Resolution of a silicon pixel detector

determined by means of a Monte Carlo simulation with Allpix^2

Instrumentation School in Particle, Nuclear and Medical Physics (24 August to 1 September 2023)

Authors: P. Schütze and S. Spannagel, DESY

Håkan Wennlöf, DESY
hakan.wennlof@desy.de

August 29, 2023

Contents

1. Overview

This lab exercise involves a detailed investigation on how to perform Monte Carlo simulations of silicon pixel detectors. The key objectives are 1) to simulate the response of a silicon pixel detector to a traversing highly relativistic ionizing particle, 2) to find a way to determine the spatial resolution of a binary planar silicon pixel detector, and 3) to study the influence of parameters such as the introduction of a magnetic field, altering the pixel pitch, effects of moving from a binary detector readout to sampling of the signal with a QDC and the consequences of having an inclined particle incidence on the spatial resolution.

1.1. Prerequisites

This lab session requires some basic knowledge and interest in the field of semiconductor physics. A prior knowledge in particle physics, experience in data analysis and programming is not necessary for the successful completion of this lab session. Few important literatures are mentioned in Chapter 2 for the interested readers.

The data visualisation for the tasks in this lab session is carried out using ROOT [1] and Allpix^2 [2]. ROOT is an open-source data analysis framework developed by CERN. Although designed originally for particle physics data analysis, it is also used in other applications such as astronomy and data mining. Some useful ROOT commands can be found in Appendix A.2. A more detailed ROOT "cheat-sheet" can be found at [3]. Allpix^2 is a generic simulation framework for silicon tracker and vertex detectors with the objective to provide an easy-to-use package for simulating the performance of silicon detectors, starting with the passage of ionizing radiation through the sensor and finishing with the digitization of hits in the readout chip [4]. Both these software frameworks are written in C++, but using Allpix^2 does not require modification of any C++ code.

1.2. Preparations

For a proper preparation for this lab session, the readers are advised to read through the whole manual carefully.

2. Short excursion into the world of silicon pixel detectors

Pixel detectors are most commonly solid state (semiconductor) particle detectors, which have been originally developed for particle physics applications. The current state of the art is represented by the detectors in operation or under construction at the Large Hadron Collider (LHC) at CERN, in the ATLAS, CMS, LHC-b, and ALICE experiments. Although born originally for the needs of particle physics community, pixel detectors have proven to be potentially useful in developments with applications in imaging, most notably biomedical imaging and imaging for X-ray astronomy.

The working principle of a pixel detector is based on the fundamental properties and characteristics of semiconductor materials used to build these detectors, such as silicon or germanium. Since the early 1960s semiconductor detectors have been used in nuclear physics, particularly for gamma ray energy measurements. Around late 1970s, the tracking capabilities of semiconductor detectors was realized in the field of particle physics and silicon still remains the number one choice of material for these applications owing to its abundance and well-known electrical properties.

In order to understand and interpret the results of the lab exercises, it is useful to have an understanding of the underlying physics. A comprehensive overview of semiconductor physics, types of solid-state detectors with an emphasis on pixel detectors, and detection principles are be covered in the lecture sessions of this instrumentation school. The students are encouraged to ask as many questions as possible in the lecture sessions for a superior understanding of various concepts. Furthermore, interested readers can check out some of the literature mentioned below.

A very nice introduction to particle detectors (both in German [5] and translated to English [6]) can be found in “Particle Detectors, fundamentals and applications” by Kolanoski and Wermes.
A slightly more advanced book dedicated entirely to Pixel Detectors is ”Pixel Detectors, From Fundamentals to Applications”, by L.Rossi et al. [7].

3. The Allpix Squared framework

Allpix^2 is a generic, open-source software framework for the simulation of silicon pixel detectors. Its goal is to ease the implementation of detailed simulations for both single detectors and more complex setups such as beam telescopes, from incident radiation to the digitised detector response. It is built in a modular fashion, which separates central infrastructure components from the actual physics simulation implemented in individual modules. Therefore the development of new simulation algorithms can be performed with little knowledge of the underlying structure.
The four main stages of the simulation from a particle crossing a silicon sensor to the final detector response can be summarized as follows:

Interaction: In this very first stage, an incident particle interacts with the detector material, depositing energy in the material. This energy loss is converted to electron-hole pairs in silicon. In this tutorial, the Geant4 framework [8–10] is used to simulate the amount and position of the energy deposited in the sensor via ionization processes. Geant4 is a toolkit for the simulation of the passage of particles through matter. It includes facilities for handling geometry, tracking, detector response, run management, visualization, and user interface. After the simulation of energy deposition in the sensor, the number of electron-hole pairs is calculated based on a (configurable) average creation energy for electron-hole pairs.

