ESIPAP European School of Instrumentation in Particle and Astroparticle Physics

2016 edition

Supervisers: Eric CHABERT, Eric CONTE

Computing session 4

Simulation of a simple tracker based on the GEANT4 package

Abstract:
This computing session is dedicated to detector simulation using the GEANT4 package. The aim of this session is to explain and to illustrate the main principles of GEANT4 simulation. The physics context is adapted from the GEANT4 example B2: a tracker made up of five gas chambers in the presence of a uniform magnetic field. The students will learn to build progressively a complete application based on GEANT4 which must describe the setup. Then ingoing particles will be generated and the detector response must be analyzed.

Pedagogical goals:

GEANT4 package

- Describing the geometry of a detector.
- Choosing properly the physics reference list.
- Simulating the response of a detector.
- Generating events and analyzing the simulation results.
- Respecting the programming conventions of the GEANT4 collaborations.
- Handling the user interactive console and the visualization.
- Using the official GEANT4 guides available on the website.

Compiling/linking

- Accessing information related to the GEANT4 installation.
- Configuring a Makefile for using the GEANT4 package.

Requirements:

- Inheritance from an abstract class and polymorphism.
- Attending the introduction ESIPAP course about detector simulation.

Contents

Part I

Introduction to the ESIPAP computing sessions

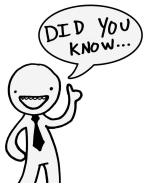
1 Foreword

Computing sessions belong to the educational program of the ESIPAP (European School in Instrumentation for Particle and Astroparticle Physics). Their goal is to teach the secrets of C++ programming through practical work in the context of high energy physics. The session is designed to be pedagogical. It is advised to read this document section-by-section. Indeed, except the *Physics context*, each section of the document is a milestone allowing to acquire computing skills and to validate them. The sections related to C++ programming are ranked in terms of complexity. In order to facilitate the reading of this document and to measure his progress, the student must **fill up the dedicated roadmap** which includes a check-list and empty fields for personal report.

In the document, some graphical tags are used for highlighting some particular points. The list of tags and their description are given below.

The student is invited to perform a pratical work by **writing a piece of code** following some instructions.

Analyzing or interpreting task is requested and the results must be reported in the roadmap.



Some **additional information** is provided for extending the main explanations. It is devoted to curious students.

A piece of **advice** is given to help the student in his task.

2 The ESIPAP framework

The practical works must be performed on devoted machines where all required software are properly installed. The user will find below all the instructions for setting the environment at each beginning of session.

2.1 Launching the `Windows` machine

You must choose a computer in the computing room, spot its name and check that no peripheral is missing (mouse, keyboard, ...). Then boot it and login to the `Windows` operator system (supervisors will provide the password access).

2.2 Accessing the `Linux` virtual machine

The practical sessions will be achieved on a `Linux` machine for pedagogical motivations. You must connect a virtual machine. First click on the "Start" button, i.e. the button with the `Windows` logo, located on the bottom left of the screen (see Figure 1).

Figure 1: The `Windows` Start button

According to Figure 2, click on the virtual machine called **"ESIPAP_slc6"**. A password could be necessary and should be supplied by the supervisors.

Figure 2: The screen showing the available virtual machines

2.3 Setting the environment

To load the work environment, you can issue the command below at the shell prompt.

```
bash$source␣/home/esipap/tools/setup.sh
```

If the system is properly installed, the version of each tool to study should be displayed at the screen like below. If you have an error, please call the supervisors.

```
-----------------------------------------------
              ESIPAP environment
-----------------------------------------------
 - GNU g++   version 4.9.1
 - ROOT      version 6.06/00
 - Geant4    version 10.2.0
-----------------------------------------------
```

You must work in your local folder. Of course, it is advised to create one folder for each practical session like: `session1`, `session2`, `session3` and `session4`. Do not overwrite or remove files that you wrote in a previous session.

2.4 Saving your work on a share disk

Your work will be evaluated from the the piece of code that you wrote. At the end of each session you must save your production on a share disk. The virtual machine is equipped with one share disk called "ESIPAP-SHARE" and saved everyday. For accessing this disk, click on the `Linux` tab named **"places"** according to Figure 3 and select the disk **"ESIPAP-SHARE"**.

