Physics Simulation with Geant4

Fortgeschrittenenpraktikum (FOPRA)

T. Le Bleis, P. Klenze

November 25, 2019

Note: You will always find the latest version at http://users.ph.tum.de/pklenze/fopra/
# Contents

## 1 Introduction

- **1.1** Calorimetry physics
  - 1.1.1 Interaction of particles with matter
  - 1.1.2 Scintillators
- **1.2** The CSI(Tl) calorimeter
  - 1.2.1 Scintillator-based calorimeters
  - 1.2.2 Detection of visible photons
  - 1.2.3 Detector Readout
  - 1.2.4 Online monitoring
  - 1.2.5 Offline analysis
- **1.3** Particularites of calorimetry
  - 1.3.1 Compton scattering in calorimeters
  - 1.3.2 Efficiency
- **1.4** Correlation plots
  - 1.4.1 Resolution
  - 1.4.2 Doppler effect
- **1.5** Simulation of particle detectors
  - 1.5.1 Why simulate detectors?
- **1.6** C++
  - 1.6.1 Formatting
  - 1.6.2 Variables
  - 1.6.3 For loops
- **1.7** GEANT4
  - 1.7.1 Usercode
  - 1.7.2 Material
  - 1.7.3 Geometry
  - 1.7.4 Physics list
  - 1.7.5 Event generator
  - 1.7.6 Generation of primary particles
  - 1.7.7 Visualisation
- **1.8** Analysis
  - 1.8.1 With ROOT
  - 1.8.2 With PyROOT
2 Experiment

2.1 General procedure

2.1.1 Detectors

2.1.2 Source

2.1.3 Side measurements

2.2 Setups

2.2.1 Setup 0: Calibration

2.2.2 Setup 1: Closest distance to large box

2.2.3 Setup 2: Closest distance to small box

2.2.4 Setup 3: $d_{\text{big,small}} = 5 \text{ cm}$

2.2.5 Setup 4: $d_{\text{big,small}} = 10 \text{ cm}$

2.3 Experiment instructions

2.3.1 DAQ howto

3 Simulation

3.1 Reproduction of Experiment

3.1.1 First Simulation

3.1.2 Simulate experimental setup

3.2 Development of a detector system

3.2.1 Detect a $\gamma$ emitted by a moving ion

4 Analysis

4.1 Calibrated spectra

4.2 Comparing experiment and simulation

4.3 Resolution

4.4 Effective distances and photopeak efficiencies

4.5 Detector material impact

4.6 Moving Ion setups
Chapter 1

Introduction

A simulation is a computer program that intends to reproduce the results of a possibly observable process by modelling its individual component and behaviour. Simulations are used in a large variety of fields: from economy to biology, from sociology to physics. The process is always the same: to make an experiment on a computer in order to predict or understand some situation.

1.1 Calorimetry physics

In order to follow the explanations later, it is needed to get some notes about some physics process.

1.1.1 Interaction of particles with matter

When a particle reaches a piece of matter, it will interact with the constituents of the matter, that is to say its atoms (electrons and nuclei). Depending on different factors (distance from the centre, energy, charge, etc.), different interactions can occur. Generally, two main effects can happen: the particle can be absorbed or scattered. Both effects involve a loss of energy.

From a macroscopic point of view, the particle may be stopped (hence the stopping power of some materials), or slightly deflected (straggling) while pursuing its course, or even strongly deflected (e.g. Compton scattering). Furthermore, the particle may not interact, interact once, or even interact more than once while passing through the matter.

However, as the exact position of the atoms in the material is not known, and the interactions described by quantum mechanics can only define probabilities of outcome, it is not possible to predict a priori what will happen for a given incoming particle impinging on the matter. The interaction of a particle with matter is then a highly statistical process. If this interaction is reproduced for a statistically significant number of incoming particle, the different global characteristics of the interaction can be observed.

For photons, there are are three main interaction processes with matter: the photoelectric effect, where a bound electron is excited to a state of higher energy, Compton scattering, in

\footnote{for us, that includes photons}
photon absorption in lead

Figure 1.1: The photon absorption coefficient for a sample material (here: lead) at different photon energies. Data Source: [14]

which a free (or weakly bound) electron collides with the photon and *pair production*, which means the photon produces an electron positron pair.

Each of these processes is strongly dependent on the specific material. Typically, the photoelectric effect is dominant at small energies in the keV range, Compton scattering dominates in the range from a few hundreds of keV to a few MeV, and governed by pair production above a few MeV.

Any *charged particle* – including Compton-scattered electrons – which passes through matter will collide with the hull electrons of the matter, ionizing them and thus losing parts of its energy. This process is described by the *Bethe-Bloch-Equation*.

For light charged particles such as *electrons*, at energies above roughly $600\frac{\text{MeV}}{Z}$, another process dominates the energy loss. The charge of the nuclei in the matter form local electric and magnetic fields inside the atoms. Any particle travelling through such fields will emit photons in a process called Bremsstrahlung. The energy loss due to Bremsstrahlung in magnetic fields at a given energy is proportional to $1$ over the particle mass to the fourth power, so this effect

\[ \frac{\mu}{\rho} \text{ photon absorption coefficient in } g/cm^2 \]

\[ 10^3 \]

\[ 10^2 \]

\[ 10^1 \]

\[ 10^0 \]

\[ 10^{-1} \]

\[ 10^{-2} \]

\[ \text{photon energy } E \text{ in MeV} \]

\[ -3 \times 10^{-2} \]

\[ -2 \times 10^{-1} \]

\[ 1 \times 10^0 \]

\[ 2 \times 10^1 \]

\[ 3 \times 10^2 \]

\[ 1 \times 10^3 \]

\[ \text{compton} \]

\[ \text{photoelectric} \]

\[ \text{pair production} \]

\[ \text{total} \]
electron stopping power of lead

Figure 1.2: The energy loss of electrons in lead at different energies. For our purposes, collisions mean ionization and radiative losses mean Bremsstrahlung. Data source: [15]

affects lighter particles at much lower energies.

1.1.2 Scintillators

Scintillators are materials in which ionizing radiation can excite certain states in the material. When these deexcite, photons (typically in the visible or UV band) are emitted. There are various types of scintillators available, among them organic compounds and anorganic crystals such as CaWO$_4$ or CsI(Tl)\(^4\) – the latter of which will be used in the experiment.

Picking the right scintillator for a calorimeter is by no means an easy task: among others, one has to consider the density of the material (which greatly affects the stopping power), the light yield (the number of photons emitted per keV of deposited energy), the decay time of the excited state(s), the processing options for the material (plastics can be molded to any form, while some anorganic crystals are quite brittle), possible hygroscopy (for many inorganic crystals), output wave length and, of course, cost.

---

\(^4\)Read that as “cesium iodide doped with thallium”.

which accelerated electrons and positrons in the same ring tunnel.
1.2 The CSI(Tl) calorimeter

1.2.1 Scintillator-based calorimeters

When an incoming particle interacts with the scintillator, it can produce secondary particles, possibly forming a cascade of particles with successively less energy, which is called an electromagnetic shower. If the whole cascade is contained within the scintillator, all of the kinetic energy of the incoming particle is absorbed by the scintillator and the number of visible photons which will be emitted will be proportional to that energy.

