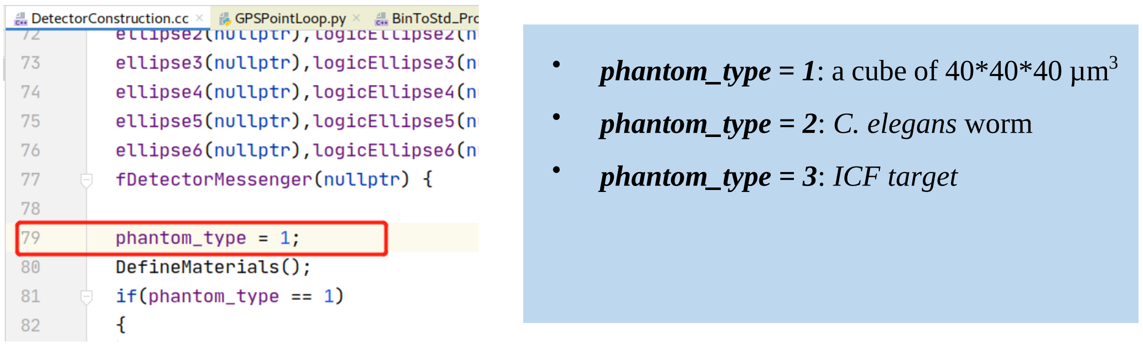
- One contains the energy and momentum of emitted X-rays (secondary) ***at creation***, *i.e.* at the point when they are generated.
- The other contains the energy and momentum of emitted X-rays (secondary) ***at exit***, *i.e.* when getting out of the object.

More details about the output file(s) can be found in sections 2.5, 2.6 and 2.10.

2.2. Construction of phantoms

According to the basic rules of Geant4, the construction of objects is implemented in the ***DetectorConstruction.cc/.hh*** codes, particularly, in the ***DetectorConstruction::Construct()*** method. In the example, three built-in phantoms are available. Users can choose the phantom by specifying the value of ***phantom_type*** (Figure 5):

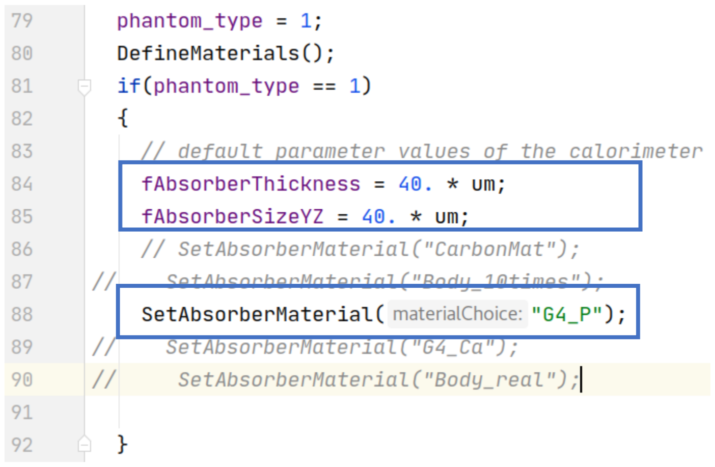
- ***phantom_type* = *1***, a cube of 40*40*40 µm^3 is constructed by ***DetectorConstruction::Construct_Phantom1()***. The cube is of uniform density and composition. In the present example, the material is the same as the "Body" part of *C. elegans* phantom. Obviously, the material composition and density, as well as the size of the cube, can be easily modified by the user (Figure 6). Regarding materials, necessary materials that are used in the phantoms are defined in the ***DetectorConstruction::DefineMaterials()*** method.
- ***phantom_type*** = 2, the upper part of *C. elegans* worm (Figure 7) is constructed by ***DetectorConstruction::Construct_Phantom2().*** The shape and size are derived from experimental data of PIXE-T and STIM-T performed at LP2I (CENBG) [2].
- ***phantom_type* = *3***, a inertial confinement fusion (ICF) target phantom is constructed by ***DetectorConstruction::Construct_Phantom3()***, the shape and size are derived from experimental data of PIXE-T and STIM-T performed at Fudan university [5, 6].



```
ellipse2(nullptr),logicEllipse2(n
ellipse3(nullptr),logicEllipse3(n
ellipse4(nullptr),logicEllipse4(n
ellipse5(nullptr),logicEllipse5(n
ellipse6(nullptr),logicEllipse6(n
fDetectorMessenger(nullptr) {

phantom_type = 1;
DefineMaterials();
if(phantom_type == 1)
{
```

- ***phantom_type* = *1***: a cube of 40*40*40 µm^3
- ***phantom_type* = *2***: *C. elegans* worm
- ***phantom_type* = *3***: *ICF target*

Figure 5. Choice of phantom type in ***DetectorConstruction.cc***



```
phantom_type = 1;
DefineMaterials();
if(phantom_type == 1)
{
  // default parameter values of the calorimeter
  fAbsorberThickness = 40. * um;
  fAbsorberSizeYZ = 40. * um;
  // SetAbsorberMaterial("CarbonMat");
//    SetAbsorberMaterial("Body_10times");
  SetAbsorberMaterial( materialChoice: "G4_P");
//    SetAbsorberMaterial("G4_Ca");
//     SetAbsorberMaterial("Body_real");

}
```

Figure 6. Setting the size and material of the built-in cube phantom

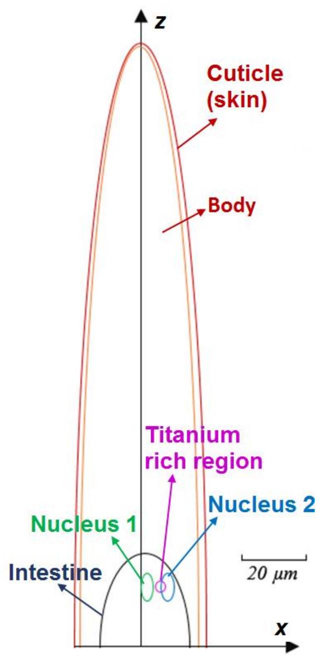
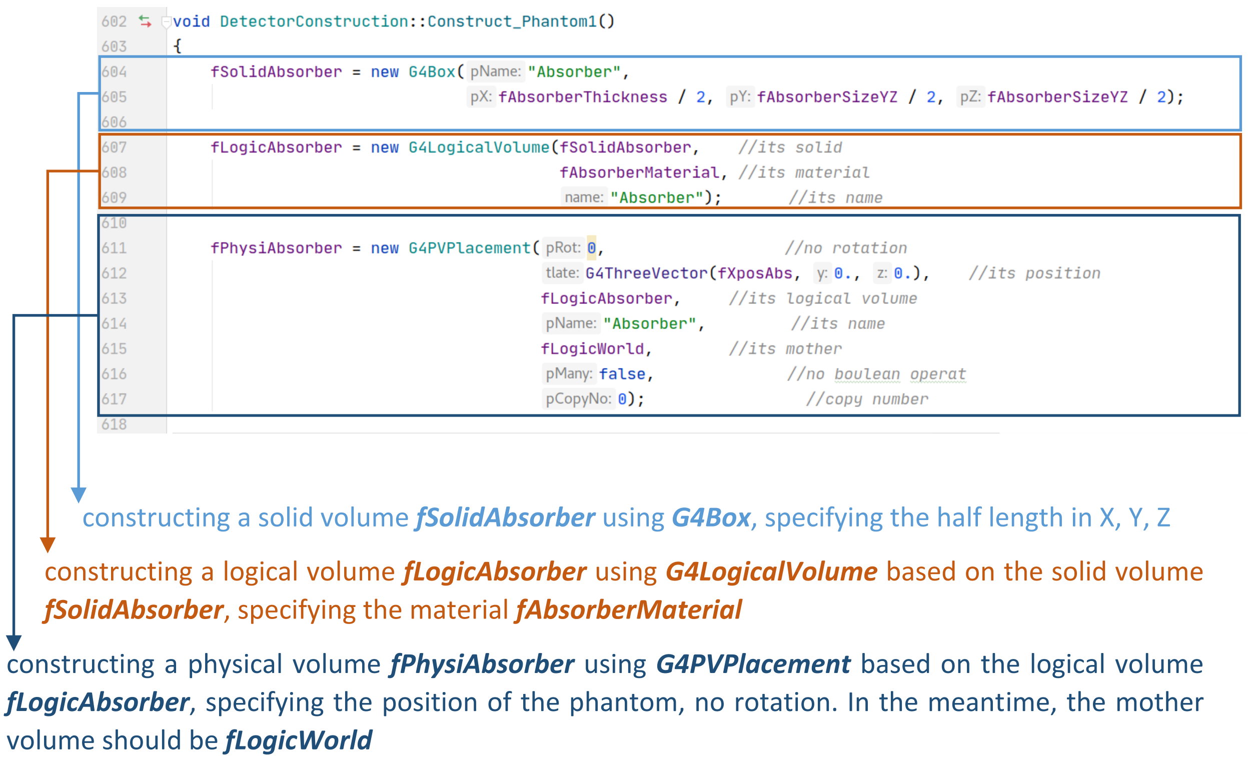


Figure 7. x-z cross sectional view of the upper part of C. elegans

Of course, the user can construct a new phantom according to the needs. In general, an object is made of three volumes (Figure 8, build-in cube for instance):

- Solid volume, specifying the shape and the size of the phantom
- Logical volume, specifying the composition (materials) of the solid volume you built
- Physical volume, specifying position and rotation of the logical volume

There is a ground rule in Geant4 in terms of building phantoms that all the phantoms should be contained in an object called "World". Thus, the user has to build a "World" object in addition to the phantom. In the ***stim_pixe_tomography*** example, the "World" is a cube, whose size is adjusted according to the size of the phantom that the user builds. Indeed, it is important to check that "World" is geometrically bigger than the phantom.



```
void DetectorConstruction::Construct_Phantom1()
{
    fSolidAbsorber = new G4Box( pName: "Absorber",
                                    pX: fAbsorberThickness / 2,  pY: fAbsorberSizeYZ / 2,  pZ: fAbsorberSizeYZ / 2);

    fLogicAbsorber = new G4LogicalVolume(fSolidAbsorber,    //its solid
                                         fAbsorberMaterial, //its material
                                          name: "Absorber");       //its name

    fPhysiAbsorber = new G4PVPlacement( pRot: 0,                  //no rotation
                                        tlate: G4ThreeVector(fXposAbs,  y: 0.,  z: 0.),    //its position
                                        fLogicAbsorber,      //its logical volume
                                         pName: "Absorber",          //its name
                                        fLogicWorld,         //its mother
                                         pMany: false,               //no boulean operat
                                         pCopyNo: 0);                  //copy number
```

constructing a solid volume ***fSolidAbsorber*** using ***G4Box***, specifying the half length in X, Y, Z

constructing a logical volume ***fLogicAbsorber*** using ***G4LogicalVolume*** based on the solid volume ***fSolidAbsorber***, specifying the material ***fAbsorberMaterial***

constructing a physical volume ***fPhysiAbsorber*** using ***G4PVPlacement*** based on the logical volume ***fLogicAbsorber***, specifying the position of the phantom, no rotation. In the meantime, the mother volume should be ***fLogicWorld***

Figure 8. Definition of Solid, Logical and Physical volumes for the built-in cube phantom.

2.3. Configuration of beam scan using GPSPointLoop.C

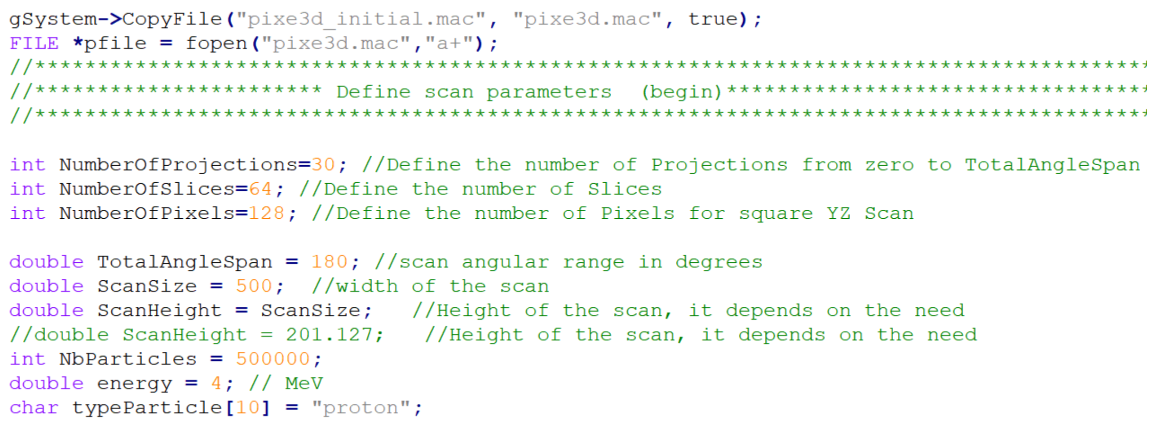
The configuration of the beam is performed by using the ***GPSPointLoop.C*** script. It reads a predefined macro ***pixe3d_initial.mac***, which contains the information of physics processes.

GPSPointLoop.C reads ***pixe3d_initial.mac*** to generate ***pixe3d.mac***, which contains all the parameters for the simulation and will be read by Geant4 to run the simulation. In ***GPSPointLoop.C***, the user defines the following variables (Figure 9), which will automatically generate the successive positions and directions of the beam written in ***pixe3d.mac***:

- ***NumberOfProjections***: number of projections, for example 100
- ***NumberOfPixels***: number of pixels (in horizontal scan), for example 128
- ***NumberOfSlices***: number of slices, for example 1 for PIXE-T and 128 for STIM-T
- ***TotalAngleSpan***: total angle of the beam scan in degree, for example if ***TotalAngleSpan*** = 180 and ***NumberOfProjections*** = 100, in this case, the first projection is at 0°, the last is at 178.2°.
- ***ScanSize***: maximal length of scan horizontally; this value should be big enough to ensure that the phantom can be completely scanned at any angles/projections. Ideally, we take 1.8*maximal width of the phantom.
  The pixel width is equal to ***ScanSize/NumberOfPixels***
- ***ScanHeight***: maximal length of scan vertically.