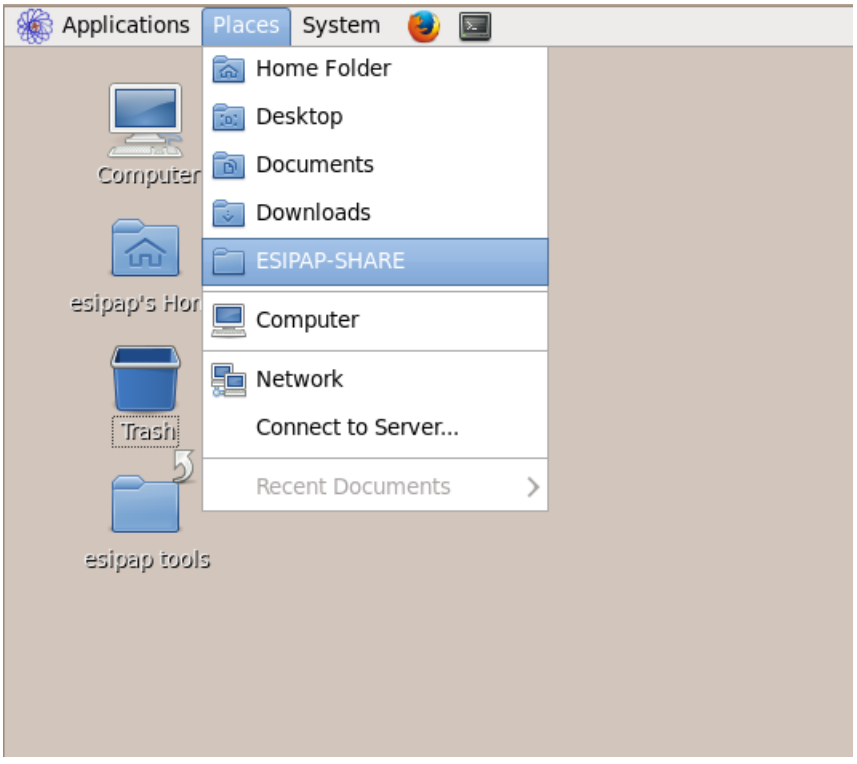


Figure 3: The ***Linux*** *tab named "places"*

After entering a password, the list of all connected machines in the room is displayed (see Figure 4). Select the folder corresponding to your machine and put there all you work. Please organize this folder by creating one folder for each practical session like: `session1`, `session2`, `session3` and `session4`.

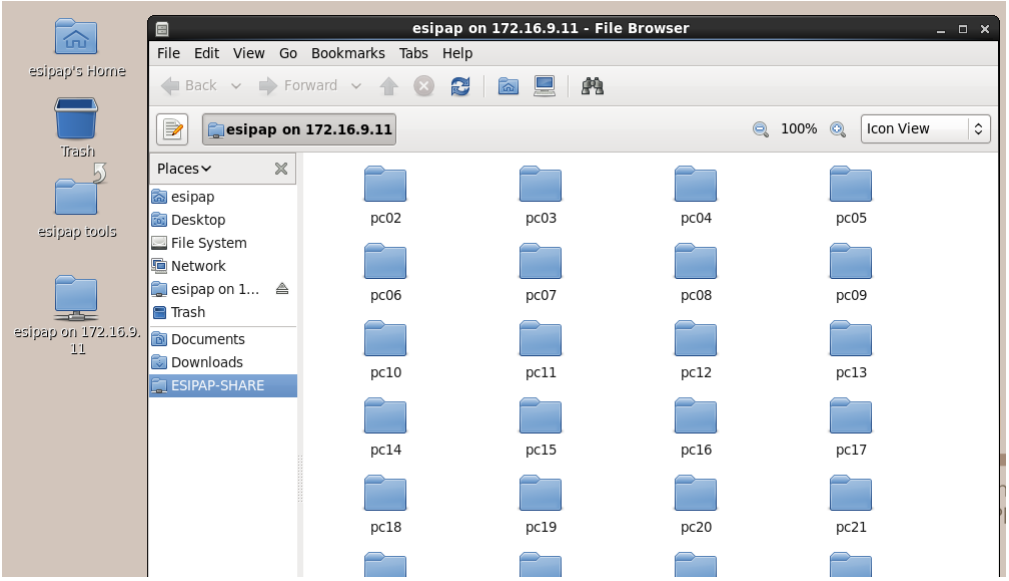


Figure 4: List of all available machines in the room

Part II

Getting started with GEANT4

3 First contact with GEANT4

In this section, the design of an application using the GEANT4 package will be discussed. The requirements will be enumerated and checked. Implementing and running an example program, very similar to the so-called "hello world!" example, will be the final step to reach.

3.1 Available online documentation

All developers of GEANT4 applications must use the updated documentation available on the official website of the package: `http://geant4.web.cern.ch` and students must to learn to use it as a tool. The main page of the user documentation enumerates the different kinds of support provided. This is the summarized list:

- **Guide for application developers**
- **Guide for toolkit developers**
- **Physics reference manual**
- **Courses and tutorials**
- **FAQ**: Frequently Asked Questions
- **doxygen documentation**
- **LXR code browser**: interactive viewing and searching facility for the Geant4 source code

In this documents explanations will refer as much as possible to this online documentation.

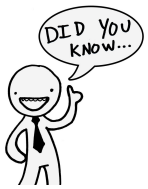
3.2 Programming conventions

This is a non-exhaustive list of recommendations for GEANT4 software developpers. In the context of the exercise, the students must respect as much as possible these conventions in their source files.