1.2.2 Detection of visible photons

The photons are then detected with a suitable detector, such as a photomultiplier tube (PMT) or – in the case of this experiment – an avalanche photodiode (APD). In a nutshell, the signal output from these devices serves as a measurement of the energy of the incoming particle. The APD current is proportional to the photon flux, and thus the integrated current – the charge – is proportional to the total number of photons emitted by the CsI(Tl) – which in turn is proportional to the energy deposited in the crystal.

1.2.3 Detector Readout

As discussed above, a detected particle deposits some energy in a crystal, and that energy is converted to light by the scintillation process. Every crystal is connected to an avalanche photo diodes (APD) which generates a current signal when it is hit by light. The APDs are connected to the preamplifier, which serves two functions: it generates the high voltage for the APD bias, and it amplifies (and integrates) the current signal from the APD. The differential output signals are connected to a signal processing board called FEBEX. On the board, the analogue signal is first sampled with analogue digital converters (ADCs). Then, an onboard

\[ \text{Figure 1.3: The readout scheme for the FEBEX system.} \]

---

5 Or, for a photon, the total energy.

6 In electronics, differential signaling means that you have transmit your signal through two lines which are complementary to each other. The actual signal is in the difference between both lines. Unlike for a single line, any noise induced on both lines will cancel out.
field programmable gate array (FPGA) does a pulse shape analysis (PSA) on the digital signal to get an energy estimate.

As shown in figure 1.4 the signal is split into two branches for the PSA: a fast branch for trigger generation and a slow branch for energy reconstruction.

In the fast branch, the signal is first shaped by averaging about a few samples to smooth it. The discriminator compares the slope of the resulting signal with a predefined threshold and emits a trigger signal if it is reached. In analogue electronics, the smoothing (lowpass filter) and differentiation (high-pass filter) would typically be combined into a single bandpass filter, which would then be fed into a discriminator.

In the slow branch, the signal is first written into a FIFO\(^7\) to delay it so that the pulse will still be there when the fast branch raises a trigger. With analogue electronics, this might be implemented using a delay box, which basically contains a long cable through which the signal travels. Afterwards, the signal is again smoothed by a shaper. Afterwards, the baseline – the signal voltage when there is no event – is subtracted from the signal. In the analogue world, this would correspond to a pole-zero cancellation filter. The preamplifier does not store the charge forever, but is build to let the charge slowly decline over time. To compensate that effect, a technique called moving window deconvolution (MWD) is used. This step can not be done with analogue electronics and is one of the reasons to use a digital system. After these steps, the pulse height corresponds to the energy deposited in the crystal. Additional pulse form analysis can be done to identify different types of ionizing radiation (QPID).

The Data Acquisition (DAQ) is controlled by a software called Multi-Branch System (MBS\(^10\)). When it receives a trigger\(^8\) it reads the energy estimates for the channels from the FEBEX boards makes them available on the network.

---

\(^7\) First In First Out

\(^8\) The trigger is also generated by the FEBEX board, but the details are beyond the scope of these instructions.
1.2.4 Online monitoring

Go4 [11] is a C++-based software developed at GSI. It can connect to the MBS stream server, but may as well read different types of files (particularly the LMD files written by MBS). It can be used for both on- and off-line analysis. In the current experiment, however, it will be used only to monitor the on-going reaction online and record histograms.

1.2.5 Offline analysis

Generally, the offline analysis starts by the unpacking phase which consists in extracting the relevant data from the files written during the acquisition. The unpacker extracts the data from the file and store the important information into a ROOT [8] tree. In the present case, the unpacking is done directly by Go4 and the relevant histograms are directly produced. The ROOT interpreter will then be used to produce the relevant spectra and calculate the efficiency and resolution of the $\gamma$ detection, the effect of Compton scattering using the more complex setups. See section 1.8 for more details.

1.3 Particularites of calorimetry

1.3.1 Compton scattering in calorimeters

As discussed previously, Compton scattering is the elastic scattering of a photon with a free electron. In this interaction, the photon loses a part of its energy to the electron and is scattered at a different angle [1]. The electron will then in turn lose its energy due to ionization and Bremsstrahlung.

If the scattered photon leaves the crystal, its full energy will not be reconstructed. The highest energy that can be detected in this case – the maximum possible energy transfer to the electron – is called the Compton edge [3]. At energies around 1 MeV, electrons are absorbed much quicker than photons, so the electron will typically not escape the detector.

If there are other scintillators, it is possible that the escaping $\gamma$ is detected in a neighbouring crystal. The total energy of the original photon can then be reconstructed by the sum of the energy detected in the first crystal and the energy detected in the second crystal. This is the so-called add-back method, for which the energies from all the crystals are added to reproduce the original energy distribution (it is a very important technique in $\gamma$ calorimetry).

In the present exercise, the effect of the Compton scattering can be observed in the experimental measurement as well as its simulation. In the development of a detector, the add-back technique will be important to reconstruct the proper energy spectrum.

1.3.2 Efficiency

To characterise any detector, two quantities are very important: its efficiency and its resolution. For a given photo peak, the total photopeak efficiency $\epsilon_{tot}$ is the number of counts in the photopeak $N_{detected}(E)$ divided by the number of photons with that energy the source emitted $N_{emitted}(E)$. This efficiency is a product of two factors – a geometrical factor $\epsilon_{geo}$ describing
the chance that a photon will enter the detector, and a detector specific factor $\epsilon_{det}(E)$ giving the probability that the detector will absorb all of the particles energy.

$$
\epsilon_{tot}(E) = \frac{N_{detected}(E)}{N_{emitted}(E)} = \epsilon_{geo}\epsilon_{det}(E)
$$

(1.1)

Geometric efficiency

The geometric efficiency is just the solid angle of the detector, as seen from the source, divided by the solid angle of the whole sphere, which is, of course, $4\pi$. It can be thought of as the chance that a particle emitted from the source in an isotropically random direction will hit the detector at all.

$$
\epsilon_{geo} := \Omega \approx \frac{A}{4\pi r^2}
$$

(1.2)

Unfortunately, our crystals are not shaped so that their area $A$ increases proportional to $r^2$, so picking the correct value for $r$ is paramount. Using only physical distance (red shape) for the radius will count photons which are just touching the edge of the crystal, which do not really have a chance to interact. A better choice would be $\tilde{r} := r + \ell_{int}$ (blue outline), where $\ell_{int}$ is of the order of the effective photopeak interaction length. The obvious downside of this approach is that we do not know what $\ell_{int}$ should be.