The pixel height is equal to ***ScanHeight/NumberOfSlices***

- ***NbParticles***: number of protons of the beam. The same number is taken for each position of the source beam.
- ***energy:*** energy of the proton beam
- ***typeParticle:*** type of the particle, for example "proton"



```
gSystem->CopyFile("pixe3d_initial.mac", "pixe3d.mac", true);
FILE *pfile = fopen("pixe3d.mac","a+");
//*****************************************************************************************
//*********************** Define scan parameters  (begin)**********************************
//*****************************************************************************************

int NumberOfProjections=30; //Define the number of Projections from zero to TotalAngleSpan
int NumberOfSlices=64; //Define the number of Slices
int NumberOfPixels=128; //Define the number of Pixels for square YZ Scan

double TotalAngleSpan = 180; //scan angular range in degrees
double ScanSize = 500;  //width of the scan
double ScanHeight = ScanSize;   //Height of the scan, it depends on the need
//double ScanHeight = 201.127;   //Height of the scan, it depends on the need
int NbParticles = 500000;
double energy = 4; // MeV
char typeParticle[10] = "proton";
```

Figure 9. Parameters to define in GPSPointLoop.C

Once the configuration is done, the user runs the following command to generate the ***pixe3d.mac***:

```
root GPSPointLoop.C
```

When it comes to the exact position of the beam, one thing should be pointed out. Let's assume that ***NumberOfPixels* = *4*** in Figure 10, so the scanned area is evenly divided into 4 pixels. The position of the beam is at the center of the pixel.

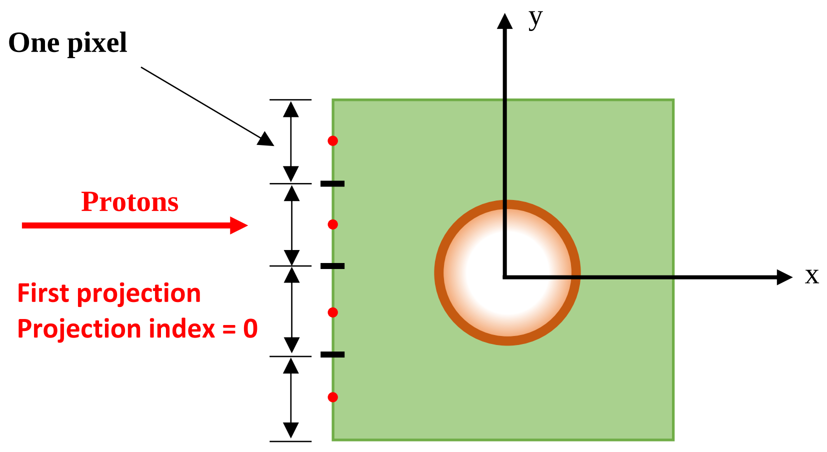


Figure 10. Position of the beam for a given horizontal slice xy.

2.4. Setting production cut

The generation of secondary particles follows a general principle in Geant4: a secondary particle is generated only if its energy is higher than the production cut. The production is set as range cut, which is converted to energy cut internally for secondary gamma, electron, positron and proton production.

2.4.1. Range cut

A range cut value is set by default to 1 mm for Livermore model; this value can be specified by the following command, for example 1 nm for microscopic objects:

```
/run/setCut 1 nm
```

We should note that a range cut can be set for a given particle type, if the user would like (we have not done this in ***pixe3d.mac***). Here is an example of setting production cut for gamma if one would like to do it:

```
/run/setCutForAGivenParticle gamma 0.5 um
```

2.4.2. Lowest energy cut

It should be noted that a lowest cut value in energy is specified by default in Geant4. The default lowest cut value E_{low} is 990 eV. When the user sets a range cut, if the converted energy cut E_{cut} corresponding to the range cut is lower than E_{low} 990 eV, this cut E_{cut} will not be valid, because Geant4 takes the highest of these two energy values. The following command is used to change the lowest energy cut E_{low}, for example to 900 eV.

```
/cuts/setLowEdge 900 eV
```

2.4.3. Final set of the cut

For PIXE applications, the chemical elements of interest are generally Na and beyond ($Z \geq 11$) because low energy X-rays would not be able to go through the entrance window of conventional detectors. So, the X-rays that we are interested in have an energy ≥ 1 keV. For this reason, we chose 900 eV as the minimal energy of X-rays that will be generated. Then, secondary X-rays less than 900 eV are not generated, that can save a tremendous amount of time.

Since the default lowest energy cut $E_{low} = 990$ eV is bigger than 900 eV, thus we have to decrease E_{low} by: ***/cuts/setLowEdge 900 eV***

In this case, the range cut **must** be carefully specified by ***/run/setCut*** to make sure that the converted energy cut E_{cut} is not bigger than $E_{low} = 900$ eV.

- If $E_{cut} > E_{low}$, Geant4 will take E_{cut} as energy cut
- If $E_{cut} < E_{low}$, Geant4 will take E_{low} as energy cut

- If the ***/run/setCut*** is not specified, Geant4 will take the default value 1 mm, and the energy cut E_{cut} risks being bigger than 900 eV depending on the material of the sample

Thus in the ***pixe3d.mac***, the following two commands are necessary:

```
/run/setCut 1 nm

/cuts/setLowEdge 900 eV
```

2.4.4. UI commands related to cut

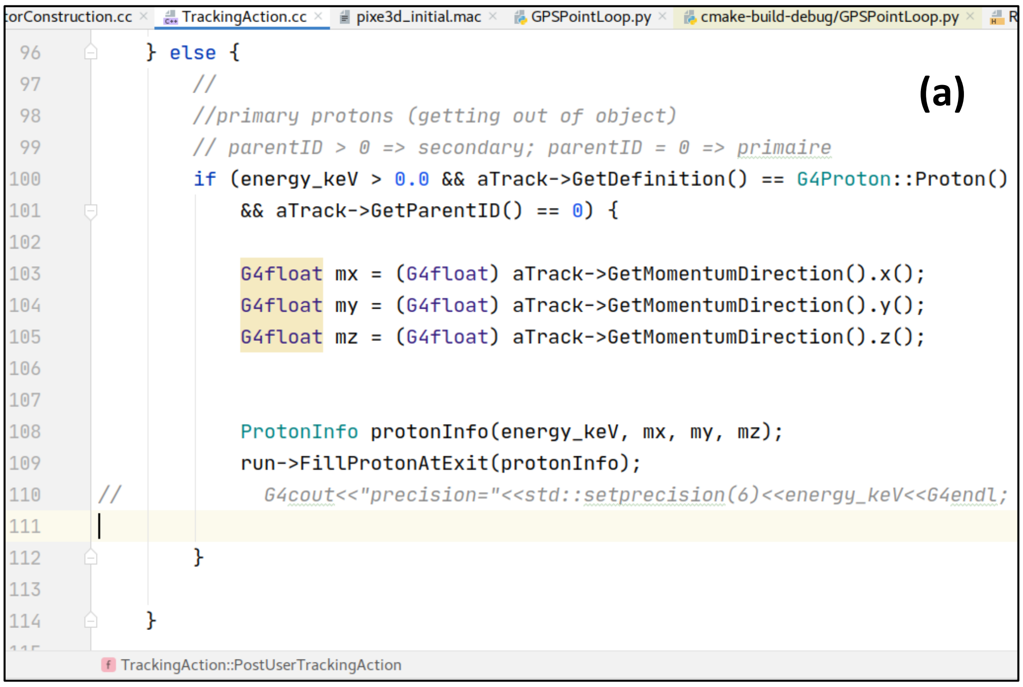
Here we give the indication for some UI commands.

- ***/process/em/deexcitationIgnoreCut***: Enable/Disable the usage of production threshold for fluorescence and Auger electron production. By default, it is false, which means the production cut is valid. This command is defined in ***G4EmLowEParametersMessenger***.
- ***/process/em/pixe***: Enable/disable PIXE along step deexcitation. By default, it is false, we should specify true in the simulation. This command is defined in ***G4EmLowEParametersMessenger***.
- ***/process/em/fluo***: Enable/disable atomic deexcitation. This command is defined in ***G4EmLowEParametersMessenger.***
- ***/process/em/auger***: Enable/disable Auger electrons production. By default, it is false, we should specify true in the simulation. This command is defined in ***G4EmLowEParametersMessenger***.
- ***/process/em/augerCascade***: Enable/disable simulation of cascade of Auger electrons. By default, it is false, we should specify true in the simulation. This command is defined in ***G4EmLowEParametersMessenger***.
- ***/process/em/fluoBearden***: Enable/disable usage of Bearden fluorescence files when modeling PIXE simulation. By default, it is false, we should specify true in the simulation. This command is defined in ***G4EmLowEParametersMessenger***.
- ***/process/em/pixeXSmodel***: Set the name of PIXE cross section files used for modeling PIXE simulation. By default, "Empirical" data are used. "ECPSSR_Analytical" and "ECPSSR_FormFactor" are available.
- ***/process/em/applyCuts:*** Enable/disable applying cuts for discrete processes, like photoelectric process. By default, it is false. This command is defined in ***G4EmParametersMessenger***.

2.5. Detection of protons for STIM-T

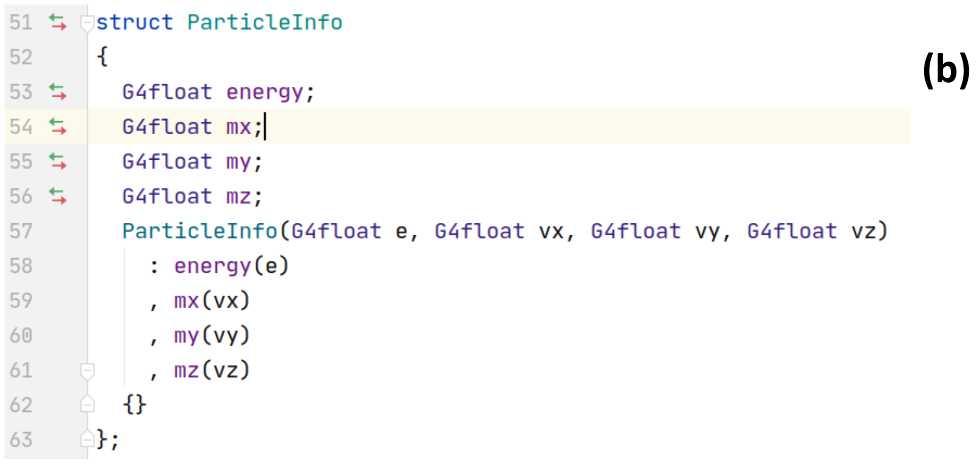
For STIM-T, we collect the energy and momentum of the transmitted protons, after they have gone through the phantom (at exit). The information is saved in one output file. The info is collected in the ***TrackingAction::PostUserTrackingAction()*** method (Figure 11a). A structure type ***ParticleInfo*** is used to save the energy and momentum of every transmitted proton. This structure is defined in ***Run.hh***

(Figure 11b). Attention, the definition of ***ParticleInfo*** in Figure 11b is used by default. It can be defined differently, if the position of the particle is considered additionally, as shown in Figure 22.



orConstruction.cc
TrackingAction.cc
pixe3d_initial.mac
GPSPointLoop.py
cmake-build-debug/GPSPointLoop.py
(a)
TrackingAction::PostUserTrackingAction

```
    } else {
        //
        //primary protons (getting out of object)
        // parentID > 0 => secondary; parentID = 0 => primaire
        if (energy_keV > 0.0 && aTrack->GetDefinition() == G4Proton::Proton()
            && aTrack->GetParentID() == 0) {

            G4float mx = (G4float) aTrack->GetMomentumDirection().x();
            G4float my = (G4float) aTrack->GetMomentumDirection().y();
            G4float mz = (G4float) aTrack->GetMomentumDirection().z();


            ProtonInfo protonInfo(energy_keV, mx, my, mz);
            run->FillProtonAtExit(protonInfo);
//              G4cout<<"precision="<<std::setprecision(6)<<energy_keV<<G4endl;

        }

    }
```



```
struct ParticleInfo
{
  G4float energy;
  G4float mx;
  G4float my;
  G4float mz;
  ParticleInfo(G4float e, G4float vx, G4float vy, G4float vz)
    : energy(e)
    , mx(vx)
    , my(vy)
    , mz(vz)
  {}
};
```

Figure 11. Code to collect the transmitted protons for STIM-T (a).
*Definition of the **ParticleInfo** struct (b).*

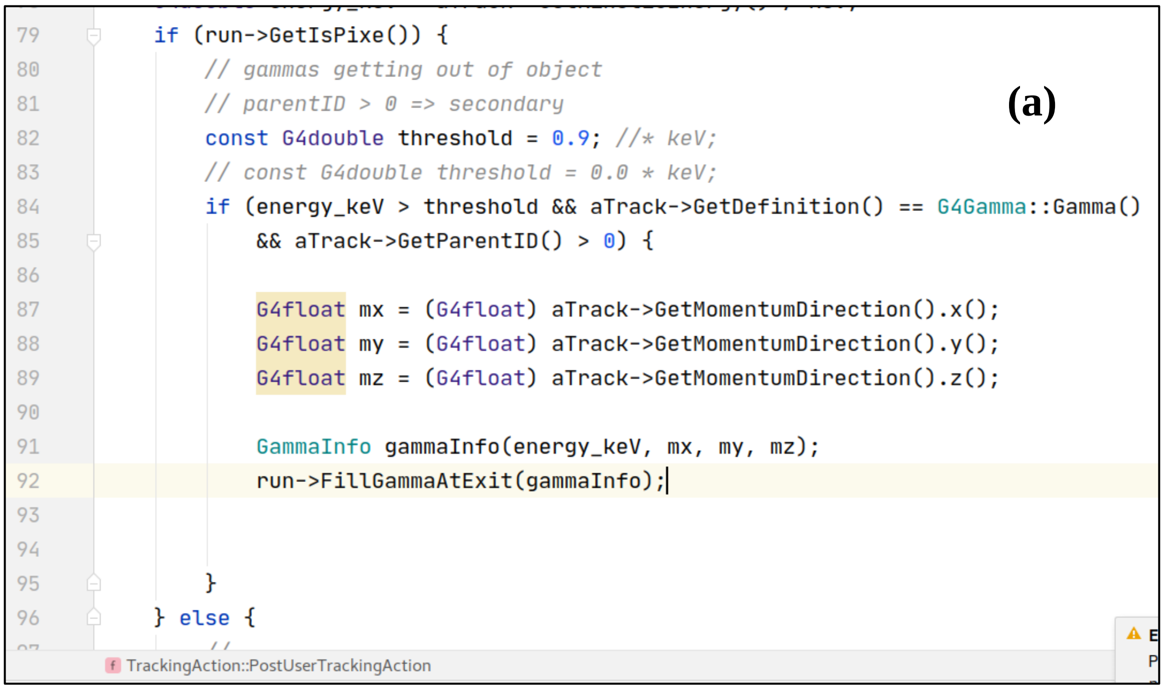
2.6. Detection of X-rays for PIXE-T

For PIXE-T, we collect the energy and momentum of the following two types of X-rays in two different files separately.