- One source file and one header file per class. Naming rules: class name + suffix (**.cc** or **.hh**)
- Usually GEANT4 class names begin with `G4`. User classes do not respect this convention.
- Start method names with an upper case letter. Use also upper case characters for following words. Example: `CollisionPoint()`.
- Start data member names with the character "f". User upper case characters for following words. Example: `fCollisionPoint`.
- Do not use single character names, except for loop indices.
- Protect each header file from multiple inclusion with:

```
#ifndef className_h
#define className_h
...
#endif
```

- Header files must not contain any implementation except for class templates and code to be inlined.
- Limit line length to 120 character positions.
- GEANT4 code uses independent-machine types such as `G4bool`, `G4double`, `G4int`, ... All these types are defined in the header file: `G4Types.hh`.
- GEANT4 code does not use all STL functions. Many of functionnalities have been implemented in the package such as `G4string`, `G4cout`, `G4cerr`, `G4Exception` ... Nonetheless `std::vector` of the STL is recommended.
- When you would like to introduce some data in your code, you must specify the units. Main units are defined in the header file `G4SystemOfUnits.hh`. When a value is introduced, it must be multiplied to the proper unit. Example: `size=15*km;`. If you would like to print a value with the good unit, you have to divide the value by the unit. Example: `G4cout « size/nm;`.



Headers of all GEANT4 basic objects such as units, types, G4STRING and `G4Exception` are gathered in only one header called `globals.hh`. Usually it is more convenient to include the header `globals.hh` in your source file instead of several specific headers.

- **Read carefully this extract of GEANT4 policy for developers. Your code must respect as much as possible these rules.**

3.3 Main program with an instance of `G4RunManager`

The main program will be contained in a source file called `main.cc`. A simple example respecting fully the `Geant4` programming rules is presented here. It displays at the screen the message *"Hello World!"* and the size of the Eiffel tower.

```
1 // Geant4 headers
2 #include "G4RunManager.hh"
3 #include "globals.hh"
4 #include "G4SystemOfUnits.hh"
5
6 // Main program
7 int main(int argc, char** argv)
8 {
```

```
9   // Construct the default run manager
10  G4RunManager* runManager = new G4RunManager;
11
12  // Display messages at screen with Geant4 streamers
13  G4cerr << "ERROR: hello world!" << G4endl;
14  G4double length = 324 * m;
15  G4cout << "INFO: Eiffel tower length = "
16         << length/m << " m" << G4endl;
17
18  // Free the memory
19  delete runManager;
20
21  // Normal program termination
22  return 0;
23 }
```

Listing 1: Hello World! with GEANT4

In this main program, a object of type `G4RunManager` is created. This class is the main core class of GEANT4 and manages all operations. During this creation, a header mentionning the release version of the package is automatically printed at the screen.

- **Recopy this example in a new file called `main.cc` in your working folder**

3.4 GEANT4 configuration

Before working with GEANT4, we will interest in its configuration and its installation on your machine. For LINUX/UNIX system, a lot of information can be obtained with program GEANT4-CONFIG. For launching this program and obtaining some help about its use, the user has to issue the following command line in the prompt shell:

```
bash$geant4-config␣--help
```

at any place of the disk (thank to the setup script). Among all the possible commands, the user can focus on these several points:

- accessing the GEANT4 version:

  ```
  bash$geant4-config␣--version
  ```

- options to supply to the compiler for building a C++ program with GEANT4:

  ```
  bash$geant4-config␣--cflags
  ```

- options to supply to the linker for building a C++ program with GEANT4:

  ```
  bash$geant4-config␣--libs
  ```

- check that all physics datasets are installed:

```
bash$geant4-config␣--check-datasets
```

- check that the openGL driver for visualization is enable:

```
bash$geant4-config␣--has-feature␣opengl-x11
```

- **Type and execute the above command lines using GEANT-CONFIG.**
- **Check you do not see any issues ou inconsistencies in the configuration**

3.5 Makefile with GEANT4

```
1  CC=g++
2  CFLAGS=$(shell␣geant4-config␣--cflags)
3  LDFLAGS=$(shell␣geant4-config␣--libs)
4  SRCS␣=␣$(wildcard␣*.cc)
5  HDRS␣=␣$(wildcard␣*.hh)
6  OBJS␣=␣$(SRCS:.cc=.o)
7  EXEC=myprog
8
9  all:␣$(EXEC)
10
11 $(EXEC):␣$(OBJS)
12 ␣␣$(CC)␣$(LDFLAGS)␣$(OBJS)␣-o␣$@
13
14 %.o:␣%.cc␣$(HDRS)
15 ␣␣$(CC)␣$(CFLAGS)␣-c␣$<␣-o␣$@
16
17 print:
18 ␣␣@echo␣"CFLAGS␣␣=␣"␣$(CFLAGS)
19 ␣␣@echo␣"LDFLAGS␣=␣"␣$(LDFLAGS)
20 ␣␣@echo␣"SRCS␣=␣"␣$(SRCS)
21 ␣␣@echo␣"HDRS␣=␣"␣$(HDRS)
22 ␣␣@echo␣"OBJS␣=␣"␣$(OBJS)
23 ␣␣@echo␣"EXEC␣=␣"␣$(EXEC)
24
25 clean:
26 ␣␣rm␣-f␣$(OBJS)␣$(EXEC)
```