However, if we have multiple measurements for different values of $r$, we can figure out $\ell_{int}$. Starting from

$$
\frac{N_{detected}}{N_{emitted}} = \epsilon_{geo}\epsilon_{det}
$$

, we substitute $\epsilon_{geo}$ to get:

$$
\frac{N_{detected}}{N_{emitted}} = \epsilon_{det}\frac{A}{4\pi r^2}
$$

Moving the unknowns to one side and taking the square root, we end up with

$$
\frac{\tilde{r}}{\sqrt{\epsilon_{det}}} = \sqrt{\frac{N_{emitted}A}{4\pi N_{detected}}} =: H
$$
In this lab course, you will have to estimate the total number of events from the activity of the source and the time measured. For simulations, use the number of events you simulated.

1.4 Correlation plots

Estimate the geometric efficiency from the distance between the source and the detector and from the front face area of the detector. Use the integral function of ROOT to get both the number of photopeak events. Calculate \( \epsilon_{\text{det}} \). Think about how many gamma rays of which energies are emitted per decay.

1.4.1 Resolution

The resolution is a measure of the sharpness of a peak. This parameter is very important, e.g. if two peaks are close by, a good resolution would allow to distinguish them, whereas a too bad resolution would mix the two peaks in one. To classify peaks widths, a common parameter is the full width half maximum, or FWHM, which is the distance between the points where the count is half as height as the maximum count. Typically, peaks can be described by a gaussian function. For a Gaussian function with standard deviation \( \sigma \) the FWHM is given by:

\[
\text{FWHM} \approx 2.35\sigma
\] (1.3)

Of particular interest is the relative resolution, which is just the FWHM divided by the mean of the peak. This is always a dimensionless number, and commonly given as a percentage. The resolution is energy dependent \((\Delta E \sim \sqrt{E})\), so it is good practise to mention at which energy it was measured: for example, some calorimeter might have a resolution of 5% for 1 MeV photons.

The energy resolution is one of the key parameters of a calorimeter – it gives you an idea about the statistical uncertainty of an individual energy measurement.

1.4.2 Doppler effect

In a nutshell, the frequency of a plane wave – or any wave – depends on the coordinate frame. This is observed by the sound of a motorbike or a firetruck passing at high speed compared to a listener walking. For a wave which travels at the speed of light, such as an electromagnetic wave, Galilean Doppler equations have to be modified as following \[2\]

\[
\omega' = \gamma \omega (1 - \beta \cos(\theta))
\]

\[
tan(\theta') = \frac{\sin(\theta)}{\gamma (\cos(\theta) - \beta)}
\] (1.4)

where \( v \parallel e_z \) is the relative velocity between both frames of reference, \( \omega' \) and \( \omega \) are the apparent frequencies in the moving and fixed frames respectively, \( \theta' \) and \( \theta \) are the polar angles in the moving and fixed frames, \( \beta = \frac{v}{c} \) and \( \gamma = \frac{1}{\sqrt{1 - \beta^2}} \) is the Lorentz factor. The inverse equations are obtained by moving the \( \prime \) sign and changing the sign of \( \beta \).
A gamma ray is a photon which is a quantized electromagnetic wave. When emitted from an ion moving at relativistic speed, it is subject to Doppler shifting before being detected by a given (fixed) detector. We remember that the energy of a photon is given by \( E = \hbar \omega \).

Later, when the energy is deposited in one crystal, the angle corresponding to the center of the face of the crystal will be used.

**Error determination**

As values without their relative precisions is meaningless in physics, the precision on the resolution and efficiency should also be determined.

If you fit a gaussian to a peak with ROOT, ROOT will helpfully provide you with a precision for the parameters of the gaussian. This value is basically worthless, as it is – at most – a rough estimate of the statistical and numerical uncertainty from the fit. It will not tell you if your fit went horribly wrong in the first place\(^9\) nor will it tell you anything about the systematic uncertainties\(^10\) which might easily end up dominating the actual uncertainty.

Rather than doing pages upon pages of uncertainty propagation or quoting the misleading numbers given by ROOT, you should think about possible sources of errors, try to estimate them and how they might affect the overall result.

For the efficiency, the number of hits on the detector is assumed to follow a Poisson distribution, and therefore the absolute uncertainty is taken as the square-root of that number. For example, if you determine that based on source activity and geometric efficiency, 10000 gammas will hit the detector in a certain time span, the uncertainty on that number will be \( \sqrt{10000} = 100 \).

### 1.5 Simulation of particle detectors

Simulation of particle detectors consist essentially of defining some materials to represent a detector system, simulating a large number of particles travelling through it and looking at the resulting energy distribution in the material.

#### 1.5.1 Why simulate detectors?

Simulations in physics can be very powerful tools. In case one needs to design a detector to perform a particular task, for example. In order to get the best performance for the assigned task, the design of the detector has to be thought through carefully and tested. One way to do it, would be to realise the detector and test it during an experiment. However this cost a substantial amount of time and money. And if it turns out that it was ill-designed, then all the time and money would have been to a large extend wasted. A simulation allows to do them faster and for a much cheaper price. Furthermore the simulation can give the response of the detector regardless the flaws of the electronic to which it is connected. Similarly, the

---

\(^9\)Actually, there are indicators for the goodness of a fit, such as the \( \chi^2/\text{ndf} \), but these are beyond the scope of this course. You can learn about these in a data analysis lecture.

\(^10\)For instance, the APD response is temperature dependent, so if the lab temperature changes, so will the calibration.
geometrical disposition of a setup can be simulated in order to place the different detectors at the optimal position to detect the reaction studied.

When an experiment is performed, only global macroscopic information can be retrieved. This could be (and usually is) a superposition of different microscopic effects. In order to study the respective weight of the different effects taking place, the experiment can be simulated introducing controlled microscopic effects and thus allowing to disentangle the effects. In general, the level of agreement between simulation and experiment gives some indication if the model of the relevant processes in the experiment is adequate.

Apart from detectors, also primary interactions can be simulated\(^\text{11}\). This is important because in the experiments, one does not know what the actual primary interaction was, while in a simulation, one does. This is important if there is lots of background for the process one is examining, because it means one can simulate just the background processes and subtract them from the data.

### 1.6 C++

“When your hammer is C++, everything begins to look like a thumb.”

— Steve Haflich in alt.lang.design, December 1994

The programming language C++ was created in 1985 by adding the object orientation paradigm (e.g. “classes”) to the C programming language, a powerful but “bare metal” language. Over the decades, lots of features have been added to C++, the specification of C++17 has 1605 pages. A lot of the features have overlapping use cases, such as raw pointers, references and smart pointers. This means that almost everyone uses only a subset of C++, but everybody tends to use a different subset.

Like C, C++ is a compiled language. This means your program is translated from text to machine code by a compiler. In our case, you start this process by running `make`.

Two outcomes are possible: Either the compilation will finish without errors, or the compiler will complain about a particular line of code, which one will have to fix and try again. In general, bugs which cause the compiler to complain are easier to fix than logical bugs which will cause your program to do a different thing than what you wanted it to do, which may be obvious (such as a crash with “division by zero”) or non-obvious (such as silently corrupting your data).