- The first file contains the information of emitted X-rays after they have gone through the phantom, which are also called "gamma at exit", because their information is collected after they get out of the phantom. The "at exit" information is collected in the ***TrackingAction::PostUserTrackingAction()*** (Figure 12a).

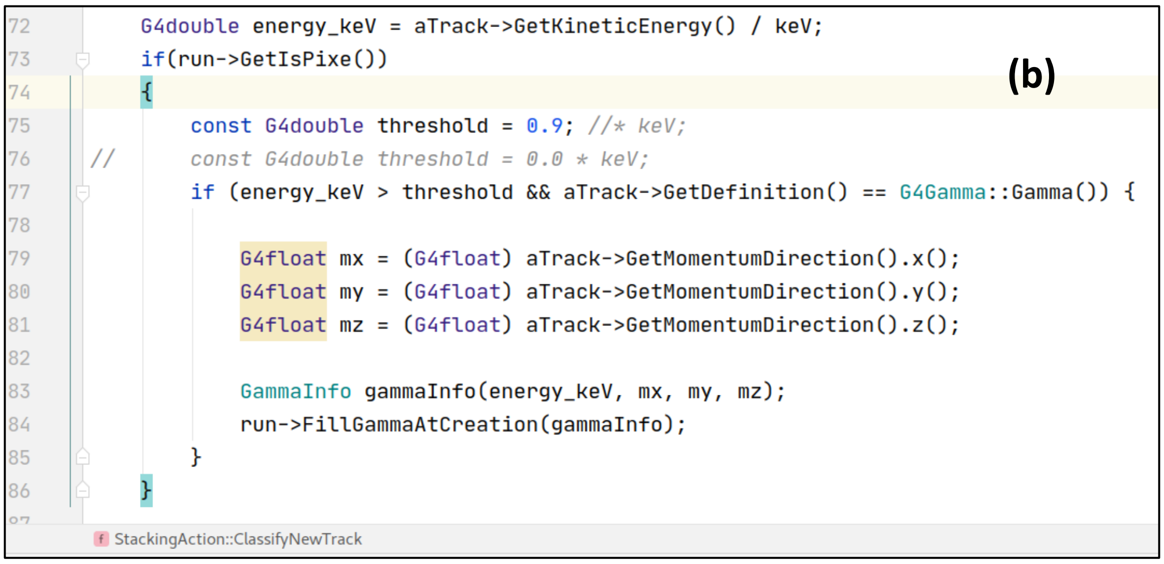
- The second file contains the information of X-rays just when they are created, which are also called "gamma at creation", because their information is collected at the point when they are generated. The "at creation" info is collected in the ***StackingAction::ClassifyNewTrack()*** method (Figure 12b).

The same structure type ***ParticleInfo*** is used to save the energy and momentum of every X-ray (Figure 11b).



```
    if (run->GetIsPixe()) {
        // gammas getting out of object
        // parentID > 0 => secondary
        const G4double threshold = 0.9; //* keV;
        // const G4double threshold = 0.0 * keV;
        if (energy_keV > threshold && aTrack->GetDefinition() == G4Gamma::Gamma()
            && aTrack->GetParentID() > 0) {

            G4float mx = (G4float) aTrack->GetMomentumDirection().x();
            G4float my = (G4float) aTrack->GetMomentumDirection().y();
            G4float mz = (G4float) aTrack->GetMomentumDirection().z();

            GammaInfo gammaInfo(energy_keV, mx, my, mz);
            run->FillGammaAtExit(gammaInfo);


        }
    } else {
```

TrackingAction::PostUserTrackingAction



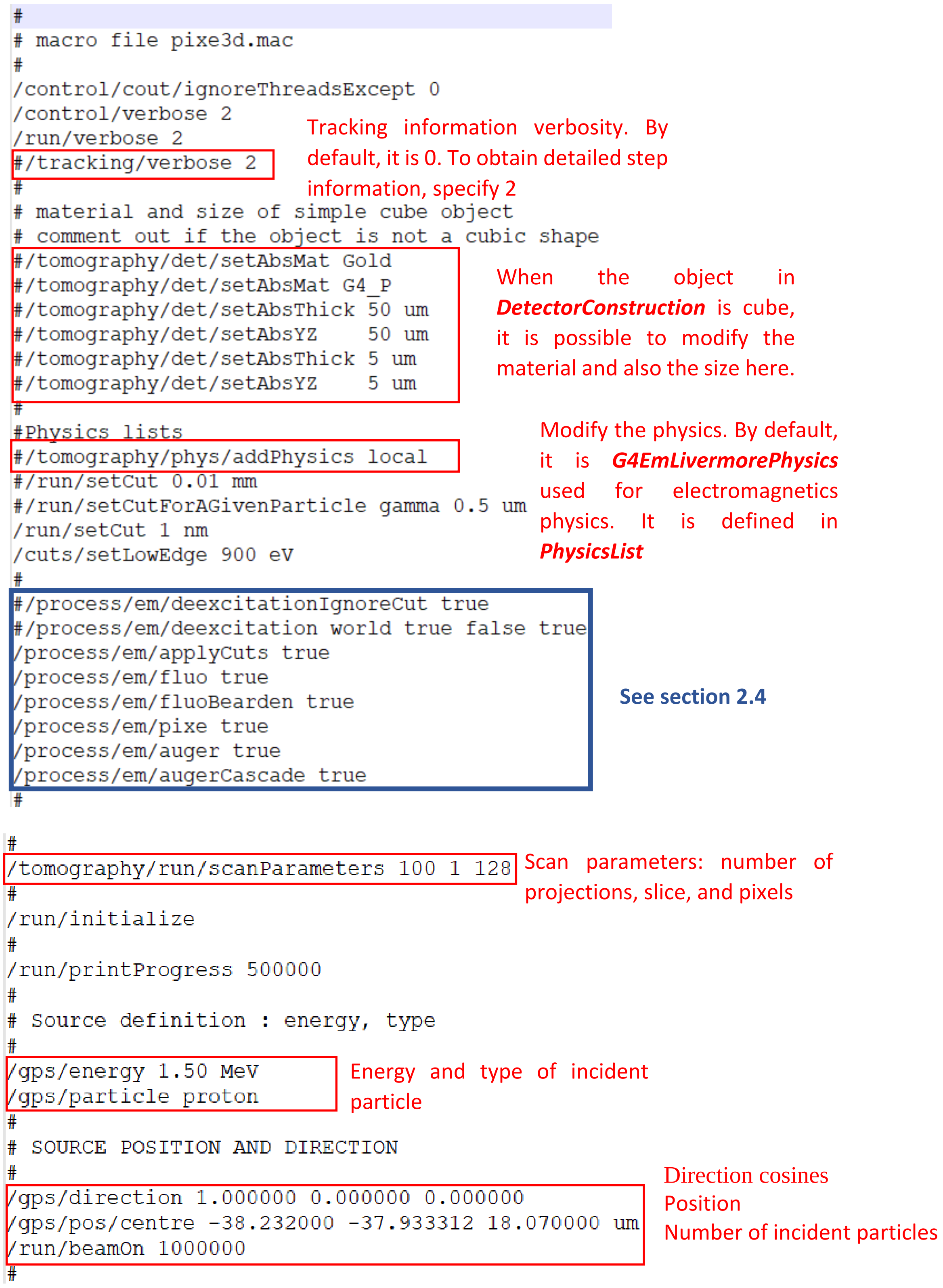
```
    G4double energy_keV = aTrack->GetKineticEnergy() / keV;
    if(run->GetIsPixe())
    {
        const G4double threshold = 0.9; //* keV;
//      const G4double threshold = 0.0 * keV;
        if (energy_keV > threshold && aTrack->GetDefinition() == G4Gamma::Gamma()) {

            G4float mx = (G4float) aTrack->GetMomentumDirection().x();
            G4float my = (G4float) aTrack->GetMomentumDirection().y();
            G4float mz = (G4float) aTrack->GetMomentumDirection().z();

            GammaInfo gammaInfo(energy_keV, mx, my, mz);
            run->FillGammaAtCreation(gammaInfo);
        }
    }
```

StackingAction::ClassifyNewTrack

Figure 12. Codes to collect the X-rays: (a) for X-rays at exit, (b) for X-rays at creation for PIXE-T.

2.7. An example of pixe3d.mac

Here we explain the role of the important lines in ***pixe3d.mac*** (Figure 13).



```
#
# macro file pixe3d.mac
#
/control/cout/ignoreThreadsExcept 0
/control/verbose 2
/run/verbose 2
#/tracking/verbose 2
#
# material and size of simple cube object
# comment out if the object is not a cubic shape
#/tomography/det/setAbsMat Gold
#/tomography/det/setAbsMat G4_P
#/tomography/det/setAbsThick 50 um
#/tomography/det/setAbsYZ    50 um
#/tomography/det/setAbsThick 5 um
#/tomography/det/setAbsYZ    5 um
#
#Physics lists
#/tomography/phys/addPhysics local
#/run/setCut 0.01 mm
#/run/setCutForAGivenParticle gamma 0.5 um
/run/setCut 1 nm
/cuts/setLowEdge 900 eV
#
#/process/em/deexcitationIgnoreCut true
#/process/em/deexcitation world true false true
/process/em/applyCuts true
/process/em/fluo true
/process/em/fluoBearden true
/process/em/pixe true
/process/em/auger true
/process/em/augerCascade true
#

#
/tomography/run/scanParameters 100 1 128
#
/run/initialize
#
/run/printProgress 500000
#
# Source definition : energy, type
#
/gps/energy 1.50 MeV
/gps/particle proton
#
# SOURCE POSITION AND DIRECTION
#
/gps/direction 1.000000 0.000000 0.000000
/gps/pos/centre -38.232000 -37.933312 18.070000 um
/run/beamOn 1000000
#
```

Tracking information verbosity. By default, it is 0. To obtain detailed step information, specify 2

When the object in ***DetectorConstruction*** is cube, it is possible to modify the material and also the size here.

Modify the physics. By default, it is ***G4EmLivermorePhysics*** used for electromagnetics physics. It is defined in ***PhysicsList***

See section 2.4

Scan parameters: number of projections, slice, and pixels

Energy and type of incident particle

Direction cosines
Position
Number of incident particles

*Figure 13. Important lines in **pixe3d.mac**.*

2.8. Execution of a simulation

First of all, let's assume that the user has already installed Geant4 under for illustration only, ***/your-G4-install/***, and ***stim_pixe_tomography*** example under ***/home/you/stim_pixe_tomography***. To run a simulation, the first step is to create a ***build*** directory.

```
cd stim_pixe_tomography        // enter in stim_pixe_tomography example
mkdir build                    // create a build directory in stim_pixe_tomography example
cd build                       // enter in build directory
cmake -DGeant4_DIR=/your-G4-install/lib64/your-G4-version /home/you/stim_pixe_tomography
                               // run CMake to generate the Makefiles needed to build the
                               application
make                           // build the executable
./stim_pixe_tomography -s pixe3d.mac      // run a simulation for STIM-T, the argument "-s" is
                               necessary, otherwise an error will occur
```

Similarly, for PIXE-T, run as follows:

```
./stim_pixe_tomography -p pixe3d.mac      // run a simulation for PIXE-T, the argument "-p" is
                               necessary, otherwise an error will occur.
// After correct configuration, the following lines will appear on the screen:
-- Configuring done
-- Generating done
-- Build files have been written to: …/stim_pixe_tomography/build
```

2.9. Visualization of phantoms

Type the following commands to visualize the phantoms:

```
make                           // compile the stim_pixe_tomography example
./stim_pixe_tomography         // visualize current phantom
```

Figure 14 - Figure 16 show the visualization of three built-in phantoms respectively.