Listing 2: Just a test

To compile the source files with this Makefile, just issue at the prompt:

```
bash$make
```

- **Recopy this makefile.**
- **Compile your program** `main.cc`**.**

To remove all objects produced during the compilation, the user can type:


```
bash$make␣clean
```

In the same spirit, it is possible to print at the screen the value of the Makefile internal variables by issuing:

```
bash$make␣print
```

3.6 Launch the program

The command line for launching the program is:

```
bash$./myprog
```

- **Execute the program and see if the program runs properly.**
- **Normally the header of GEANT4 should appear at the screen. Check that the release number is consistent with the one given by GEANT4-CONFIG**

4 Structure of a GEANT4 application

We focus now on the structure of an application based on the GEANT4 package. The minimal structure will be provided and the students are invited to analyze it and to use it.

4.1 Downloading the template

The minimal structure of a GEANT4 application is made up of several source files. A template of these files is given as a starting point of the study. First the user can download this tarball **G4Template.tgz** and untar it following the command:

```
bash$ tar xvzf G4Template.tgz
```

In the same way, the students must download another tarball called **VisuConfigFiles.tgz** containing configuration files required for the GEANT4 visualisation. The instructions are very similar for untaring the tarball:

```
bash$ tar xvzf VisuConfigFiles.tgz
```

- **Execute the instructions in order to have the files required for the following.**

4.2 A new main program

We present here a new main program which will replace the one introduced previously.

```
1  // Geant4 headers
2  #include "G4RunManager.hh"
3  #include "G4UImanager.hh"
4  #include "G4VisExecutive.hh"
5  #include "G4UIExecutive.hh"
6  #include "QGSP_BERT.hh"
7  #include "Randomize.hh"
8
9  // User headers
10 #include "DetectorConstruction.hh"
11 #include "ActionInitialization.hh"
12
13
14 // Main program
15 int main(int argc,char** argv)
16 {
17   // Choose the Random engine
18   G4Random::setTheEngine(new CLHEP::RanecuEngine);
19
20   // Construct the default run manager
21   G4RunManager * runManager = new G4RunManager;
22
```

```
23    // Set mandatory initialization: detector construction
24    runManager->SetUserInitialization(new DetectorConstruction());
25
26    // Set mandatory initialization: physics
27    G4VModularPhysicsList* physicsList = new QGSP_BERT;
28    runManager->SetUserInitialization(physicsList);
29
30    // Set mandatory initialization: action
31    runManager->SetUserInitialization(new ActionInitialization());
32
33    // Initialize G4 kernel
34    runManager->Initialize();
35
36    // Initialize visualization
37    G4VisManager* visManager = new G4VisExecutive;
38    visManager->Initialize();
39
40    // Get the pointer to the user Interface manager
41    G4UImanager* UImanager = G4UImanager::GetUIpointer();
42    UImanager->ApplyCommand("/control/execute init_vis.mac");
43
44    // interactive mode
45    G4UIExecutive* ui = new G4UIExecutive(argc, argv, "tcsh");
46    ui->SessionStart();
47
48    // Free the memory
49    delete ui;
50    delete visManager;
51    delete runManager;
52
53    // Normal program termination
54    return 0;
55 }
```

Listing 3: a complete GEANT4 *main program*

This source file needs some explanations:

- Line 18: a random generator is initialized. It will be used for generating random values.
- Before initializing the only instance of `G4RunManager`, three mandatory inputs are required:
  - a detector geometry describing by the class `DetectorConstruction` at Line 24. The related header is included at Line 10.
  - a physics reference list at Lines 27-28.
  - actions (such as generating particles) describing by the class `ActionInitialization` at Line 31. The related header is included at Line 11.
- Line 34: Initializing the only instance of `G4RunManager`.

- Creating a graphical view at Line 38 and initializing it at Line 42 with the configuration file called `init_vis.mac`.
- Lines 45-46: Creating an user interactive console.

More details can be found here.

- **Recopy this code and put it in the file `main.cc`.**
- **Check that the project compile properly.**

4.3 Analyzing the class `DetectorConstruction`

We would like to analyze the class `DetectorConstruction` described in the file `DetectorConstruction.cc` and `DetectorConstruction.hh`.

- **From which base class this class is derived?**
- **Is the base class is abstract? If yes, which methods must be absolutely defined in the derived class?**
- **Draw the UML diagram corresponding to the studied class by specifying also the methods and data members inherited.**
- **With the help of your GEANT4 courses, explain the content of the method `construct`.**

4.4 Analyzing the class `ActionInitialization`

We would like to analyze the class `ActionInitialization` described in the file `ActionInitialization.cc` and `ActionInitialization.hh`.

- **From which base class this class is derived?**
- **Is the base class is abstract? If yes, which methods must be absolutely defined in the derived class?**
- **Draw the UML diagram corresponding to the studied class by specifying also the methods and data members inherited.**

4.5 Analyzing the class `PrimaryGeneratorAction`

We would like to analyze the class `PrimaryGeneratorAction` described in the file `PrimaryGeneratorAction.cc` and `PrimaryGeneratorAction.hh`.