C++ aims for zero-overhead abstraction. This means that a C++ program should be easier to read (and write) than a corresponding program using only C features (the abstraction part), while still having a similar runtime (the zero-overhead part). The downside is that there are no implicit safety checks. In Java, accessing an array with 10 elements with an index of 15 will result in a `IndexOutOfBoundsException`. With C arrays, the same thing will result in Undefined Behavior, meaning sometimes your program might crash, sometimes it might just corrupt data. For example, C++ provides `std::array`, an abstraction from C arrays. If you

\(^{11}\)This is beyond the scope of GEANT4, which focuses on particle matter interactions, in HEP, PYTHIA is widely used.
use an invalid index, the result will be just as bad as in C, because checking the array boundaries
would add overhead and C++ assumes you know what you are doing.\[^{12}\]

In experimental particle and nuclear physics, C++ has replaced FORTRAN as the main
language for both data analysis (e.g. with ROOT) and simulation (e.g. with GEANT4). If you
plan to do a thesis in that area, a course on C++ might be well worth your time.

Giving an introduction to C++ is out of the scope of this course by several orders of
magnitude, so instead, we will focus on a paint-by-numbers style approach.

1.6.1 Formatting

C++ does not care about whitespace characters such as blanks, tabulators or newlines. You
can format your code however you like without affecting functionality. Nevertheless, consistent
indentation increases readability.

1.6.2 Variables

Variables and constants are declared and defined using:

\[
\text{TYPENAME VARIABLENAME = INITIALISATION ;}
\]

Note that the single = is always an assignment operator. Use == if you want to compare
for equality.

In this course, TYPENAME is either:

- \text{int}: an integer number of a reasonable range, which we will use in loops.
- \text{const double}: A double precision floating point number which approximates real num-
  bers, which is not allowed to change its value. So basically a shorthand notation for some
  expression.
- \text{auto}: Not actually a type, but tells the compiler to figure out the type of the variable
  based on INITIALISATION. Useful when dealing with objects.

VARIABLENAME should be a combination of lowercase letters and underscore in this lab
course.

INITIALISATION can be an expression composed of numbers (42), operators (a+b), func-
tion calls (\text{sin(x)}) and the like.

Here are some examples:

\begin{verbatim}
int i=0; 
const double halflength=30.0*mm; 
auto sc = new G4Box("sc", halflength, halflength, halflength);
\end{verbatim}

\[^{12}\text{There is also a slower but safe way to access an element by using the at method, which will behave similar to Java.}\]
In line 2, the symbol \texttt{mm} refers to a constant conversion factor from GEANT which will convert millimeters to whatever GEANT uses internally as a length unit.

Note that expressions can sometimes have an effect without assignment to a variable (this is called a \textit{side effect}). In our code, \texttt{new G4PVPlacement(...)} is a common example of this.

1.6.3 For loops

One worthwhile goal when programming is to avoid redundancies. So when placing six crystals (two rows, three columns) in GEANT, instead of having six placement statements which would all have to be changed when moving anything, one can use loop over the columns and just have two placement statements, or even have an outer (row) and an inner (column) for loop and just one placement.

For loops follow the following blueprint:


code
\begin{verbatim}
1 for (VAR_INITIALISATION ; CONDITION ; INCREMENT)
2 BLOCK
\end{verbatim}

description
VAR\_INITIALISATION is just a variable initialisation of some loop variable. It is the first thing executed.

CONDITION is typically a comparison of the loop variable with something. Common operators are less-than (\texttt{<}) and equal-or-less-than (\texttt{<=}). The condition is checked every time before the loop is entered. If it is false, the loop is done and whatever comes after BLOCK is executed next. If it is true, BLOCK is run once. After that, INCREMENT is executed and we go back to the condition checking step.

Note that INCREMENT often contains shorthand assignment operators. \texttt{x++} is shorthand for \texttt{x=x+1}, “add the current value of x and one and store the result as the new value of variable x” or “increment x by one”. Similarly, \texttt{x+=2} is shorthand for \texttt{x=x+2}.

BLOCK is either the next statement, ended by a semicolon, or it is a list of statements enclosed in curly braces.

1.7 GEANT4

GEANT4 stands for GEometry ANd Tracking (version 4) \cite{4}. It is a set of libraries providing tools to perform simulations. It is programmed in C++, and follows an Object-Oriented philosophy. The algorithms developed are based on the Monte-Carlo approach \cite{5}. The Monte-Carlo algorithm consists in repeating similar process but with randomised starting conditions or events. In our case, the emission direction of the photons (which we will provide to GEANT), as well as the different interactions that will take place (which GEANT will do for us) are obtained randomly.

Geant has a rather long standing history as the very first version of Geant was developed at CERN in 1974. In 1998, the fourth version of Geant was released in order to replace Geant3 which was programmed in FORTRAN (as part of CERNLIB). In 1999 it was adopted by the XMM collaboration in space physics, and then high energy physics community adopted it too: Babar (2001), ATLAS, CMS or LHCb (2002). It is the de-facto standard for simulating particle-matter interactions in physics.
The structure of Geant4 operates around different components: material, geometry, physics lists, event generator, visualisation and analysis. But before, the global organisation of the code has to be presented.

1.7.1 Usercode

In order to simplify the programming part, a working simulation is provided as a starting point. It is composed of the following parts:

- **exampleN01.cc**: this is the main file of the program (which contains the `main()` function of the C++ code). It will be called when running the program, and is responsible for calling the other code.

- **Makefile**: this file is used for the compilation. In order to compile the program, one should simply type `make` in the directory.

- **include**: this directory contains the class/headers definitions. Header files give other parts of the code (such as exampleN01.cc) information on how to interact with it. For example, `ExN01DetectorConstruction.hh` gives information on how to interface with the code written in the class `ExN01DetectorConstruction.cc`.

- **src**: this directory contains the sources definitions. Each class has a certain number of corresponding functions. For this lab course, only the files in this directory needs to be modified.

The code organisation is then summarised in Figure 1.6.

Figure 1.6: Summary of the different parts of the usercode program with details on the files used.

The simulation represents a certain geometry composed with specific material and following some physics rules. The simulation is a run on several events, each starting with the action of a given source, and following the different steps of interaction.
1.7.2 Material

Geant4 allows for different definition of materials. The first and simplest way to define some material to be used in the simulation is by using the predefined library of Geant4 as

```cpp
auto CsI = man->FindOrBuildMaterial("G4_CESIUM_IODIDE");
```

And finally, one can define more complex material using, e.g.,

```cpp
auto CsI_Tl = new G4Material("CsI_Tl", density= 4.51*g/cm3, ncomponents=2); 
CsI_Tl->AddMaterial(CsI, 99.6*perCent); 
CsI_Tl->AddElement(Tl, 0.4*perCent);
```

In the code on which you will be working, the material is defined in the file `ExN01DetectorConstruction.cc`.

1.7.3 Geometry

The building of the geometry is composed of different definitions: solids, logical volumes and physical volumes. On the top, there is a special volume, in which all the others are placed (experimental hall). As the material, the geometry is defined in `ExN01DetectorConstruction.cc`.