Propagation: After eletrons and holes have been created, they move in the silicon lattice owing to drift and diffusion processes. In the simulation, a (user-defined) electric field inside the active sensor volume is used to guide the charge carriers towards the collection electrodes. Several methods for this propagation exist, including a fourth-order Runge-Kutta stepping method which used in this tutorial.

Assignment to readout channel: The signal formed by the movement of charge carriers is processed by the front-end electronics. In the simulation, the signal has to be assigned to a front-end channel, i.e. a pixel collection electrode, before this takes place. Therefore, depending on the final position of the charge carriers, they are associated with (or: "transferred to") a readout channel.

Front-end response: Finally, the behaviour of the readout circuitry is simulated: The charge assigned to individual pixels is processed by the amplifier and discriminator in this pixel cell, digitizing the signal with a configurable conversion from charge carriers to charge-to-digital converter (QDC) counts (gain), dispersed thresholds, as well as noise contributions.

In all of these steps there are various parameters that can be customized by the user in order to either adapt it to a specific sensor, the details required as an output, or for an optimization of the simulation performance. More information on individual parameters can be found further down in this tutorial.
A few resources for the software framework:

Website: https://cern.ch/allpix-squared/

User manual: https://allpix-squared.docs.cern.ch/docs/

Repository: https://gitlab.cern.ch/allpix-squared/allpix-squared

4. Detailed description of the task

One of the goals of silicon sensor R&D is to develop position-resolving detectors with improved spatial resolution. Although the spatial resolution of a silicon pixel detector is to a large extent determined by the pixel pitch, the choice of readout mode (analog or single threshold binary), particle incidence angle, and the influence of a magnetic field are some tricky and less evident contributing factors to the resolution.

This exercise exercise consists of two parts. In this first part, students will perform a visual inspection of the detector setup by playing around with various detector parameters to simulate the response of the detector to the incidence of a high energetic relativistic particle. This will be followed by simulation studies to determine the standard binary resolution, and find ways to improve the spatial resolution of the detector.

The sections below are intended to give the readers a short description of the various tasks in this lab exercise. As you follow the instructions in Chapter 5, you will recognize all the steps discussed here and understand them in more detail.

4.1. Visual inspection of the setup

The particle accelerators at CERN boost particles to high energies before they are made to collide inside detectors. The detectors gather information about the collision products such as particle speed, mass, and charge, from which physicists can work out a particle's identity. The process requires accelerators, powerful electromagnets, and layer upon layer of complex sub-detectors. In modern particle detectors, the tracking detectors determine the path and through that the momentum of a charged particle; calorimeters measure a particle's energy, and dedicated particle-identification detectors use a range of techniques to pin down a particle's identity.

Particles produced in collision events normally travel in straight lines. However, in the presence of a magnetic field, the paths of charged particles become curved. The particle detectors are immersed in an approximately uniform magnetic field generated by electromagnets around them to exploit this effect. Momentum measurements can be made by applying a magnetic field perpendicularly to the direction of travel in a tracking detector, which causes the charged particle to curve into a circular orbit with a radius proportional to the momentum of the particle (via the Lorentz force given by $\vec{F} = q\vec{v} \times \vec{B}$, which gives a relation between the momentum and the radius as $p = rqB$). If the momentum of the particle is large, the radius of the trajectory is large, and the path is almost straight. If the momentum is small however, the radius of the trajectory is small, and the path is

tightly curved.

As the particles pass through the detector, they interact with silicon pixel detectors at multiple points, producing small electrical signals in each detector plane. The signals are then amplified and recorded. A series of electrical "hits" is used to determine the trajectory of the particle in the tracking system, and a computer-generated "best fit" to this trajectory gives the track radius of curvature and therefore the particle momentum.

Have a close look at the **`detector.conf`** and **`visual.conf`** files described in Appendix A.3. Play around with the various parameters (e.g. the `magnetic_field` and the `source_energy`) and try to comprehend the results.

4.2. First simulation

Depending on the angle of incidence and the incident point of a traversing particle, electron-hole pairs can be created in more than one pixel. In such a scenario, two or more adjacent pixels can be triggered by the same particle if signal charge (electron-hole pairs) is shared between the pixels. The group of pixels showing the signal from the same particle is called cluster. The **`DetectorHistogrammer`** module, as included in Appendix A.3, performs the clustering of the input hits by adding all the hits which are adjacent to each other to an existing cluster. The clusters are merged if there are multiple adjacent clusters. Electron-hole pairs created in a single pixel can also spread to adjacent pixels during charge propagation, causing a larger cluster size.