- **From which base class this class is derived?**
- **Is the base class is abstract? If yes, which methods must be absolutely defined in the derived class?**
- **Draw the UML diagram corresponding to the studied class by specifying also the methods and data members inherited.**
- **With the help of your GEANT4 courses, explain the content of the construct of the class `PrimaryGeneratorAction`.**

4.6 Launching the program

The command line for launching the program is:

```
1 bash$./prog
```

If the program runs properly, an initialization sequence is launched. At the end of the sequence, a user console with a prompt `Idle>` is opened and a OpenGL viewer is created, displaying the detector geometry.

- **Execute the program and see if the program runs properly.**

4.7 Handling the interactive console

The interactive console allows the user to change inline the GEANT4 configuration, to change the visualization settings and to launch GEANT4 processing. The syntax of the commands is very simple.

- Some short instructions allow to do special actions. The most important one are: `quit` for exiting the program and `help` for opening the inline help menu.
- The different options or actions of GEANT4 are represented as executable files sorted in folder tree. To set an option, you have the possibility to execute the command from the root. Example: `/run/verbose 3`.
- For displaying the current value of an option, the command must be preceded by the '?' character. For instance: `?/run/verbose`.
- It is possible to browser the folder tree by command `ls` and `cd`.
- The tab completion is very useful!

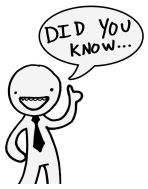
Here we would like to describe some useful commands related to the OpenGl viewer:

- **camera rotation:** `/vis/view/set/viewpointThetaPhi`
  example: `/vis/viewer/set/viewpointThetaPhi 0.  0.`
  example: `/vis/viewer/set/viewpointThetaPhi 90.  180.`
- **zoom:** `/vis/viewer`
  example: `/vis/viewer/zoom 1.4`
- **add/change axes**: `/vis/viewer/add/axes`
  example: `/vis/viewer/add/axes 0 0 0 0.5 m`
  for 0.5m-length axis centered in (0,0,0)
- **refresh the view**: `/vis/viewer/refresh`

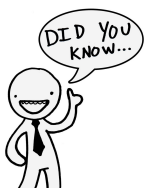
Finally the following command allows to launch one run of `n` events:

```
/run/beamOn 10
```

- **Pratice the interactive console, especially learn to use the commands handling the OpenGL viewer.**



All the commands you type can be gathered in a script file (called for instance `myscript.mac`). To execute the script from the console just issue: `/control/execute myscript.mac`



The default configuration for the OpenGL viewer is loaded with the file `vis_gui.mac`. It is possible to tune this file.

Part III
A simple simulation of a tracker

5 Physics context

The physics context is adapted from the example B2 of the GEANT4 package. The application to design must simulate a simplified fixed target experiment. The setup consists of a target in lead followed by five chambers of increasing transverse size at defined instances from the target. These chambers have a cylindrical shape and are filled by xenon. They are located in a air-filled region called the tracker region. In addition, a global, uniform, and transverse magnetic field is applied: (0.2T,0,0). The position and the size of the different item are shown in the figure below.