![Diagram](image.png)

Figure 1.7: Steps to create a physical volume placement.

Figure 1.7 gives an overview of the different steps necessary to create an actual physical volume: First, a solid has to be defined. This is just an abstract geometrical object. Then, a solid and a material are used to create a logical volume. A logical volume can be thought of the idea of (for example), a 1 cm$^3$ CsI cube.

Finally, to create an actual physical volume, one has to create a placement using the logical volume and a vector indicating the point in space where its center of mass should be. A logical volume can be placed multiple times, each time generating a different physical volume.
Solids

Geant4 offers the possibility to define different geometrical forms and methods to combine them in order to produce the solids that will be needed to build the detector [7].

A simple box[^13] can then be defined in the following

```c++
const double al_x = 6.0*mm;
const double al_y = 6.0*mm;
const double al_z = 21.0*mm;
auto al_box = new G4Box("al_box",al_x,al_y,al_z);
```

As you can see, for predefined Geant4 solids, the length are given relative to the center of mass, similar to radii.

Similarly to the box, there are several others: G4Cons, G4Orb, G4Para, G4Sphere, G4Torus, G4Trap, G4Trd and G4Tubs. And those can be combined, like, e.g.

```c++
auto detector = new G4Box("detector",ge_x,ge_y,ge_z);
auto housing= new G4SubtractionSolid("housing",
  al_box,detector);
```

but also G4UnionSolid and G4IntersectionSolid. Combining all those, it is possible to define all the elements of the geometry.

Logical volumes

From the solids defined, one can then define some logical volumes as

```c++
auto housing_log =
  new G4LogicalVolume(housing,Al,"housing_log",0,0,0);
```

Essentially, it means to assign some material to one solid figure.

Physical volumes

Once the logical volumes have been defined, one can physically place them in the space using a G4Placement:

```c++
new G4PVPlacement(  
  0,G4ThreeVector(0,0,al_z+dist),
  housing_log,"housing", experimentalHall_log,false,0);
new G4PVPlacement(  
  0,G4ThreeVector(0,0,al_z+dist),
  detector_log,"detector", experimentalHall_log,false,0);
```

[^13]: In this case, the size of the box is 12x12x42mm³, as the dimensions are indicated from the centre of the solid.
Note that the position vector sets the position of the center of mass of the logical volume.\[1\]

It should be noted that in the particular code running, only the energy of physical volumes with the name "detector" are recorded, and furthermore, the copy number, indicate which elements it will be (it should then be different for each crystals in the same geometry). Calling the same line varying the (x,y) coordinates would allow to place the corresponding number of boxes into the geometry.

Take care that you do not place overlapping volumes, or the results might not be what you expect.

**World Volume**

There is a special volume called here the experimental hall in which the simulation will take place. It should be noted that the previous physical volumes were placed in the mother volume experimentalHall log. The definition of the hall follows the same principle as any others, with two differences: The fifth parameter of the placement (the mother volume) is 0, and the result of the new-expression is not actually ignored but returned by the Construct method.

### 1.7.4 Physics list

When running, the simulation have to know what are the rules that will apply and which particles can be generated. This is done in EXN01PhysicList.cc. In particular the particles are defined as

```cpp
void ExN01PhysicsList::ConstructParticle()
{
    // In this method, static member functions should be called
    // for all particles which you want to use.
    // This ensures that objects of these particle types will be
    // created in the program.
    ConstructBosons();
    ConstructLeptons();
    ConstructMesons();
    ConstructBaryons();
}
```

And the rules or processes as

```cpp
void ExN01PhysicsList::ConstructProcess()
{
    AddTransportation();
    ConstructEM();
    ConstructGeneral();
}
```

\[14\] Assuming primitive solids.
Here we only consider the electromagnetic interaction (no hadron) which details can be seen in

```cpp
void ExN01PhysicsList::ConstructEM()
```

1.7.5 Event generator

Now that the material, the geometry and the rules have been defined, the simulation can run.

Gamma Photon Source

The first thing is to define the source of the event: what will happen which creates the event (incoming ion, etc.). This is made in ExN01PRIMARYGENERATORACTION.CC. And we use a simple photon generator as

```cpp
auto particleTable = G4ParticleTable::GetParticleTable();
auto part_gamma = particleTable->FindParticle("gamma");
particleGun->SetParticleDefinition(part_gamma);
particleGun->SetParticleEnergy(511.0*keV);
particleGun->SetParticlePosition(G4ThreeVector(0.0, 0.0, 0.0));
```

Run

Afterwards the simulation can start. This is a run that is composed of several events. It is possible to have some actions at the beginning of the run as

```cpp
void ExN01RunAction::BeginOfRunAction(const G4Run* aRun)
{
    G4cout << "### Run " << aRun->GetRunID() << " start."
    << G4endl;
    fRootFout = new TFile("test.root","RECREATE",
    "My GEANT4 simulation");
    fRootTree = new TTree("tree","My GEANT4 simulation");
    fRootTree->Branch("e",e,"e[100]/D");
}
```

defined in EXN01RUNACTION.CC. A new ROOT file is opened and a tree with 100 branches (maximal number of crystals) is created (see [1.8.1]).
1.7.6 Generation of primary particles

But the main part of the event is the event action generator. Using the source (and in
ExN01PrimaryGeneratorAction.cc), the photon is emitted with an isotropic probability
distribution\[\phi_{\gamma} = \text{RandFlat}::\text{shoot}(-\pi, \pi)\;\]
\[\theta_{\gamma} = \text{RandFlat}::\text{shoot}(0, \theta)\;\]
\[\sin\theta = \text{RandFlat}::\text{shoot}(0, \sin\theta)\;\]
while( \(\sin\theta > \sin(\theta_{\gamma})\) )
\[v.\text{rotateY}(\theta_{\gamma})\;\]
\[v.\text{rotateZ}(\phi_{\gamma})\;\]
\[\text{if}(\theta_{\gamma}<\text{threshold})\;\]
\[\text{particleGun}\rightarrow\text{SetParticleMomentumDirection}(v)\;\]
\[\text{particleGun}\rightarrow\text{GeneratePrimaryVertex}(\text{anEvent})\;\]

This “emits” a photon with a polar and azimuthal angle determined randomly and a fixed
energy.

Event

Each event has four levels: beginning, event action generator, steps and end.

At the beginning of the event, one can initialise the variables that will be used, and at the
end, the resolution of the detection can be applied, the energy of each detector stored in the
ROOT tree. Both are defined in ExN01EventAction.cc.