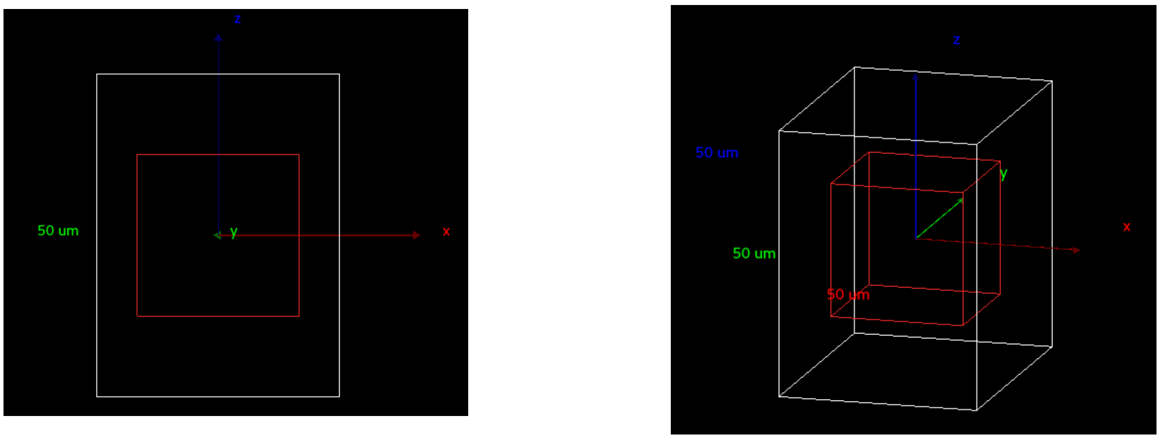


Figure 14. Visualization of built-in cube phantom

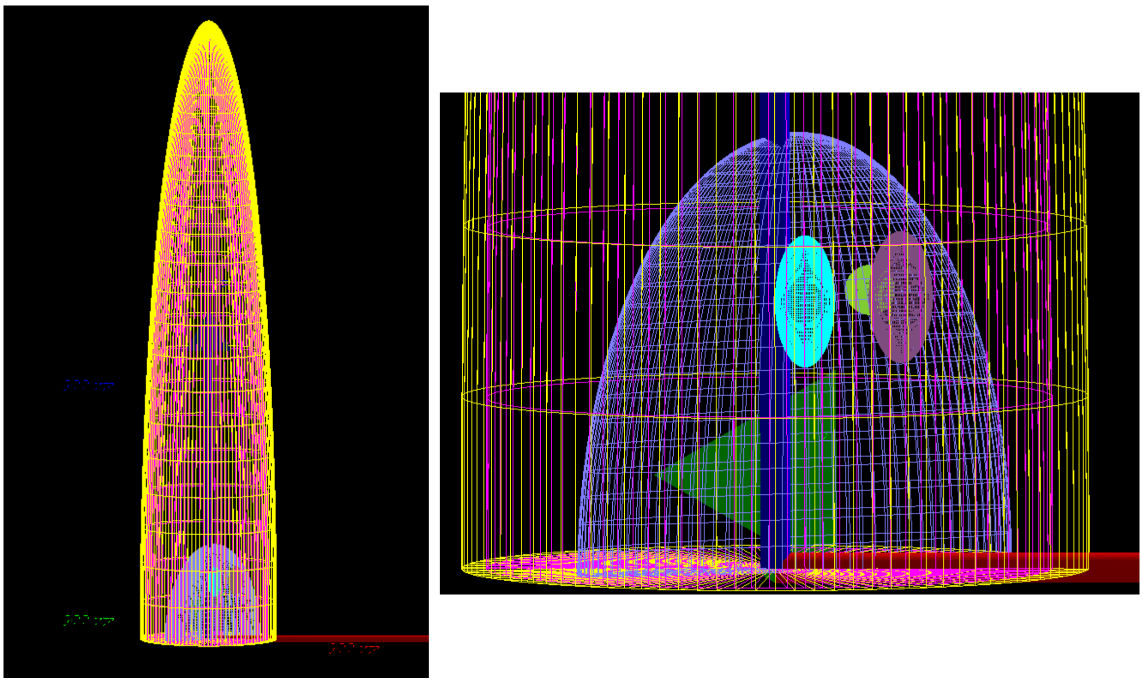


Figure 15. Visualization of built-in C. elegans phantom

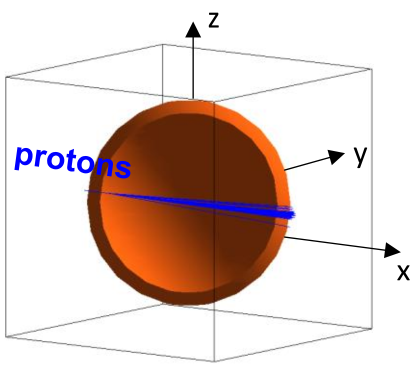


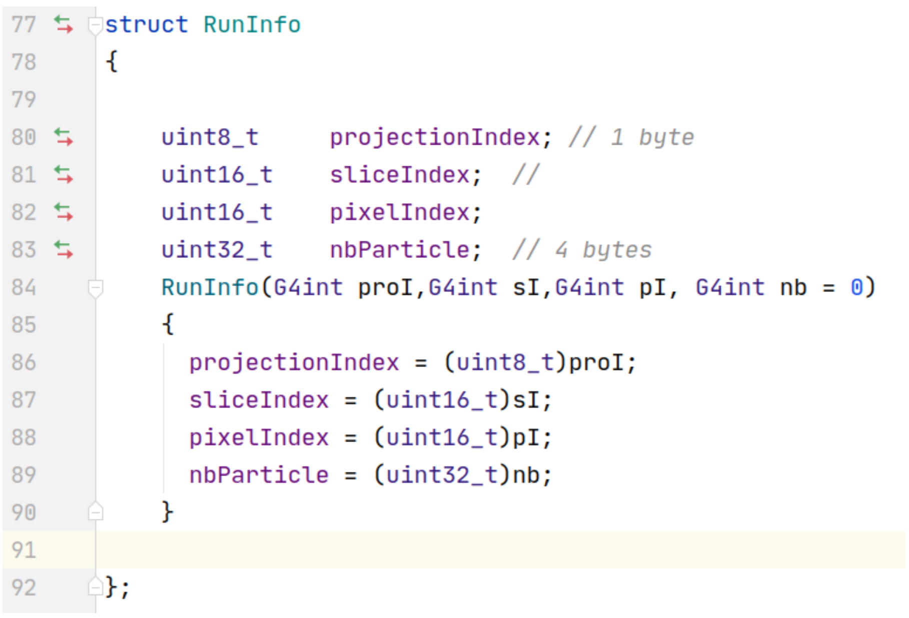
Figure 16. Visualization of the ICF target phantom

2.10. Format of simulation output files

For STIM-T, the output is a binary file called ***ProtonAtExit.dat***, saving the energy and momentum information of protons at exit for all the runs.

For PIXE-T, two binary files ***GammaAtCreation.dat*** and ***GammaAtExit.dat*** are generated at the end of the simulation, saving the energy and momentum information of X-rays at creation and at exit for all the runs.

For each run, the index of projection, slice, and pixel and the number of particles are written in the form of a structure type ***RunInfo*** in the output (Figure 17). Then the vector containing the energy and momentum of particles, in the form of a structure type ***ParticleInfo*** (Figure 11b) is written after the ***RunInfo*** (Figure 18).



```
struct RunInfo
{

    uint8_t     projectionIndex; // 1 byte
    uint16_t    sliceIndex;  //
    uint16_t    pixelIndex;
    uint32_t    nbParticle;  // 4 bytes
    RunInfo(G4int proI,G4int sI,G4int pI, G4int nb = 0)
    {
      projectionIndex = (uint8_t)proI;
      sliceIndex = (uint16_t)sI;
      pixelIndex = (uint16_t)pI;
      nbParticle = (uint32_t)nb;
    }

};
```

Figure 17. Structure type ***RunInfo***

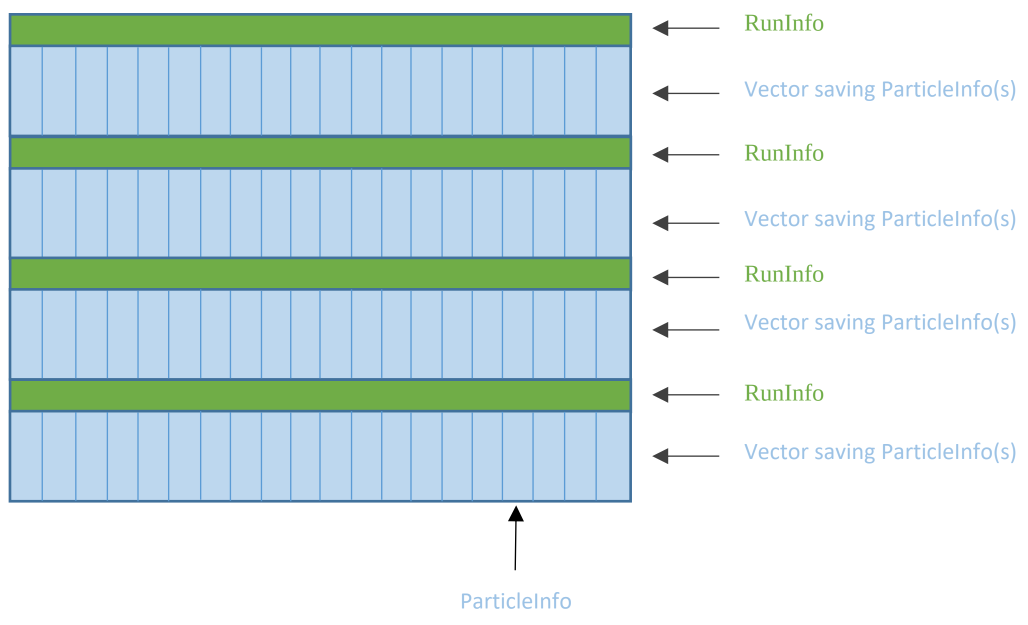

RunInfo
Vector saving ParticleInfo(s)
RunInfo
Vector saving ParticleInfo(s)
RunInfo
Vector saving ParticleInfo(s)
RunInfo
Vector saving ParticleInfo(s)
ParticleInfo

Figure 18. Structure of the output file for a STIM-T or PIXE-T simulation

3. Selection of particles of interest

For STIM-T, the particle of interest is transmitted proton after the simulation.

For PIXE-T, the particle of interest is secondary emitted X-ray.

During the simulation, all the transmitted protons or all the X-rays (in 4π solid angle) are collected whatever their direction. In order to model a detector at a given position and with a specific angular aperture, we select the data ***after*** the simulation.

3.1. Parameters of selection

To understand the principle of the selection, we need to first specify some necessary variables:

- ***nbProjection***: number of projections
- ***nbSlice***: number of slices
- ***nbPixel***: number of pixels
- ***totalAngleSpan***: angle of scan, for example 180°
- ***angleOfDetector***: angle between the position of detector and the direction of incident protons ($\widehat{AOx}$ in Figure 19a for first projection and ($\widehat{AOx'}$) in Figure 20).
- ***distanceObjectDetector***: distance between scanned object and detector (OA in Figure 19a).
- ***radiusOfDetector***: radius of entrance window of detector (AB in Figure 19a)
- **theta:** half apex angle of detection, $\theta = \widehat{AOB}$ in Figure 19 and Figure 20.

Figure 19 shows the positions of detector and object at first projection 0.

Figure 20 shows the positions of detector and object at the projection i. Whatever the projection is, the angle between the position of detector and the direction of incident protons keeps fixed.

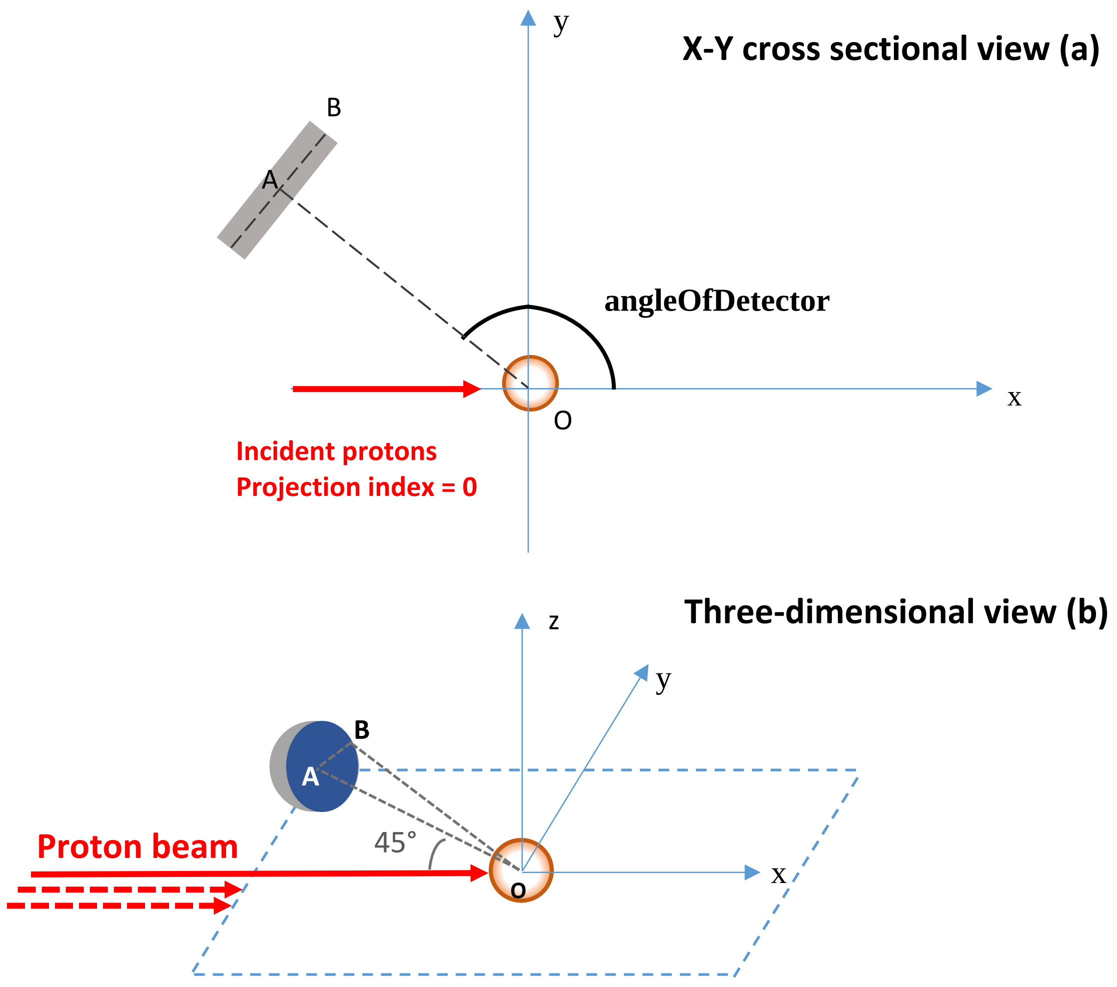


Figure 19. Schematic position of detector at projection index = 0 for PIXE-T experiments, in this case, ***angleOfDetector*** *= 135°*