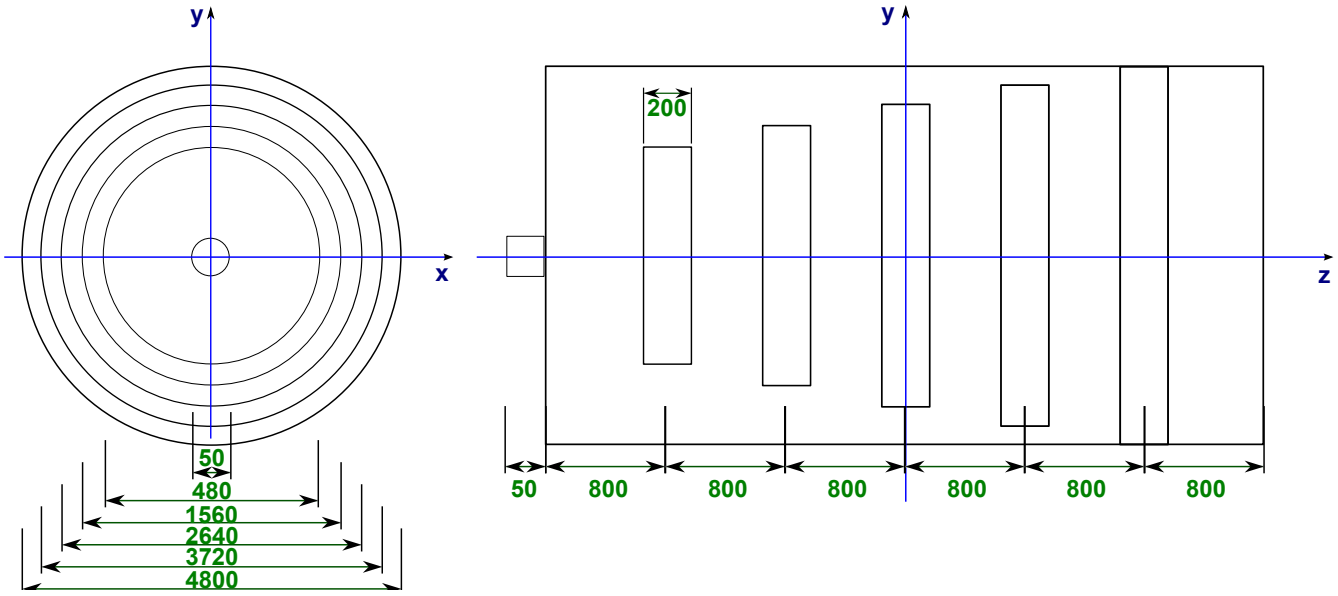


Figure 5: Layout of the experimental setup [size in millimeter]

The particle's type and the physic processes are set in the FTFP_BERT physics list. This physics list requires data files for electromagnetic and hadronic processes, *i.e.* the datasets G4LEDATA, G4LEVELGAMMADATA and G4SAIDXSDATA are mandatory.

The primary kinematics consists of a single particle which hits the target perpendicular to the entrance face. Several particle kind must be investigated: electron, proton and pions. The energy of these particles will be fixed to the value of 3 GeV.

Each chamber will be considered as a sensitive detector and particle hits in its matter will be collecd. Histograms will be achieved in order to display some distributions or to compute some efficiencies.

6 Detector construction

The design of the application is based on the structure introduced in the previous section. We work first on the detector geometry, so on the code on the files `DetectorConstruction.cc` and `DetectorConstruction.hh`.

6.1 Material

According to the physics topics, three types of material must be implemented: Air, Pb and Xe. There are several to define this matter ; we will investigate two of them:

- **First Method:** it is possible to define Pb and Xe elements by specifying their Z, their A and their density. For the air, we consider a mixture of O and N elements (30%-70%) with a density of 1.290 mg/cm^3. More details and explanations could be found here and here.

Element	Z	A	density
Pb	82	207.22 g/mole	11.350 g/cm^3
Xe	54	131.29 g/mole	5.485 mg/cm^3
N	7	14.01 g/mole	
O	8	16.00 g/mole	

- **Second Method:** use the GEANT4 material database based on the NIST (National Institute of Standards and Technology) work. A NIST manager must be implemented. List of pure materials and list of compounds can be found here in the GEANT4 documentation appendices.

The material must be defined at the beginning of the function `Construct`. To print the description of a `G4Material` or a `G4Element`, it is possible to use the traditionnal GEANT4 streamer:

```
G4Material *air = [...]
G4cout << *air << G4endl;
```

- **Implement the three materials according to both methods.**
- **Test and compare the implementations by printing them at screen.**

The list of the material defined in your application can be displayed by only one command line:

```
G4cout << *(G4Material::GetMaterialTable())
       << G4endl;
```

6.2 Detector geometry

- **Implement in the function `Construct` the geometry shown in the physics context.**
- **Check there is no overlaps between the different item**

6.3 Geometry tolerance

The tolerance value defining the accuracy of tracking on the surfaces is by default set to a reasonably small value of 0.001 nm. Such accuracy may be too abusive for simulations of big detectors. That's why it is possible to specify the surface tolerance to be relative to the extent of the world volume defined for containing the geometry setup.

To compute in this way and display the geometry tolerance, the piece of code below must be implemented. It is assumed that the size of the item `world` is contained in the variable `worldLength`. **Becareful: the implementation must be done before all volume definition and must be done only one time.**

```
G4GeometryManager::GetInstance()
             ->SetWorldMaximumExtent(worldLength);
G4cout << "Computed tolerance = "
       << G4GeometryTolerance::GetInstance()
            ->GetSurfaceTolerance()/nm
       << " nm" << G4endl;
```

- **Set a geometry tolerance adapted to the application geometry.**

6.4 Colour attributes for visualization

From the OpenGL view, all the detector items appear in white colour. It is possible to assign colours to each item in order to highlight it. In GEANT4 the colour is described by a class called `G4Colour`. The list of colours and the way to design new ones could be found here.

The colour assignment can be done after the implementation of the detector logical volume. The following lines allow to set in white colour the shape of the item `world`.

```
G4VisAttributes* worldAtt= new G4VisAttributes(G4Colour(1.0,1.0,1.0));
worldLV ->SetVisAttributes(worldAtt);
```

Of course corresponding headers need to be included at the beginning of the source file.

```
#include "G4VisAttributes.hh"
#include "G4Colour.hh"
```

- **Assign a different colour for each component of the setup: world, tracker and chamber**

6.5 Magnetic field

The way to introduce a uniform magnetic field is discussed here. First the following lines allow to include the required header files:

```
#include "G4UniformMagField.hh"
#include "G4TransportationManager.hh"
#include "G4FieldManager.hh"
```

The magnetic field is described by the class `G4UniformMagfield`. An instance of this class can be declared at any place of the function `Construct` by the following line:

```
G4UniformMagField* magField = new G4UniformMagField(
                      G4ThreeVector(0.2*tesla,0.,0.));
```

To take into account the magnetic field in the tracking step, the following piece of code must be implemented:

```
G4FieldManager* fieldMgr =
  G4TransportationManager::GetTransportationManager()
    ->GetFieldManager();
fieldMgr->SetDetectorField(magField);
fieldMgr->CreateChordFinder(magField);
```

More details can be found here.