Step

That photon, or more generally the particle(s) generated are sent through the experimental hall
and until they interact with one of the constituents (matter). This is a step. Each step ends
with either an interaction or when the particle leaves the experimental hall. Some action can
be done at each step, in particular, collecting the energy deposit by the particle

\[\text{if (aStep}\rightarrow\text{GetTotalEnergyDeposit}()>0)\{\]
\[\text{if(strcmp(vol,}\text{"detector")}\==0)\{\]
\[\text{runAction}\rightarrow\text{e}[\text{copyNo}] += \text{aStep}\rightarrow\text{GetTotalEnergyDeposit}();\]
\[\}
\[\}\]

\[\text{This means we have to sample the flat probability distribution of unit vectors, which is not trivial. The}
\text{listing shows one way to do it.}\]
1.7.7 Visualisation

When everything is correctly defined, the code can be compiled, and the first step is to check the geometry, this can be done using some visualisation method. There are a few available. Geant 4 offers the possibility to visualise the output by connecting with different renderer. Just a quick word of some visualisation method (not exhaustive)

- VRML file: generate a VRML file which can be viewed by a browser,
- DAWN file: generate a DAWN file which can then be converted as an eps (using dawn and wish),
- HepRep file: generate a HepRep file which can be viewed by a specific JAVA software. Contrary to previous ones, the viewpoint is not fix and it is possible to scroll the image dynamically,
- ASCII file: generate a text file containing the coordinates of the elements,
- RayTracer: can generate a JPG file from a fixed view point. Ultimately it is the simplest and safest as it is based on Geant4 geometry navigator. Indeed the others may have issues with complex representation. RayTracer can handle what Geant4 can.
- OpenGL: draw the image directly on the screen without the need of extra file. It is the simplest to use as it is well integrated into Geant4: that’s the system we will be using today.

Knowing those principle is a thing, how to get a visualisation of my setup? You can use the interactive mode of Geant4 with:

```
./bin/Linux-g++/exampleN01 -v
```

It is then possible to see a trajectories of primary particles by specifying both -n and -v:

```
./bin/Linux-g++/exampleN01 -n 10 -v
```

Later, you will use a predefined spherical detector covering almost $4\pi$. You can enable a cutaway mode by passing the -c flag to the program.

1.8 Analysis

1.8.1 With ROOT

The most straightforward way to analyze the data is with ROOT. Your supervisor will show you how to fit a Gaussian function to a photopeak in a histogram and display the parameters. Please do not analyze your data in the lab until you have finished both the experiment and the simulation. To use ROOT on another computer, you have different options:

16It depends on the configuration of each system to know which ones are available.
Debian and Ubuntu users can just install the package `root-system`.

Other distributions probably also have some prebuild packages.

For Mac OS X and Windows, there are also builds available on the ROOT website.

If you want to work with the CIP computers (either locally or remotely via FreeNX or ssh), root is already installed there.

As a last resort, you can also compile the ROOT sources.

There should be time to do the ROOT analysis in the lab. If there is not sufficient time, you might want to come back at a later time to finish the plots, or you might want to run ROOT at home.

One can open the .root files \[8\] with

```
root -l test.root
```

The simulation does not directly output histograms, but just a TTree\[17\]. In order to generate the histograms, one should execute the ROOT script as

```
.x script.C
```

Then you could do the following

```
root [2] .ls
TFile** test.root My GEANT4 simulation
TFile* test.root My GEANT4 simulation
OBJ: TH1I addback addback : 0 at: 0x2a6ce60
OBJ: TH1I single single : 0 at: 0x2a703c0
OBJ: TH1I energy0 energy0 : 0 at: 0x2a74010
OBJ: TH1I energy1 energy1 : 0 at: 0x2a77270
...
OBJ: TH1I energy98 energy98 : 0 at: 0x2bb3a50
OBJ: TH1I energy99 energy99 : 0 at: 0x2bb6ea0
OBJ: TTree tree My GEANT4 simulation : 0 at: 0x2bbaa90
KEY: TTree tree;1 My GEANT4 simulation
root [3] energy0->Draw();
root [4] .q
```

where `.ls` list the objects available, and the Draw function will draw that particular histogram. Alternatively, you can type `new TBrowser()` and draw the histogram by double-clicking on it in the list. After it has been drawn, it is possible to use different analysis tools. In particular, by dragging the mouse along the x-axis of the histogram, one can zoom in on a peak. The integral function, which is accessible by clicking right on the statistics at the top left corner,

\[17\] A sort of TTree elaborate event list
selecting `SetOptStat` and entering `11111111`, shows the number of counts that appear within the zoomed histogram. One can use that to calculate the detector efficiency and photo peak efficiency.

Using a right click on the histogram, one can choose the fit panel option, which allows to perform a Gaussian ("gaus") fit of the peak. The resulting $\sigma$ is then used to indicate the resolution of the device.

To scale the X-axis of a histogram by a linear function (e.g., for an energy calibration or to apply a Doppler correction), you can use the `~/histRescale.C` script. Read the usage example there. To scale the height of a histogram by a factor of $c$, use `h->Scale(c)`.

To draw a second histogram into the same plot as a first one, use `h2->SetLineColor()` and then `h2->Draw("same")`. More information can be found in the ROOT documentation.

**ROOT caveats**

Regarding ROOT programming, it should be stressed that whatever language CINT interprets, it is not really C++, but something subtly dangerously different. If you know C or C++, do not rely on proper scoping, RTTI, preprocessor macros, templates and STL and various other things to work correctly in CINT. The recommended workaround for these limitations is not to write CINT macros, but instead either use PyROOT or write standard C++ and link to ROOT as a library for anything more complicated than two lines.

Apart from CINT, another quite effective tripwire is the fact that in ROOT, every named object (e.g., a histogram) must have a unique name. If you create a histogram with a specific name, and then create another one with the same name, ROOT will simply replace the first histogram, which is probably not what you want. Another thing to remember is that ROOT makes use of no less than 30 global variables to simplify interactive use, so expect hidden side effects from any ROOT method calls.

### 1.8.2 With PyROOT

PyROOT is a python interface to ROOT. It provides a way to interact with ROOT without using the (so-called) CINT. Unlike C/C++ and ROOT, Python is taught in the introduction to programming lecture, so you might feel more comfortable using it than using C directly.

---

18. To display the fit parameters, right click on the statistics box again and set `SetFitStat` to `11111111`
19. The following is not so important for the analysis in the lab course, but quite important if you ever work on larger ROOT projects. Note that the situation has improved a bit with ROOT6.
21. A program named `root-config` exists which can tell your compiler (or Makefile) which flags to use to support ROOT.
22. As you don’t need to write much code for the analysis here, using CINT is ok.
23. sort of
24. A good practise is to set frequent offenders like gPad explicitly to zero after you are done with them, so they won’t be used by whatever unrelated part of your code gets executed next.
25. There are additional arguments to be made for the use of Python instead of CINT, by these are beyond the scope of this text.
To use PyROOT, open a python shell and type `import ROOT`. For most methods of most ROOT objects, there is a direct mapping to python. If you want to do something not directly accessible from python, such as executing a CINT script, you can use `ROOT.gInterpreter.ProcessLine`. So, an interactive python session could look like this:

```python
import ROOT
f=ROOT.TFile("SetUp1.root") # or sys.argv[1] in a .py script
ROOT.gInterpreter.ProcessLine(".x script.C")
tb=ROOT.TBrowser()
```

Calling a CINT function with python variables as arguments is a bit more tricky, the easiest way to scale histogram axis would be to rewrite histRescale.C in python. Pointer-specific stuff (including C arrays) is also rather hard to get right from python, in some cases, ctypes can help.