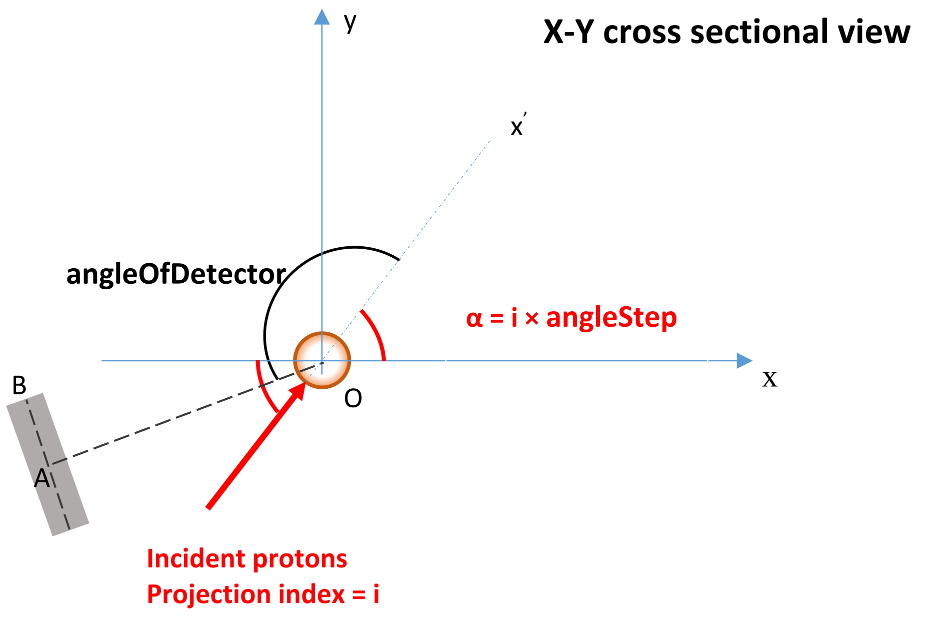


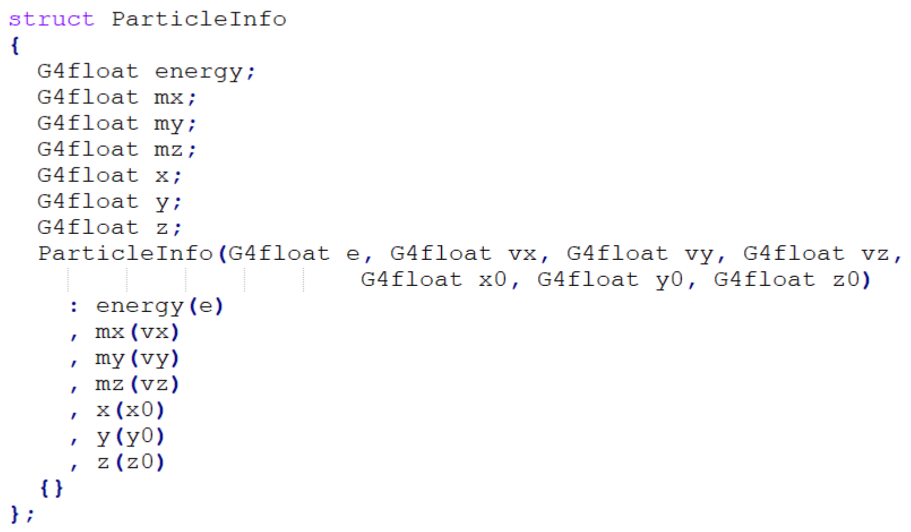
Figure 20. Schematic position of detector at projection index = i for PIXE-T experiments, in this case, ***angleOfDetector*** *= 135°.*

3.2. Principle of selection

The principle of the selection is based on the position of the detector according to the sample and the beam. It also depends on the shape and size of the entrance window, which is assumed circular in our case. In fact, we provide two methods of selection:

- Selection with particle momentum
- Selection with particle position and momentum

By default, the first option, *i.e.* the selection with particle momentum is proposed. It is based on the assumption that the phantom size is negligible relative to the size and distance of the detector, which is usually the case for micro-tomography. Of course, this will significantly increase the size of simulated data. Indeed, the method of selection determines the definition of the ***struct ParticleInfo***, which stores particle information as mentioned in section 2.5 (Figure 11b). For the selection using position and momentum, the position of the particle of interest is additionally saved in ***ParticleInfo*** (Figure 21) defined in ***Run.hh***.



```
struct ParticleInfo
{
  G4float energy;
  G4float mx;
  G4float my;
  G4float mz;
  G4float x;
  G4float y;
  G4float z;
  ParticleInfo(G4float e, G4float vx, G4float vy, G4float vz,
                         G4float x0, G4float y0, G4float z0)
    : energy(e)
    , mx(vx)
    , my(vy)
    , mz(vz)
    , x(x0)
    , y(y0)
    , z(z0)
  {}
};
```

Figure 21. ***ParticleInfo*** *definition in case of selection with momentum and position.*

3.3. Selection with particle momentum

The approximate selection using particle momentum is based on the assumption that the phantom size is negligible relative to the size and distance of the detector. In this case, the selection only requires to parameterize the angular position $\boldsymbol{\alpha}$ of detector and its half apex angle $\boldsymbol{\theta}$. The X-ray is considered "detected" only if the angle between its momentum and $\overrightarrow{OA}$ (Figure 19b) is less than the half apex angle $\boldsymbol{\theta}$ ($\widehat{AOB}$ in Figure 19b). $\boldsymbol{\theta}$ depends on the radius and distance of detector:

$$\boldsymbol{\theta} = \text{atan}\,\frac{\text{radiusOfDetector}}{\text{distanceObjectDetector}}$$

In the scripts described in the following section 3.3, there are two ways to define $\boldsymbol{\theta}$:

- Users can specify directly ***radiusOfDetector*** and ***distanceObjectDetector***
- Users can arbitrarily define $\boldsymbol{\theta}$ value especially when they want a large solid angle for the detection. A large solid angle can significantly reduce the PIXE-T simulation time since it requires less incident particles.

3.4. Selection with particle position and momentum

The selection using particle position and momentum is a more precise method that should used when the assumption of negligible phantom size is not valid. In this case, the selection requires to parameterize the angular position $\boldsymbol{\alpha}$, distance $\boldsymbol{l}$ (***distanceObjectDetector***) and radius$\boldsymbol{r}$ (***radiusOfDetector***) of the detector.

To describe this method, we define (Figure 22):

- Point C (x_0, y_0, z_0): the position of the particle when being collected in the simulation, *i.e.* either at the end of the track or at the point where the particle is generated, according to the user needs.
- $\overrightarrow{n_0}$ (a, b, c): the momentum of the particle
- $\vec{n} = \overrightarrow{OA}$: the vector orthogonal to the detector entrance window
- Point P(x_i, y_i, z_i): the intersection point between the trajectory of the particle and the detector entrance window.

According to the angular position of the detector, we have:

$$\vec{n} = (l\cos\alpha\,, l\sin\alpha\,, 0)$$

So the position of center point of the entrance window is: A $(l\cos\alpha\,, l\sin\alpha\,, 0)$

Knowing the vector $\vec{n}$ orthogonal to the detector entrance window and the point A, the equation of the plane of the entrance window is established as follows:

$$l\cos\alpha\,(x - l\cos\alpha) + \; l\sin\alpha\,(y - l\sin\alpha) + 0(z-0) = 0 \quad (3.1)$$

Knowing the point C (x_0, y_0, z_0) and momentum $\overrightarrow{n_0}$ (a, b, c), the trajectory of the particle is defined as follows:

$$\begin{cases} x = x_0 + at \\ y = y_0 + bt \\ z = z_0 + ct \end{cases} \qquad (3.2)$$

Where t is a constant

Substituting (3.2) to (3.1), we can obtain the intersection point P of the trajectory and the detector plane by calculating t:

$$t = \frac{(l\cos\alpha\ - x_0)l\cos\alpha + (l\sin\alpha - y_0)l\sin\alpha}{al\cos\alpha + bl\sin\alpha} = \frac{l^2 - x_0 l\cos\alpha - y_0 l\sin\alpha}{al\cos\alpha + bl\sin\alpha}$$

The condition $al\cos\alpha + bl\sin\alpha = 0$ should be excluded, because it means the trajectory is parallel to the detector plane. In this case, the particle is not detected.

Thus, the intersection point P (x_i, y_i, z_i) is:

$$\begin{cases} x_i = x_0 + at \\ y_i = y_0 + bt \\ z_i = z_0 + ct \end{cases}$$

We calculate the distance PA. If PA is less than the radius of detector $\boldsymbol{r}$, it means the particle is detected.

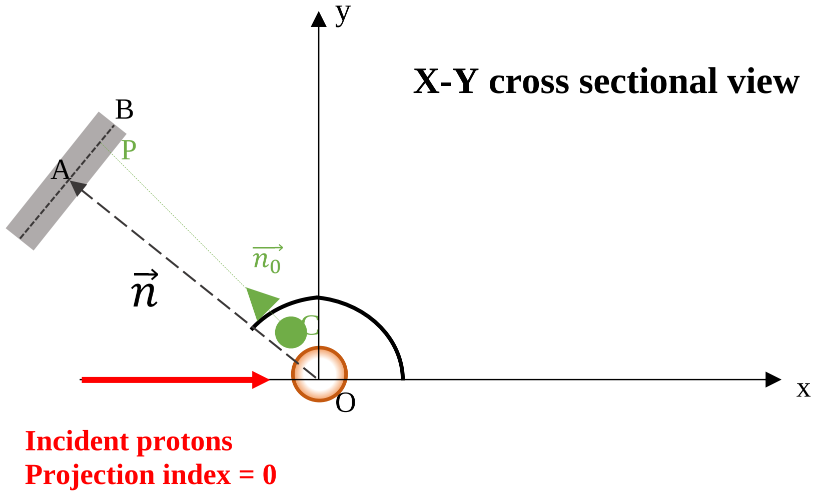


Figure 22. Detection of particle using precise selection.

3.5. Implementation of selection scripts

3.5.1. Selection procedure using particle momentum

Three scripts for the selection using particle momentum are available to read output files for STIM-T transmitted protons and PIXE-T X-rays. Running the scripts generates the final files, which will be used as an input for the tomographic reconstruction code, TomoRebuild for example, to generate the reconstructed images:

- ***BinToStd_ProtonAtExit.C***,

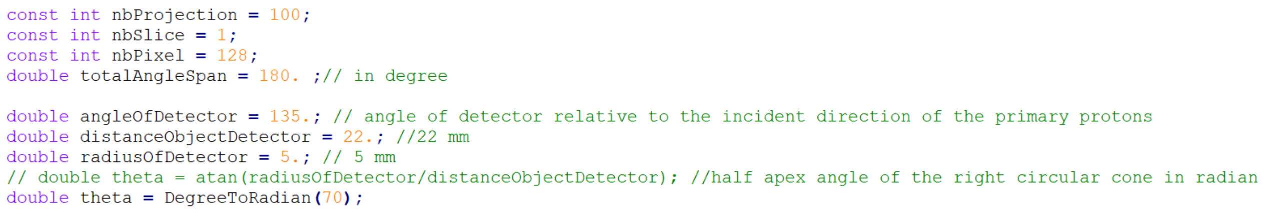
  Reads proton information from ***ProtonAtExit.dat***, selects the data and generates ***StimEvent_std_AtExit.DAT***
- ***BinToStd_GammaAtCreation.C***,

  Reads X-ray information from ***GammaAtCreation.dat***, selects the data and generates ***PixeEvent_std_AtCreation.DAT***
- ***BinToStd_GammaAtExit.C***,

  Reads X-ray information from ***GammaAtExit.dat***, selects the data and generates ***PixeEvent_std_AtExit.DAT***

It should be noted that ***BinToStd_GammaAtCreation.C*** and ***BinToStd_GammaAtExit.C*** are nearly same scripts, the only difference is the file path.

The user should specify the following variables (Figure 23) in the scripts as explained in section 3.1:



```
const int nbProjection = 100;
const int nbSlice = 1;
const int nbPixel = 128;
double totalAngleSpan = 180. ;// in degree

double angleOfDetector = 135.; // angle of detector relative to the incident direction of the primary protons
double distanceObjectDetector = 22.; //22 mm
double radiusOfDetector = 5.; // 5 mm
// double theta = atan(radiusOfDetector/distanceObjectDetector); //half apex angle of the right circular cone in radian
double theta = DegreeToRadian(70);
```

Figure 23. Parameters to define in the scripts

Attention, the ***angleOfDetector*** is usually 0° for STIM-T, which is in accordance with experiments (STIM detector in forward position at 0°).