- **Implement the magnetic field in the `construct` method.**

7 Physics specifications

Concerning the physics input, a generator of particles must be initialized and a the physics modeling the particle interaction must be precised. Both topics will be treated in this section.

7.1 Particle sources

We would like to generate only one particle an event. A particle gun generator (described by the class `G4ParticleGun`) is the best generator for this goal. The particle must be generated from the interface between the tracker and the target, centered in the transverse plane. The direction is along the z-axis.

Concerning the particle identification, the default identification should be a proton. We would like to specify the particle identification as an argument of the executable program `prog`. The argument will be a string whose the allowed values are "e+", "e-", "proton", "antiproton", "pi+" or "pi-". Launching the program with a "e+" configuration should require to execute the following instruction:

```
bash$ ./prog␣e+
```

- **Adapt the piece of code in the source file `PrimaryGeneratorAction.cc`.**
- **Adapt the piece of code in the source file `PrimaryGeneratorAction.cc` for implementing the requested particle sources.**
- **Implement a mutator function of the class `PrimaryGeneratorAction` with the following prototype `void SetParticleId(G4string name)` for changing the particle identification of the particle gun generator.**
- **Retrieve the particle identification by reading the arguments of the main program (if arguments are supplied) and propagate this setting to the class `PrimaryGeneratorAction`.**

7.2 Reference physics list

The particle interactions are described in a physics list which is highly dependent on the use case. GEANT4 provides several reference physics lists which are routinely validated and updated with each release. Their definition can be found here and their application here.

For the targeted application, the list called `FTFP_BERT` is suggested.

- Looking through the documentation, explain in few words the content of the physics list `FTFP_BERT`
- Motivate the choice of the physics list.

- In the `main.cc` program, the physics list used in the G4RunManager initialization is not the correct one. Adapt the code in order to take into account the `FTFP_BERT` list

7.3 Studying few events

- Compile and execute the program with different generated particles: e-, e+, pi+, pi-, proton and antiproton.
- For each configuration, generate an event and save the graphical view in the longitudinal plane

- Describe that you observe in the graphical view.
- Compare the views obtained with the different configurations.

8 Detector response

Implementing the response of the detector is performed in two step. First it is necessary to define the sensitive volumes and to describe the "measurement". The second step consists in defining the "measures". In the case of a tracker apparatus, the "measures" take the form of a collection of hits. Unfortunately, due to a lack to time, the students will not learn to program by themselves these two steps ; the required C++ classes will be provided.

8.1 Package to download

An example of source files required for describing the detector response is supplied. These files are gathered into a tarball which can be downloaded and uncompressed:

```
bash$ cp econte/public/ESIPAP/TP4/G4Hit.tgz ./
bash$ tar xvzf G4Hit.tgz
```

8.2 Analyzing the class `TrackHit`

In `Geant4` a hit is a snapshot of the physical interaction of a track (or an accumulation of interactions of tracks) in the sensitive region of the detector. The user must implement himself the class describing a hit and must specify the various types information to store (position time, energy deposit, ...). In the downloaded package, a such class is alreay created: `TrackerHit` implemented in the files `TrackerHit.cc` and `TrackerHit.hh`.

- **From which base class this class is derived?**
- **Is the base class is abstract? If yes, which methods must be absolutely defined in the derived class?**
- **Draw the UML diagram corresponding to the studied class by specifying also the methods and data members inherited.**
- **Try to explain the goals of each method.**

8.3 Analyzing the class `TrackerSD`

A sensitive detector creates hit(s) using the information given from a `G4Step` object. The user has to provide his/her own implementation of the detector response. The hits created will be stored them into a `HitsCollection` object. In the downloaded package, a such class is already created: `TrackerSD` implemented in the files `TrackerSD.cc` and `TrackerSD.hh`.

- **From which base class this class is derived?**
- **Is the base class is abstract? If yes, which methods must be absolutely defined in the derived class?**
- **Draw the UML diagram corresponding to the studied class by specifying also the methods and data members inherited.**
- **Try to explain the goals of each method.**

8.4 Implementing sensitive detectors

It is necessary to assign a TrackerSD object to the interested logical volume. To reach this aim, the class `DetectorConstruction` must be modified. First the header file corresponding to the class `TrackerSD` should be included.

```
1 #include "TrackerSD.hh"
```

Then an instance of the class `TrackerSD` should be created after the definition of the different volumes. Becareful: a specific name should be precised to the sensitive volume and to the hit collection produced. These names will be used in the following. The line below is an example of a such definition:

```
1 TrackerSD* trackerSD =
2    new TrackerSD("TrackerSD","TrackerHitsCollection");
```

The assignment of `TrackerSD` object to a logical volume is performed by an inherited method of the class `DetectorConstruction` called `SetSensitiveDetector`. The corresponding line of code is the following:

```
1 SetSensitiveDetector("ChambersLV", trackerSD, true);
```

In this example it is assumed that all interested volumes (the Xenon chambers) have the same logical volume name `"ChambersLV"`.