1.8.3 Without ROOT

If you want to use other software (like gnuplot) to analyze your data, you should export your root files to csv. You can do that like this:

```bash
$ root -l myfile.root
root [0]
Attaching file myfile.root as _file0...
root [1] .x script.C
0
1000
2000
file prefix is 'myfile'.
found addback, 12591 entries.
writing to myfile.sim_addback.csv
found single, 12591 entries.
writing to myfile.sim_single.csv
found energy0, 12591 entries.
writing to myfile.sim_energy0.csv
root [3] .q
$
```

This script will work both for both the experiment and the simulation root files. Note that you do not need to run `.x script.C` for experiment data as it is already histogrammed.

The script will generate a CSV file for any non-empty histogram. If you do not want to install ROOT on your computer for the analysis, it is recommended that you the export script in the lab for each root file.

Note that the two dimensional correlation plots will not be exported, please save them as images to use include them in your analysis. Unless you are using (Py)ROOT, please do not
expect your supervisor to help you with whatever software you pick for analysis.
Chapter 2

Experiment

In order to understand more fully the simulation, it is necessary to understand what happens in reality. Therefore we will start by setting up a rather simple experiment and do some measurements.

2.1 General procedure

2.1.1 Detectors

As we will only use the experiment for the basis of the simulation, rather simple detector configuration and measurement will be used. The goal will be to detect the $\gamma$-rays emitted by a $^{22}\text{Na}$ source. For that we will be using CsI(Tl) scintillating crystals coupled with Avalanche Photo-Diode (APD).

As the CsI crystals are hygroscopic, they are wrapped into isolating material and kept in boxes. For the experiment, two boxes are provided, one containing eight crystals and the large one containing six crystals. They are shown in Figure 2.1. The front sides of the boxes are labeled with prints indicating the crystal faces.

![Figure 2.1: Photos of the two boxes used for the different setup.](image)

In order to study different effects, four different setups will be considered.
Do not move the large box. Move the source and the small box only.

The detectors will be read out using Mesytec MPRB-32 preamplifiers, GSI FEBEX 3b digitalization cards as well as GSI MBS and GSI GO4 on the software side. See section 2.3.1 for detailed instruction on event recording.

Due to differences in APD gain, light collection, crystal deterioration and electronics, different detector channels will have the gamma peaks at different places in the spectra.

This means that the spectra have to be calibrated. A linear function of the form $E_{\text{cal},i} = m_i E_{\text{raw},i}$ is sufficient here. The small box is already calibrated. For the large box, you will do a calibration in Setup 1.

As the simulation is written to give the energy in units of MeV, the experiment will also be calibrated to MeV.

2.1.2 Source

The $\gamma$-emitter source looks like a clear plastic coin with the trefoil symbol printed on it. The actual source is the small dark dot in the middle. The source contains the radioactive isotope Na $\sim$ 22. On 2013-11-01, the activity of the source was 213 kBq.

**Preparation:** Look up the Na – 22 on the interactive nuclides chart of the IAEA[1]. What is the half-life of Na – 22 and how strong is the source today? What decay modes and daughter nucleides exist? What is emitted in the decay? Which energies do you expect to see in your detectors? How is the element denoted by symbol Na called in English?

The source should be mounted on a source holder at a height of 5.5 cm from the table for all experiments. Always keep the source aligned to the middle of the central crystals. For distance measurements, always measure from the box to the coin.

Given the strength of the source, measurement durations of 120 s tend to give reasonable statistics. For larger distances, longer measurements may be required.

2.1.3 Side measurements

For every setup, write down the distances between the source and the detector boxes. Also write down the time each measurement took.

2.2 Setups

2.2.1 Setup 0: Calibration

Create a calibration for the large box. Do this by placing the source next to it and running go4 for a minute or two.


2 Calculating time differences can be a bit painful. Unix users might want to try the `date` program to convert dates to the much more natural “seconds since UNIX” (aka epoch) time format. Use `date +%s` to convert the current date, `date -d 2013-11-01 +%s` to convert the reference date. You will still have to convert the half-life to seconds.
Afterwards, look at the raw spectra in lim_energy/febex_00. Find the 1.275 MeV peak and determine its exact position by fitting a gaussian to it. Calculate a calibration factor so that the peak position will come out at 1.275. Edit the file en_cal.txt in the go4 directory. Each line should contain three decimal integers separated by white spaces (denoting the channel number) followed by a term over x.

For example, the line
\[ 0 1 4 0.0005162*x \]

tells go4 that the calibration for lim_energy_0_01_04 is obtained by multiplying the raw value with 0.0005162.

After saving the changes, close go4 and restart it. Check the lim_cal_en... histograms: They should all be calibrated now.

2.2.2 Setup 1: Closest distance to large box

Place the source as close as possible to the large box and place the small box as close as possible to the far side of the source holder.

2.2.3 Setup 2: Closest distance to small box

Rotate the source holder by \( \pi \) around a reasonable axis. Repeat the last measurements.

2.2.4 Setup 3: \( d_{big,small} = 5 \text{ cm} \)

Move the source 5 cm away from the large box, and the small box a further 5 cm away.

2.2.5 Setup 4: \( d_{big,small} = 10 \text{ cm} \)

Same as before, but with \( d = 10 \text{ cm} \).

2.3 Experiment instructions

For the experiments, you should carry out the following instructions.

Your supervisor will give you access to the setup with the detector boxes placed arbitrarily and a PC running go4 recording histograms. If any problems arise (e.g., you close go4 or you don’t see any events any more), please consult your supervisor.

In go4, you have two different main windows. One window shows the energy histograms for all three channels. Channel 0 is the upper crystal in the big box, channel 1 is the lower crystal in the big box, and channel 2 is the crystal in the small box. Note that you can double click on any histogram to see it larger.

The other window shows three two-dimensional correlation histograms between the different crystals. They are clearly labeled to indicate which correlations they show.

In general, you will first want to look at the energy histograms to see if you are recording sensible data, and then – depending on your setup – also check the 2d histograms for correlations.

31
2.3.1 DAQ howto

For a run\(^3\), please follow these steps:

1. Position the detectors and source according to the setup you want to measure. Adjust the source carrier, if necessary.

2. Ensure that the “Stop Analysis” button was pressed in Go4.

3. Click on the “Clear Histograms” Button, then on “Start Analysis”.

4. Wait for a time until the relevant peaks are clearly visible\(^4\).

5. Click the “Stop Analysis” button.

6. The time the Go4 “Analysis” ran is shown in the lower right corner. Write it down.

7. Press Ctrl+Y to save the histograms. Please save it in your group directory
   (/home/geant/Groups/ < semester > / < groupnumber > /). Pick a different file name for each run. Stuff you might want to put in is the setup, the run time and the fact that it is experimental data and not simulation.

---

\(^3\)That is physics slang for a “session where you record events”.