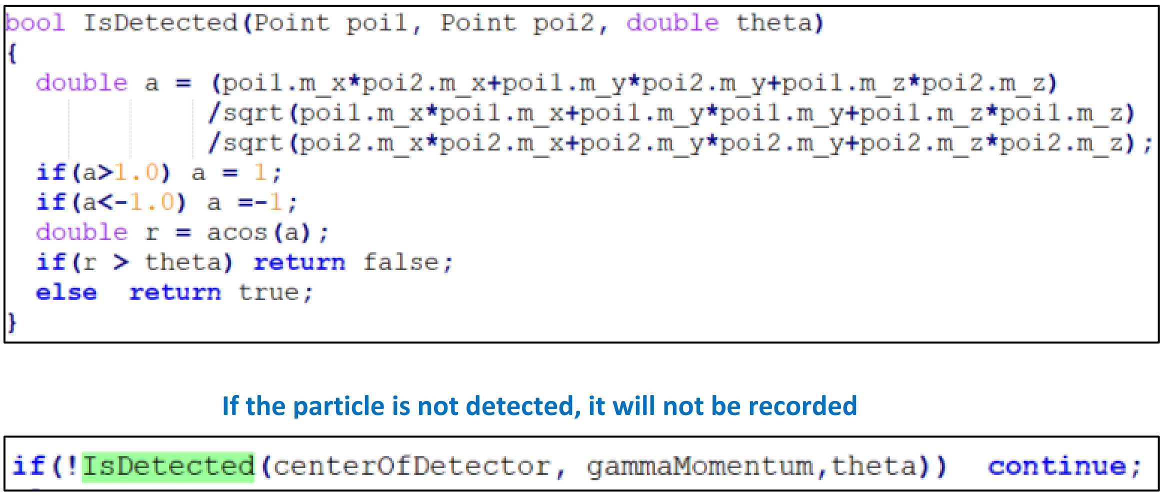
The selection of X-rays or protons is achieved by the ***IsDetected***() method (Figure 24), by comparing the angle between the particle momentum and the direction of the center of detector ($\overrightarrow{OA}$) with the half apex angle theta (Figure 19b).

- ***centerofDetector*** refers to the direction of the center of detector
- ***gammaMomentum*** or ***protonMomentum*** refer to the momentum of X-ray or proton.

Suppose the particle momentum is (x_0, y_0, z_0), and $\overrightarrow{OA}$ is (x_1, y_1, z_1). The angle between particle momentum and $\overrightarrow{OA}$ is β.

β is calculated as followsing:

$$\cos\beta = \frac{\overrightarrow{OA}\cdot\overrightarrow{OC}}{|\overrightarrow{OA}|\times|\overrightarrow{OC}|} = \frac{x_0x_1 + y_0y_1 + z_0z_1}{\sqrt{{x_0}^2 + {y_0}^2 + {z_0}^2}\times\sqrt{{x_1}^2 + {y_1}^2 + {z_1}^2}}$$



```
bool IsDetected(Point poi1, Point poi2, double theta)
{
  double a = (poi1.m_x*poi2.m_x+poi1.m_y*poi2.m_y+poi1.m_z*poi2.m_z)
             /sqrt(poi1.m_x*poi1.m_x+poi1.m_y*poi1.m_y+poi1.m_z*poi1.m_z)
             /sqrt(poi2.m_x*poi2.m_x+poi2.m_y*poi2.m_y+poi2.m_z*poi2.m_z);
  if(a>1.0) a = 1;
  if(a<-1.0) a =-1;
  double r = acos(a);
  if(r > theta) return false;
  else  return true;
}
```

If the particle is not detected, it will not be recorded

```
if(!IsDetected(centerOfDetector, gammaMomentum,theta))  continue;
```

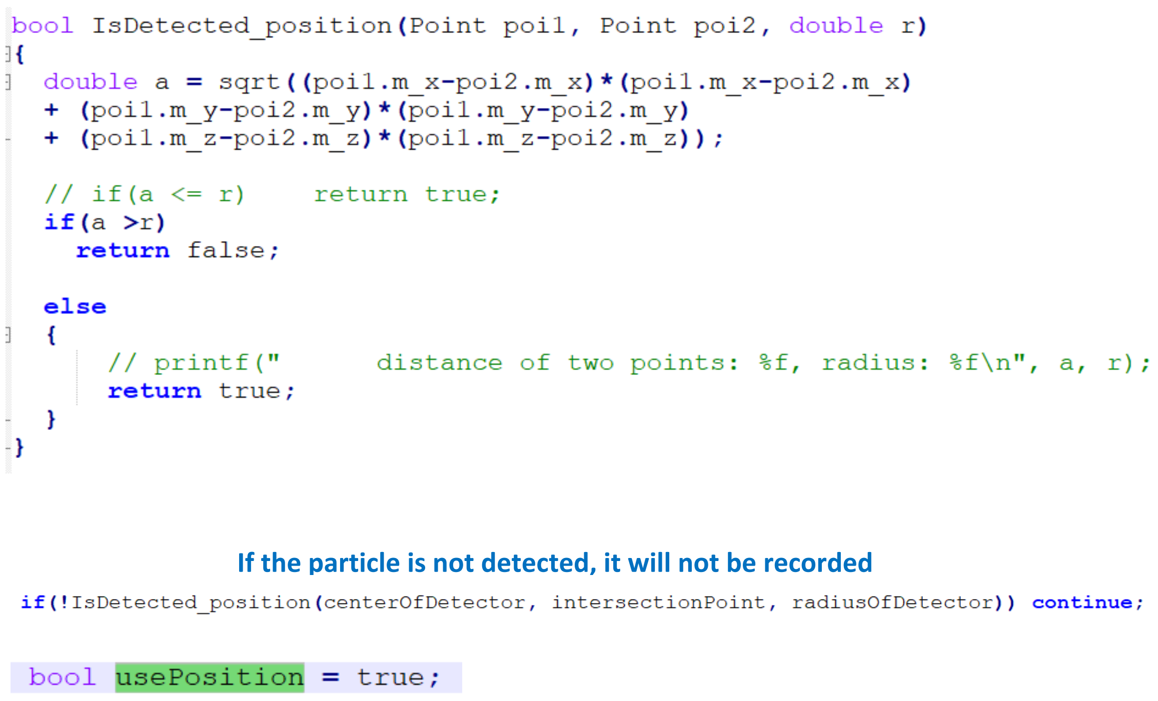
Figure 24. Codes to select particles using momentum.

<u>3.5.2. Selection procedure using particle momentum and position</u>

The scripts used for the selection for protons and X-rays using particle position and momentum are:

- ***BinToStd_proton_position.C***
- ***BinToStd_gamma_position.C***

Similarly, the selection of X-rays or protons is achieved by the ***IsDetected_position()*** method (Figure 25), by comparing the distance of intersection point P(x_i, y_i, z_i) (Figure 22) to the center of detector with the radius of the detector. Note that in these two scripts, the selection using only momentum is also available if the variable ***usePosition*** is ***false***.



```
bool IsDetected_position(Point poi1, Point poi2, double r)
{
  double a = sqrt((poi1.m_x-poi2.m_x)*(poi1.m_x-poi2.m_x)
  + (poi1.m_y-poi2.m_y)*(poi1.m_y-poi2.m_y)
  + (poi1.m_z-poi2.m_z)*(poi1.m_z-poi2.m_z));

  // if(a <= r)     return true;
  if(a >r)
    return false;

  else
  {
      // printf("      distance of two points: %f, radius: %f\n", a, r);
      return true;
  }
}
```

If the particle is not detected, it will not be recorded

```
if(!IsDetected_position(centerOfDetector, intersectionPoint, radiusOfDetector)) continue;
```

```
bool usePosition = true;
```

Figure 25. Codes to select particles using momentum and position.

3.5.3. Execution of the scritps

Type the following command to use the script:

```
root BinToStd_GammaAtCreation.C
```

- Idem for ***BinToStd_GammaAtExit.C***, ***BinToStd_ProtonAtExit.C***, ***BinToStd_gamma_position.C*** and ***BinToStd_proton_position.C***

3.6. Format of files for tomographic reconstruction

As mentioned in section 3.3, the tomographic reconstruction code that is used is TomoRebuild. The X-rays and protons are selected using the corresponding scripts. The relevant information is written in ***ProtonAtExit.dat***, ***GammaAtCreation.dat*** and ***GammaAtExit.dat***.

For STIM-T, a structure type ***StimEvent*** is defined in ***BinToStd_ProtonAtExit.C***, containing the following information of a proton (Figure 26):

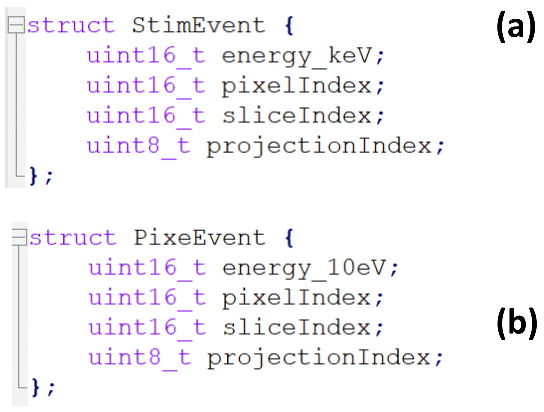
- ***energy_keV***, the energy of the proton in **keV**
- ***pixelIndex***, the index of pixel of the run that generates the proton
- ***sliceIndex***, the index of slice of the run that generates the proton
- ***projectionIndex***, the index of projection of the run that generates the proton

Eventually, a binary file named ***StimEvent_std.DAT*** file will be generated.

Similarly, for PIXE-T, a structure type ***PixeEvent*** is defined in ***BinToStd_GammaAtCreation.C*** and ***BinToStd_GammaAtExit.C***, containing the following information:

- ***energy_10eV***, the energy of the X-ray in unit **10 eV**
- ***pixelIndex***, the index of pixel of the run that generates the X-ray
- ***sliceIndex***, the index of slice of the run that generates the X-ray
- ***projectionIndex***, the index of projection of the run that generates the X-ray

Eventually, a binary file named ***PixeEvent_std.DAT*** file will be generated.



```
struct StimEvent {
    uint16_t energy_keV;
    uint16_t pixelIndex;
    uint16_t sliceIndex;
    uint8_t projectionIndex;
};

struct PixeEvent {
    uint16_t energy_10eV;
    uint16_t pixelIndex;
    uint16_t sliceIndex;
    uint8_t projectionIndex;
};
```

Figure 26. Definition of the structure for StimEvent (a) and PixeEvent (b).

4. Interruption of a simulation

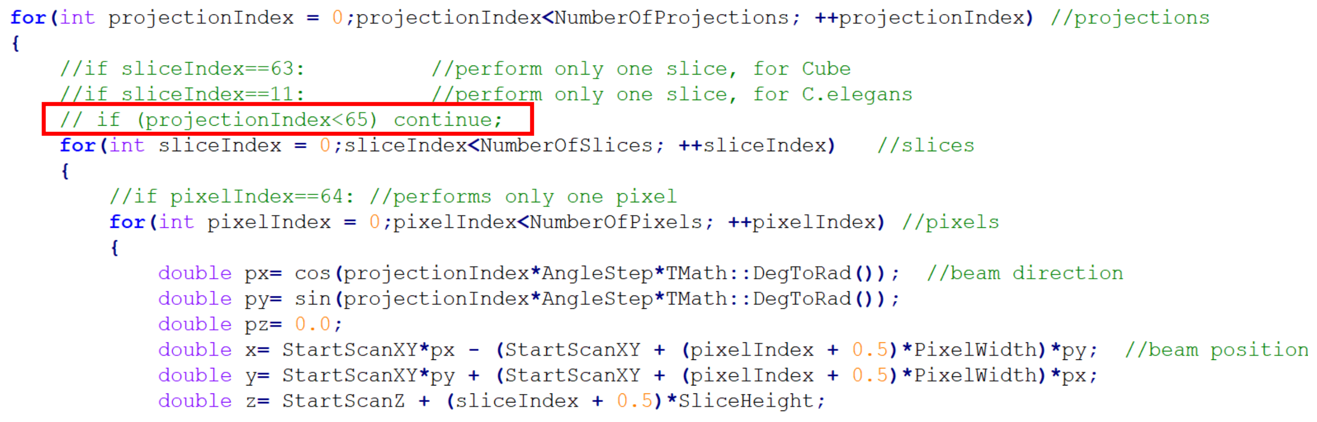
If a simulation is interrupted by accident, like power failure, it is feasible to continue the simulation from the projection when the interruption occurs. For this, you need to know exactly the projection where it stopped.

In this case, we provide the following scripts to identify the projection of interruption:

- ***LocateInterruption_ProtonAtExit.C*** for a STIM-T simulation
- ***LocateInterruption_GammaAtExit.C*** for a PIXE-T simulation

The user can resume the simulation from where it was interrupted, using a new ***pixe3d.mac*** generated by adjusting parameters in ***GPSPointLoop.C*** once the user is aware of the projection of interruption. For instance, if the simulation was interrupted at projection 65, then it should be re-starded at projection 65 (redo the interrupted projection). Thus, users just need to uncomment the line (Figure 27):

if (projectionIndex<65) continue;



```
for(int projectionIndex = 0;projectionIndex<NumberOfProjections; ++projectionIndex) //projections
{
    //if sliceIndex==63:          //perform only one slice, for Cube
    //if sliceIndex==11:          //perform only one slice, for C.elegans
    // if (projectionIndex<65) continue;
    for(int sliceIndex = 0;sliceIndex<NumberOfSlices; ++sliceIndex)   //slices
    {
        //if pixelIndex==64: //performs only one pixel
        for(int pixelIndex = 0;pixelIndex<NumberOfPixels; ++pixelIndex) //pixels
        {
            double px= cos(projectionIndex*AngleStep*TMath::DegToRad());  //beam direction
            double py= sin(projectionIndex*AngleStep*TMath::DegToRad());
            double pz= 0.0;
            double x= StartScanXY*px - (StartScanXY + (pixelIndex + 0.5)*PixelWidth)*py;  //beam position
            double y= StartScanXY*py + (StartScanXY + (pixelIndex + 0.5)*PixelWidth)*px;
            double z= StartScanZ + (sliceIndex + 0.5)*SliceHeight;
```

Figure 27. Code in ***GPSPointLoop.C****.*

It should be noted that ***LocateInterruption_ProtonAtExit.C*** and ***LocateInterruption_GammaAtExit.C*** are nearly same scripts, the only difference is the file path.