- **Implement these pieces of code.**
- **Compile the program and launch it.**
- **Generate some events and visualize the hits in the OpenGL view.**

9 Analyzing GEANT4 output

A this step, the description and the simulation of the experimental setup are achieved. Events can be generated and collection of hits are produced. Now we have to learn how to analyze them. To fulfill this goal, the program structure must be enriched by new classes.

9.1 Implementing *user action* in `UserEventAction` class

The first class, that we call `UserEventAction`, must inherit from the GEANT4 class `G4UserEventAction`. The implementation of this class will be contained in the files `UserEventAction.cc` and `UserEventAction.hh`. The two relevant virtual functions are `BeginOfEventAction` and `EndOfEventAction`. The first function is launched before processing each event and the second function is launched after processing each event.

- **Find the description of the class `G4UserEventAction` in the official documentation of GEANT4**
- **Implementing the class `UserEventAction` which publicly inherits from `G4UserEventAction`. Only the no-argument constructor, the destructor, `BeginOfEventAction` and `EndOfEventAction` must be implemented. The content of these functions will remain empty for the moment.**
- **Perform a preliminary check by just compiling the program.**

9.2 Implementing *user action* in `UserRunAction` class

The first class, that we call `UserRunAction`, must inherit from the GEANT4 class `G4UserRunAction`. The implementation of this class will be contained in the files `UserRunAction.cc` and `UserRunAction.hh`. The two relevant virtual functions are `BeginOfRunAction` and `EndOfRunAction`. The first function is launched before processing each run and the second function is launched after processing each run.

- **Find the description of the class `G4UserRunAction` in the official documentation of GEANT4**
- **Implementing the class `UserRunAction` which publicly inherits from `G4UserRunAction`. Only the no-argument constructor, the destructor, `BeginOfRunAction` and `EndOfRunAction` must be implemented. The content of these functions will remain empty for the moment.**
- **Perform a preliminary check by just compiling the program.**

9.3 Adding the *user action* classes into the GEANT4 processing

Instances of the new classes USEREVENTACTION and USERRUNACTION must be added in the class `PrimaryGeneratorAction`. After including the proper header files, the functions `BuildForMaster` and `Build` must be modified in that way:

```
void ActionInitialization::BuildForMaster() const
{
  SetUserAction(new RunAction);
}

void ActionInitialization::Build() const
{
  SetUserAction(new PrimaryGeneratorAction);
  SetUserAction(new RunAction);
  SetUserAction(new EventAction);
}
```

- **Do the changes above in the `ActionInitialization` implementation**
- **Compile and execute the program. No change in the `Geant4` behaviour is expected.**
- **Adding some printed messages in functions `BeginOfRunAction`, `EndOfRunAction`, `BeginOfEventAction` and `EndOfEventAction`. Compile and execute the program. Launch a run of several events and see your messages appear on the screen. Check in which order they appear.**

9.4 Accessing output

We would like to read through the collection of hits produced at the end of each events. Therefore the corresponding piece of code must be implemented in the function `EndOfEventAction` of the class `UserEventAction`.

Preliminary, it is necessary to include the headers at the beginning of the source file `UserEventAction.cc`.

```
#include "G4HCofThisEvent.hh"
#include "G4SDManager.hh"
#include "TrackerHit.hh"
```

In the function `EndOfEventAction`, you need to get the identification number corresponding to the collection you would like to retrieve. Becareful to the name of hits collection! Getting an identification integer to the collection:

```
G4SDManager* sdManager = G4SDManager::GetSDMpointer();
G4int collId = sdManager->
```

```
  GetCollectionID("TrackerSD/TrackerHitsCollection");
```

Getting a pointer to the list of collections produced during the event can performed by a method of the class `G4Event` called `GetHCofThisEvent()`. The following piece of code shows how to access it.

```
G4HCofThisEvent* hce = event->GetHCofThisEvent();
if (hce==0)
{
  G4cerr << "No Hit collections are found" << G4endl;
  return;
}
```

Getting a pointer to the collection of hits can be performed in the following way:

```
TrackerHitsCollection* hc = static_cast<TrackerHitsCollection*>
  (hce->GetHC(collId));
```

Browsing through the hits collection and accessing the hits data are illustrating by the following piece of code.

```
for (G4int i=0;i<hc->entries();i++)
{
  TrackerHit* hit = (*hc)[i];
  G4ThreeVector position = hit->GetPos();
  G4cout << position << G4endl;
}
```

- **For each event, print on the screen the number of hits collected.**
- **For each event and for each chamber, print the position of the first hit inside the given chamber and also the energy deposit.**