\(^4\)Generally, do not wait longer than seven minutes.
1. Outside G04, using another terminal/shell, you can open the saved files with ROOT. Using a Gaussian fit, calculate the resolution of the detection. Knowing the activity of the source used you should calculate the efficiency of detection.
Chapter 3

Simulation

Now, the students should go to the computers dedicated to the simulation. There will be two tasks to be performed: reproduce the results obtained in the experiment and design a detector system.

3.1 Reproduction of Experiment

Although it will be simplified, the experiment that you realised will be simulated.

3.1.1 First Simulation

The source provided already simulates the setup # 1 used in the experiment. You should check the geometry definition in order to understand the different components. Then, in the main directory you should type `make` in order to compile the program.

Then, by typing `exampleN01 -v`, you will be able to see the geometry of the simulation. If you run the program with `exampleN01 -n 10 -v`, you will see the visualization with the gammas from ten decays.

Alternatively, type `exampleN01 -n 1000000`. This will produce a `test.root` file. Using ROOT, you should reproduce the plots of the experiment, calculate the resolution and efficiency. To open the file in root, run `root -l test.root`. However, the simulation does not directly output histograms, so you will first have to histogram your data by typing `.x script.C`. This will generate a number of histograms: energyN, which describes the energy deposited in the detector volume with copyNo N, single, which corresponds to the energies deposited in any crystals, and addback, which is the sum of all energies deposited in the crystals for an event. To view them, you can just run `new TBrowser()`.

3.1.2 Simulate experimental setup

1. Run the simulation in visualization mode. Your should see the small box.

   1\textsuperscript{1}This will be useful when you apply the Doppler correction later on to see if the resulting distribution looks.
2. Run the simulation in batch mode, simulating a million events. Look at the resulting histograms.

3. You should modify the geometry of the simulation in order to add the large box with the six big crystals. Using the new simulation, you should reproduce the experiments and simulate one million events per setup.

4. Change the material of the detector from CsI to Ge (and change the resolution (\texttt{sigma\_detector}) accordingly (from 5% at 1MeV to 0.1% at 1MeV)).

Remember to change back the sigma global variable and use CsI as the detector material again for the following tasks.

3.2 Development of a detector system

We will now design a new detector which should detect a $\gamma$ emitted from a moving ion.

3.2.1 Detect a $\gamma$ emitted by a moving ion

![Figure 3.1: The setup viewed from the side](image)

Here, we will simulate an ion moving at relativistic speed ($\beta = 0.7$) towards a hole in a group of CsI crystals and emitting a 1MeV $\gamma$ 30 cm from the front faces of the crystals. In the first version, there are four 2x2x15cm$^3$ CsI crystals surrounded by a thin (1mm) Al frame symmetrically placed around the beam axis, as shown in the figure.
You have to modify the event generator, to emit a single $\gamma$ isotropically (in the ions center of mass system) and then apply the Doppler correction to both its energy and $\theta$.

Then, adjust the geometry part of the code. Build a logical volume both for a CsI crystal and the Al frame, and then use four placements to set up your detector.

Run the simulation.

**TODO:** Doppler Analysis.
Chapter 4

Analysis

4.1 Calibrated spectra

First, each detector channel has to be calibrated. To do this, pick a peak with a known energy in the spectrum and calculate $E_{\text{channel number}}$. Now, you just have to scale the x axis by that factor. Please calibrate to MeV – this means they can easily be compared to the simulation spectra.

As ROOT does not really make it easy to rescale the x axis, I have written a small script called histRescale.C that you can use for this task. That file also includes usage examples.

Produce calibrated spectra for all three channels of experiment setup #4.

4.2 Comparing experiment and simulation

For setup #4, compare the spectra of the simulation with the experiment. Ideally, put both the calibrated experiment spectrum and the histogram from the simulation in the same plot, rescaling the y axis of either one of them so that the heights of the photopeaks match.

In short this involves opening the experiment data, creating a calibrated histogram, drawing that histogram, opening the simulation data\footnote{TFile::Open("mysimulationfile.root")}, finding the histogram, drawing it using histpointer->Draw("SAME"), and rescaling one of the histograms by right-clicking on them and selecting the appropriate function.

As an alternative, it is also acceptable to have one of the plot below the other one in your analysis.

Discuss the differences in both histograms.

4.3 Resolution

To determine the resolution for a photopeak, you will have to fit a gaussian function to the peak. Use calibrated spectra for this. Please restrict the fit area to the range in which the counts in each bin are at least half of the height of the peak. From the fit, you get two important...
parameters: the mean and the standard deviation. This allows you to calculate the absolute resolution (in MeV or keV) and the relative resolution (in percent).

Determine the absolute and relative resolution for both the 511 keV and the 1275 keV peak for the small and one of the large crystals for setup #4 for both experiment and simulation. State unambiguously if your resolution is given as the standard deviation of the FWHM.

In the simulation, the width of the peak was designed to be proportional to the square root of the peak energy. Does this also work for the experiment, e.g. is \( \frac{\sigma_{511}}{\sqrt{511 \text{ keV}}} \approx \frac{\sigma_{1275}}{\sqrt{1275 \text{ keV}}} ? \)

### 4.4 Effective distances and photopeak efficiencies

For setup #4, show the correlation plots between

- the small crystal and a large crystal
- both large crystals

for both the simulation and the experiment. Try using a logarithmic z axis, use it if you like the result better.

Discuss the features of the correlation plots, and any differences between experiment and simulation. Motivate the addback method from one of these plots.

### 4.5 Detector material impact

For setup #4 with germanium detectors, include a plot of a histogram of a channel of your choice. Determine the photopeak efficiency and the resolution. Compare with CsI. Discuss.

### 4.6 Moving Ion setups

Edit a copy of script.C so that it will histogram the gamma energies in the reference frame of the moving ion instead of the lab frame. To do this, take the angle from the middle of the front face of each crystal and calculate the doppler correction factor for that angle. In the script, multiply each channel by the corresponding correction factor. You should get a photopeak at 1 MeV.

You will probably have to adjust your script for each setup to reflect your numbering of the crystals.

Plot the resulting addback histogram for each setup. Fit a gaussian to the photopeak. Determine the total photopeak efficiency and the resolution for each setup. Discuss.

Evaluate the cost of all the configurations studied: to simplify, the cost is estimated to be of 4 euros/cm\(^3\) of CsI crystal and 250 euros/channel of electronics.

Which design would you suggest to respect certain constraints in resolution, efficiency and cost (considering only the size of section of the crystals, their number -and organisation)? The

\(^2\)Lots of nested for loops here.
\(^3\)This is very helpful if you have a large noise peak dominating everything else.
\(^4\)One channel for one crystal.
constraints are the following: a resolution at 1MeV below 4%, a cost below 6000 euros and a total efficiency above 5%.
Bibliography

"Techniques for Nuclear and Particle Physics Experiments”, W.R. Leo.


UserDocumentation/UsersGuides/ForApplicationDeveloper/html/apas10.html


[10] "Welcome to MBS”, http://daq.gsi.de/