In case of interruption, the selection of X-rays and protons is accomplished by other scripts:

- ***Concatenate_BinToStd_ProtonAtExit.C***

  Reads X-ray information from ***ProtonAtExit_1.dat*** and ***ProtonAtExit_2.dat***, rewrites in ***StimEvent_std_AtExit.DAT***

  - ***ProtonAtExit_1.dat*** is the output file that was interrupted
  - ***ProtonAtExit_2.dat*** is the output file that completes the rest of the simulation

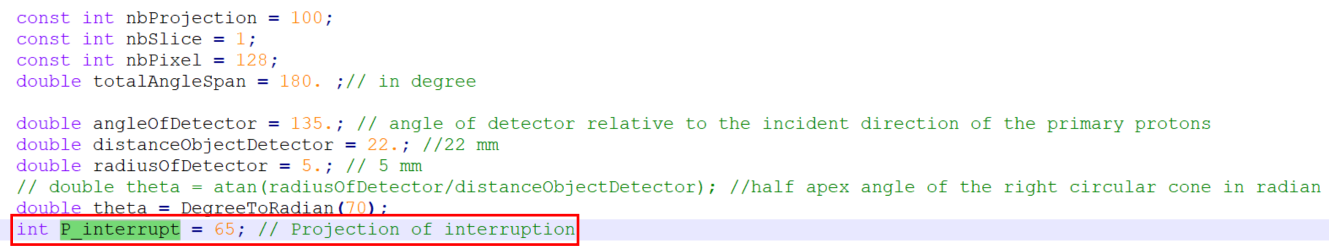
- ***Concatenate_BinToStd_GammaAtCreation.C***

  Reads X-ray information from ***GammaAtCreation_1.dat*** and ***GammaAtCreation_2.dat***, rewrites in ***PixeEvent_std_AtCreation.DAT***

- ***Concatenate_BinToStd_GammaAtExit.C***

  Reads X-ray information from ***GammaAtExit_1.dat*** and ***GammaAtExit_2.dat***, rewrites in ***PixeEvent_std_AtExit.DAT***

Parameters should be given like explained in section 3.5. In addition, the user should set ***P_interrupt*** variable (Figure 28), which is the projection of interruption.

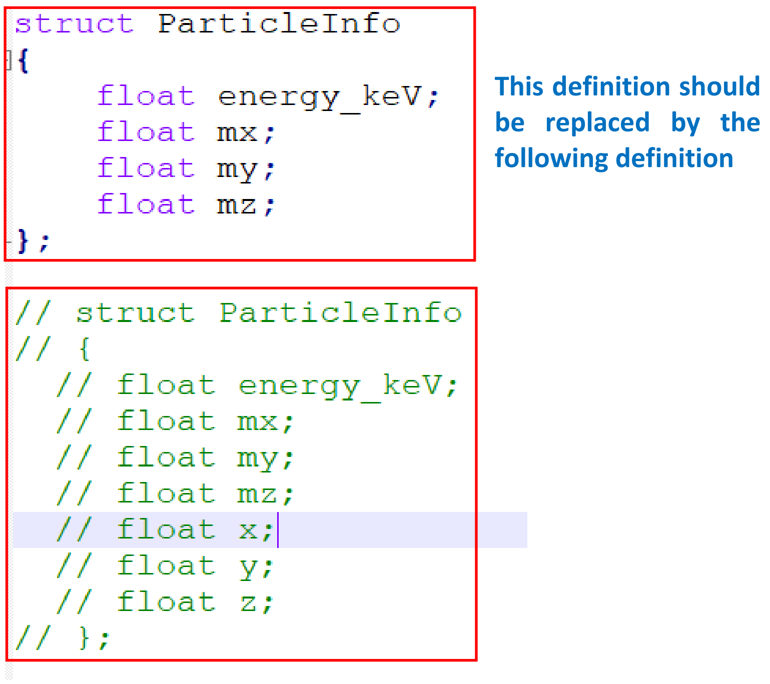


```
const int nbProjection = 100;
const int nbSlice = 1;
const int nbPixel = 128;
double totalAngleSpan = 180. ;// in degree

double angleOfDetector = 135.; // angle of detector relative to the incident direction of the primary protons
double distanceObjectDetector = 22.; //22 mm
double radiusOfDetector = 5.; // 5 mm
// double theta = atan(radiusOfDetector/distanceObjectDetector); //half apex angle of the right circular cone in radian
double theta = DegreeToRadian(70);
int P_interrupt = 65; // Projection of interruption
```

Figure 28. Parameters to define in case of interruption.

It should be noted that ***Concatenate_BinToStd_GammaAtCreation.C*** and ***Concatenate_BinToStd_GammaAtExit.C*** are nearly same scripts, the only difference is the file path.

In addition, it should be noted that all the scripts described in this section deal with ***ParticleInfo***, which does not consider the position of the particle (Figure 11b). If the interrupted simulation considers the position of the particle, with ***ParticleInfo*** defined as in Figure 21, then the ***LocateInterruption_ProtonAtExit.C*** and ***LocateInterruption_GammaAtExit.C*** scripts should be modified.



```
struct ParticleInfo
{
    float energy_keV;
    float mx;
    float my;
    float mz;
};
```

This definition should be replaced by the following definition

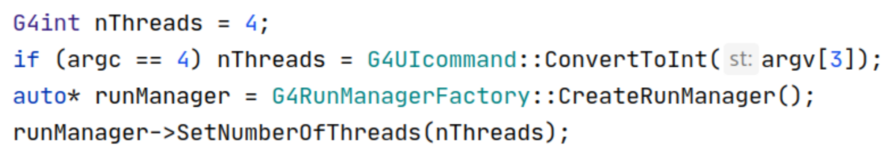
```
// struct ParticleInfo
// {
  // float energy_keV;
  // float mx;
  // float my;
  // float mz;
  // float x;
  // float y;
  // float z;
// };
```

Figure 29. ***ParticleInfo*** *definition in* **LocateInterruption_ProtonAtExit.C** *and* **LocateInterruption_GammaAtExit.C**.

5. Multithreading

stim_pixe_tomography example normally runs in multithreaded mode with the default number of threads depending on the device. But the user can specify the number of threads ***nThreads*** in ***stim_pixe_tomography.cc*** (Figure 30), or specify it as an argument when running the simulation:

```
./stim_pixe_tomography -p pixe3d.mac 100        // 100 is the number of threads

./stim_pixe_tomography -p pixe3d.mac 1          //  sequential mode
```



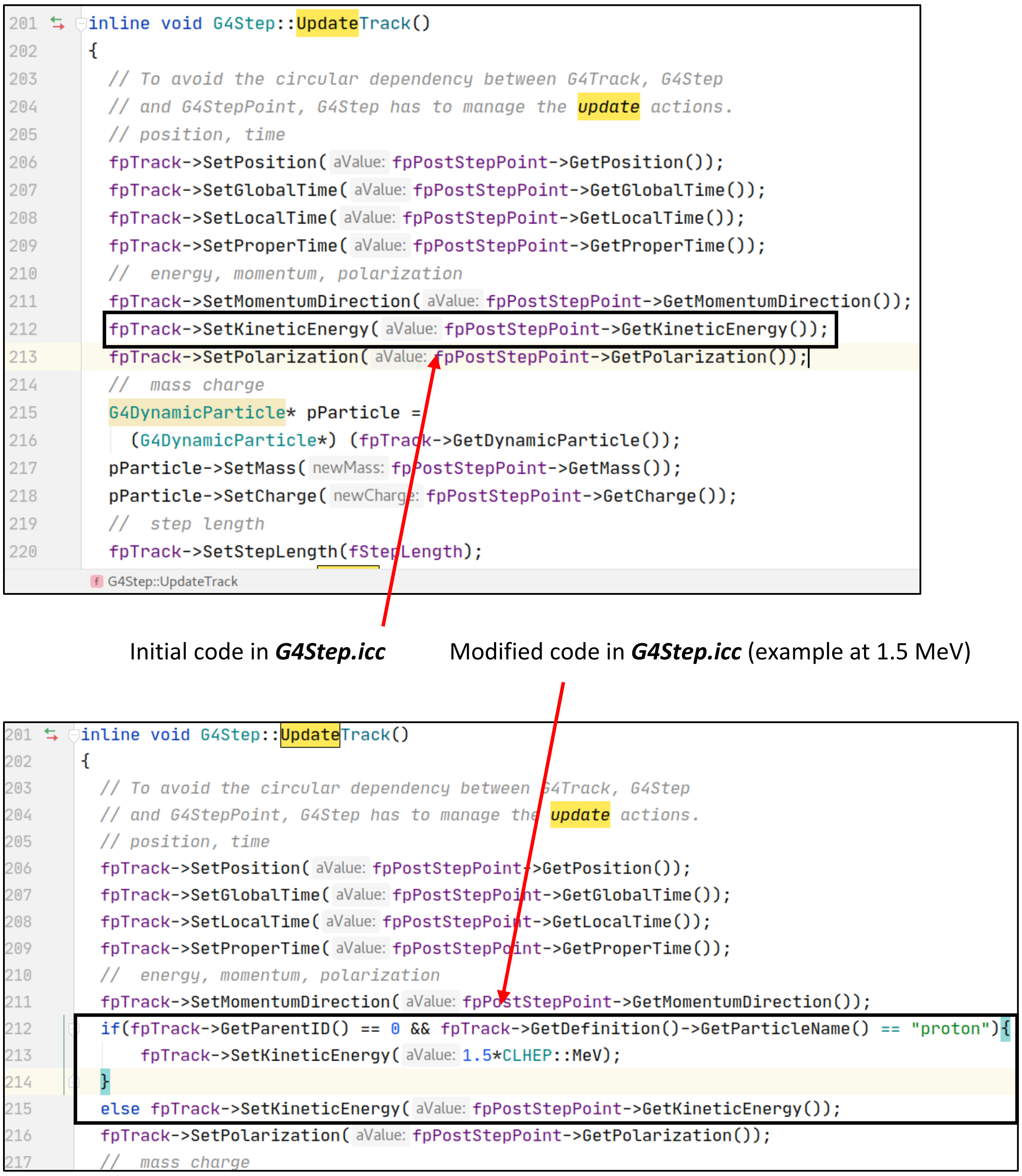
```
G4int nThreads = 4;
if (argc == 4) nThreads = G4UIcommand::ConvertToInt( st: argv[3]);
auto* runManager = G4RunManagerFactory::CreateRunManager();
runManager->SetNumberOfThreads(nThreads);
```

Figure 30. Codes to set threads.

6. Simulation keeping a constant energy for the protons

As explained in the previous publication [3], the user can make a simulation whilst keeping a constant energy for protons in the material, for example 1.5 MeV. This assumes there is no energy loss of protons.

This assumption can be achieved by modifying the ***source code of Geant4*** (Figure 31).



```
inline void G4Step::UpdateTrack()
{
  // To avoid the circular dependency between G4Track, G4Step
  // and G4StepPoint, G4Step has to manage the update actions.
  // position, time
  fpTrack->SetPosition( aValue: fpPostStepPoint->GetPosition());
  fpTrack->SetGlobalTime( aValue: fpPostStepPoint->GetGlobalTime());
  fpTrack->SetLocalTime( aValue: fpPostStepPoint->GetLocalTime());
  fpTrack->SetProperTime( aValue: fpPostStepPoint->GetProperTime());
  //  energy, momentum, polarization
  fpTrack->SetMomentumDirection( aValue: fpPostStepPoint->GetMomentumDirection());
  fpTrack->SetKineticEnergy( aValue: fpPostStepPoint->GetKineticEnergy());
  fpTrack->SetPolarization( aValue: fpPostStepPoint->GetPolarization());
  //  mass charge
  G4DynamicParticle* pParticle =
    (G4DynamicParticle*) (fpTrack->GetDynamicParticle());
  pParticle->SetMass( newMass: fpPostStepPoint->GetMass());
  pParticle->SetCharge( newCharge: fpPostStepPoint->GetCharge());
  //  step length
  fpTrack->SetStepLength(fStepLength);
```

G4Step::UpdateTrack
Initial code in ***G4Step.icc***
Modified code in ***G4Step.icc*** (example at 1.5 MeV)

```
inline void G4Step::UpdateTrack()
{
  // To avoid the circular dependency between G4Track, G4Step
  // and G4StepPoint, G4Step has to manage the update actions.
  // position, time
  fpTrack->SetPosition( aValue: fpPostStepPoint->GetPosition());
  fpTrack->SetGlobalTime( aValue: fpPostStepPoint->GetGlobalTime());
  fpTrack->SetLocalTime( aValue: fpPostStepPoint->GetLocalTime());
  fpTrack->SetProperTime( aValue: fpPostStepPoint->GetProperTime());
  //  energy, momentum, polarization
  fpTrack->SetMomentumDirection( aValue: fpPostStepPoint->GetMomentumDirection());
  if(fpTrack->GetParentID() == 0 && fpTrack->GetDefinition()->GetParticleName() == "proton"){
      fpTrack->SetKineticEnergy( aValue: 1.5*CLHEP::MeV);
  }
  else fpTrack->SetKineticEnergy( aValue: fpPostStepPoint->GetKineticEnergy());
  fpTrack->SetPolarization( aValue: fpPostStepPoint->GetPolarization());
  //  mass charge
```

Figure 31. Code for using protons with constant energy.

7. Visualization of spectrum of X-rays and protons

7.1. Spectrum from simulation results

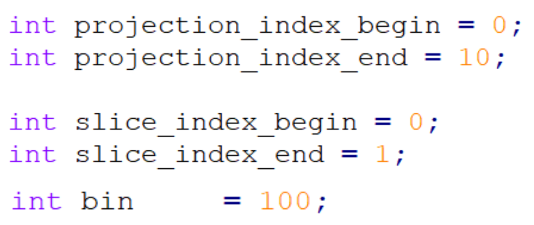
The following scripts are available to visualize the spectrum of protons and X-rays:

- ***Spectrum_proton.C***
- ***Spectrum_gamma.C***

These two scripts read simulation files: ***ProtonAtExit.dat*** for protons (STIM-T); ***GammaAtCreation.dat*** and ***GammaAtExit.dat*** for X-rays (PIXE-T). They plot a histogram of energy spectrum of protons or X-rays. The energy of the particles are ***float*** in keV. Users can choose to plot only a certain part of the data by specifying the following parameters:

- ***projection_index_begin, projection_index_end***: the first and last projection to be plotted
- ***slice_index_begin***, ***slice_index_end***: the first and last slice to be plotted
- ***bin***: the number of bins of the plotted histogram. The default value is 100.

For example, if these variables are set as shown in Figure 32, it means the particles from projection 0 to 10, slice 0 to 1 are visualized in the spectrum. Note that projection 10 and slice 1 are included.



```
int projection_index_begin = 0;
int projection_index_end = 10;

int slice_index_begin = 0;
int slice_index_end = 1;

int bin      = 100;
```

Figure 32. Parameters for visualization of the spectrum of X-rays.

7.2. Spectrum from input of tomographic reconstruction

The following scripts are available to visualize the spectrum of protons and X-rays:

- ***TomoSpectrum_HIST_proton.C*** for protons
- ***TomoSpectrum.C*** for X-rays
- ***TomoSpectrum_HIST.C*** for X-rays

These three scripts read the file for tomographic reconstruction: ***TomoSpectrum_HIST_proton.C*** reads ***StimEvent_std.DAT***, both ***TomoSpectrum.C*** and ***TomoSpectrum_HIST.C*** read ***PixeEvent_std.DAT.*** The energy of the particles are ***integer***.

After running each script, the spectrum data are written in a ***txt*** file containing the two columns:

- first column is the channel (energy in unit keV for STIM; 10 eV for PIXE)
- second column is the number of particles

Users can choose to plot only a certain part of the data by specifying the following parameters:

- ***projection_index_begin, projection_index_end***: the first and last projection to be plotted
- ***slice_index_begin***, ***slice_index_end***: the first and last slice to be plotted

They generate a spectrum of X-rays with 4096 channels for x axis and the number of event for y axis.

- For protons, each channel represents 1 keV, thus the maximal energy of X-rays is 4096*1 keV=4096 keV. In fact, when generating the input file for tomographic reconstruction (using ***BinToStd_ProtonAtExit.C***), the energy of proton is limited to less than 4095 keV. Therefore, there will no be energy overflow.
- For X-rays, each channel represents 10 eV, thus the maximal energy of X-rays is 4096*10eV=40.96 keV. In fact, when generating the input file for tomographic reconstruction (using ***BinToStd_GammaAtCreation.C*** or ***BinToStd_GammaAtExit.C***), the energy of X-rays is limited to less than 40.95 keV. Therefore, there will no be energy overflow.

The difference between the ***TomoSpectrum.C*** and ***TomoSpectrum_HIST.C*** lies in the generated spectra (Figure 33) after running the scripts. ***TomoSpectrum.C*** plots a graph (points), while ***TomoSpectrum_HIST.C*** plots a histogram (bars). ***TomoSpectrum_HIST_proton.C*** generates a histogram (bars).

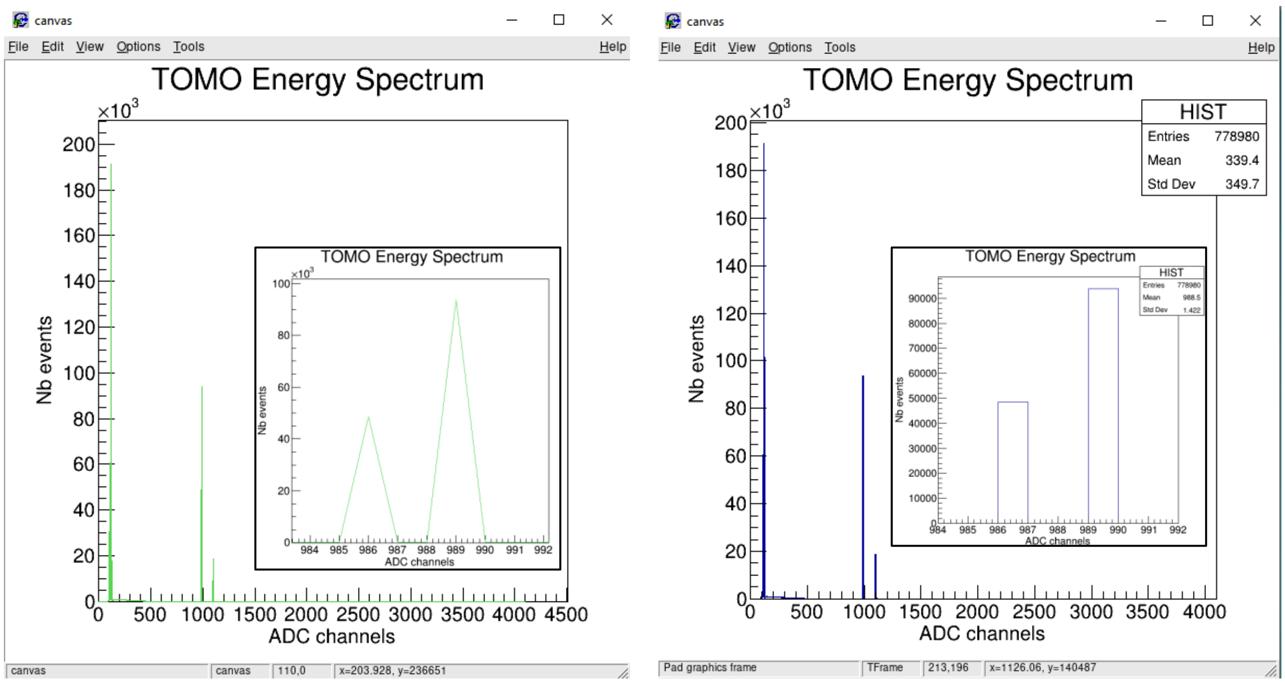


*Figure 33. Examples of the spectra: (a) generated by **TomoSpectrum.C**; (b) generated by **TomoSpectrum_HIST.C***

Type the following command to use the script:

```
root Spectrum_proton.C
```

Idem for ***Spectrum.gamma.C***, ***TomoSpectrum_HIST_proton.***C, ***TomoSpectrum.C***, and ***TomoSpectrum_HIST.C***.

8. Potential use for other applications

This example was initially developed for applications in tomographic imaging. However, it is also suited for “classical” (2D) STIM and PIXE imaging. Users just need to do the simulation with only one projection.

In addition, other types of particles can be used in the example, because Geant4 contains all physical processes for particles, including ions and X-rays in a wide energy range. Thus, users just need to modify the type of incident particles in the macro used to run the code.

9. List of scripts in the example

When doing a simulation, users may use the scripts in the following order:

- ***GPSPointLoop.C***: it generates a macro file to run the simulation.
- ***BinToStd_ProtonAtExit.C***: it reads the STIM-T simulation results and generates the data file for STIM-T reconstruction using selection with particle momentum.
- ***BinToStd_GammaAtCreation.C***: it reads the PIXE-T simulation results for X-rays at creation and generates the data file for PIXE-T reconstruction using selection with particle momentum.
- ***BinToStd_GammaAtExit.C***: it reads the PIXE-T simulation results for X-rays at exit and generates the data file for PIXE-T reconstruction using selection with particle momentum.
- ***BinToStd_proton_position.C***: it reads the STIM-T simulation results and generates the data file for STIM-T reconstruction using selection with particle position and momentum
- ***BinToStd_gamma_position.C***: it reads the PIXE-T simulation results for X-rays and generates the data file for PIXE-T reconstruction using selection with particle position and momentum
- ***LocateInterruption_ProtonAtExit.C***: in case of interruption, it locates the projection position of interruption for STIM-T simulation.
- ***LocateInterruption_GammaAtExit.C***: in case of interruption, it locates the projection position of interruption for PIXE-T simulation.
- ***Concatenate_BinToStd_ProtonAtExit.C***: in case of one interruption, it reads STIM-T simulation results and generates the data file for STIM-T reconstruction.
- ***Concatenate_BinToStd_GammaAtCreation.C***: in case of one interruption, it reads PIXE-T simulation results for X-rays at creation and generates the data file for PIXE-T reconstruction.
- ***Concatenate_BinToStd_GammaAtExit.C***: in case of one interruption, it reads PIXE-T simulation results for X-rays at exit and generates the data file for PIXE-T reconstruction.
- ***Spectrum_proton.C***: it visualizes the spectrum of protons and plots a histogram by reading simulation result ***ProtonAtExit.dat***.
- ***Spectrum_gamma.C***: it visualizes the spectrum of X-rays and plots a histogram by reading simulation result ***GammaAtCreation.dat*** or ***GammaAtExit.dat***.
- ***TomoSpectrum_HIST_proton.C***: it visualizes the spectrum of protons and plots a ***histogram*** by reading ***StimEvent*** data. It also writes the spectrum data in a ***txt*** file.
- ***TomoSpectrum.C***: it visualizes the spectrum of X-rays and plots a ***graph*** by reading ***PixeEvent*** data. It also writes the spectrum data in a ***txt*** file.
- ***TomoSpectrum_HIST.C***: it visualizes the spectrum of X-rays and plots a ***histogram*** by reading ***PixeEvent*** data. It also writes the spectrum data in a ***txt*** file.

<u>Scripts for specific use:</u>

- ***Extract_Projection.C***: it extracts 50 projections from a ***PixeEvent*** data file for tomographic reconstruction, which contains 100 projections. In fact, it extracts the projection 0, 2, 4, 6, 8…98 from projections 0-99. It eventually generates a new file with new index number of projections 0-49.
- ***Check_PixeEventFile.C***: it checks if the index of projections of a ***PixeEvent*** data file for tomographic reconstruction is correct. For example, if the user extract 50 projections from a data file composed 100 projections, it is necessary to make sure in the new data file, the index of projection starts from 0 and ends at 49.
- ***Extract_Slice.C***: it extracts a certain number of slice(s) from a ***PixeEvent*** data file for tomographic reconstruction. Users need to specify the first and the last slice to be extracted. Note that when writing a new data file, the index of slices will be initiated from 0.
- ***Concatenate_BinToStd_GammaAtCreation_fabricate.C***: if users make a PIXE-T simulation on a symmetrical object with only one projection, this script can be used to fabricate the other 99 projection data for X-rays at creation with same energy.
- ***Concatenate_BinToStd_GammaAtExit_fabricate.C***: if users make a PIXE-T simulation on a symmetrical object with only one projection, this script can be used to fabricate the other 99 projection data for X-rays at exit with same energy

<u>Scripts to generate voxelized phantoms:</u>

In order to compare the reconstructed tomographic images with original phantoms, it may be necessary to use a voxelized phantom. More information is available in the publication [7]:

- ***generate_voxelized_sphere_phantom.py***: it generates a voxelized phantom of an inertial confinement fusion target.
- ***generate_voxelized_worm_phantom.py***: it generates a voxelized phantom of the upper part of *C. elegans.*

References

[1] Michelet C, Barberet P, Moretto P, Seznec H. Development and applications of STIM-and PIXE-tomography: A review. Nucl Instrum Methods Phys Res B. 2015;363:55-60.

[2] Michelet C, Li Z, Yang W, Incerti S, Desbarats P, Giovannelli JF, et al. A Geant4 simulation for three-dimensional proton imaging of microscopic samples. Phys Med. 2019;65:172-80. https://doi.org/10.1016/j.ejmp.2019.08.022.

[3] Michelet C, Li Z, Jalenques H, Incerti S, Barberet P, Devès G, et al. A Geant4 simulation of X-ray emission for three-dimensional proton imaging of microscopic samples. Phys Med. 2022;94:85-93. https://doi.org/10.1016/j.ejmp.2021.12.002.

[4] Michelet C, Barberet P, Desbarats P, Giovannelli J-F, Schou C, Chebil I, et al. An implementation of the NiftyRec medical imaging library for PIXE-tomography reconstruction. Nucl Instrum Methods Phys Res B. 2017;404:131-9. https://doi.org/10.1016/j.nimb.2017.01.067.

[5] Lu H, He X, Meng J, Guo N, Rong C, Zhang W, et al. Reconstruction of Ge spatial distribution in ICF target using PIXE-T. Fusion Eng Des. 2016;113:43-50. https://doi.org/10.1016/j.fusengdes.2016.10.006.

[6] Guo N, Lu H, Wang Q, Meng J, Gao D, Zhang Y, et al. A dual-PIXE tomography setup for reconstruction of Germanium in ICF target. Nucl Instrum Methods Phys Res B. 2017;404:162-6. https://doi.org/10.1016/j.nimb.2017.04.041.

[7] Li Z, Incerti S, Beasley D, Shen H, Wang S, Seznec H, Michelet C. Accuracy of three-dimensional proton imaging of an inertial confinement fusion target assessed by Geant4 simulation. Nucl Instrum Methods Phys Res B. 2023; 536, 38-44. https://doi.org/10.1016/j.nimb.2022.12.026