The module **`DetectorHistogrammer`** creates a lot of histograms for an initial evaluation of data. A few important examples are:

- A hitmap of all pixels in the pixel grid, displaying the number of times a pixel has registered a hit during the simulation run
- A cluster map indicating the cluster positions for the whole simulation run
- Distributions of the cluster size in the x- and y-directions, and the total cluster size
- Residual distribution in x and y between the center-of-gravity position of the cluster and the primary particle (i.e. how well the position can be reconstructed)

The parameter "residual" is defined as the measured hit position minus the expected hit position from the track extrapolation/Monte Carlo particle incitent position. The study of the residual distribution gives valuable information on the sensor spatial resolution after accounting for the multiple scattering and charge sharing, which is the charge leak from one pixel into the neighbouring cells. The spatial resolution is obtained from the RMS of the residual distribution for all clusters.

Please pay special attention to the plots mentioned above. Discuss what you see, and make sure to save the plots.

4.3. Extracting the detector resolution

For the simplest configuration with a binary read out pixel detector with a pixel pitch of w, a spatial resolution of $\sigma = \frac{w}{\sqrt{12}}$ is reached for single-pixel clusters. In this task, we will use the Allpix^2 simulation framework to determine this standard resolution, but also to find ways to improve the resolution by considering the influence of various parameters.

4.3.1. Effect of digitization

A pixel cell registers a hit when the corresponding readout channel responds to the charge deposited in the sensor in a way that reaches above a given threshold. In binary readout mode only the information whether the pixel has seen a hit or not is provided, which leads to limitations of the spatial resolution of a detector due to statistical effects. Considering the simplified two-dimensional case where a detector is hit at a random position x between two readout pads with the pitch p, the standard deviation σ of this distribution is given by :

$$\sigma^2 = \int_{-p/2}^{p/2} \frac{x^2}{p}\, dx = \frac{p^2}{12} \tag{4.1}$$

A detailed derivation of this binary spatial resolution can be found in Appendix E.1 of [5] and [6]. Thus the best possible spatial resolution of a detector with single-pixel hits and binary readout is $\frac{p}{\sqrt{12}}$. Since the pitch cannot be made arbitrarily small this limits the resolution of the detector.

As a first step, let us examine the effect of digitization, i.e moving from a binary detector readout to an analog signal readout by the sampling of the signal with a charge-to-digital converter (QDC). In this readout mode not only the position of a hit pixel but also the measured charge of the hit is available. The free charge carriers created by an ionising particle in the sensor follow the electric field lines and create a signal in the pixels surrounding the hit trajectory. If charge is spread on more than one readout channel, algorithms such as center-of-gravity or the η-method can be used to study its effect on spatial resolution. Play around with QDC resolutions and discuss the effects on spatial resolution you see. Do not forget to check the influence of threshold on the resolution. The threshold value, measured in electrons, corresponds to the minimum charge release for a hit to be considered. Vary the threshold values in different ranges to study its effect on the resolution.

4.3.2. Effect of rotation

Rotating the detector in the test beam at different angles allows us to study the effect of the angle of incidence of the particle on the spatial resolution. Vary the rotation angle and discuss how the residual distribution is changing. Analyse the effect of rotation in terms of charge sharing and cluster size within the sensors.

4.3.3. Effect of magnetic field

The presence of a magnetic field influence how electrons and holes drift in silicon, making them move at an angle with respect to the electric field direction. This angle Θ_L is called Lorentz angle and has an influence on the spatial resolution of the detector.The Lorentz angle Θ_L, by which charge carriers are deflected in a magnetic field perpendicular to the electric field is defined by Equation (4.2) :

$$\tan(\Theta_L) = \mu_H B \approx r_H \mu B \tag{4.2}$$

The Hall mobility (μ_H) differs from the drift mobility of the charge carriers by the very weakly temperature dependent Lorentz factor r_H :

$$\mu_H = r_H \mu, \tag{4.3}$$

with the values for r of 1.15 for electrons and 0.72 for holes at 0° C.

Play around with different values for the magnetic field and observe the residual plots. Try to think about what is happening in the sensor when you change the magnetic field. Make sure to save the plots for future reference.

5. Instructions

This section contains the instructions which should be followed step-by-step during the lab session. All the steps one will perform here have been discussed in the previous section. You are encouraged to scroll back and read Chapter 4 before getting started.

5.1. Set the environment

An installation of Allpix^2 is required for this tutorial. Allpix^2 is already provided and the executable **`allpix`** is available in any terminal. Table A.1 describes a short overview on Linux terminal and some useful Linux commands are listed in Table A.1. A detailed tutorial for absolute beginners on Linux command line can be found at [11].

The configuration and geometry files **`detector.conf`**, **`visual.conf`**, **`start.conf`** and **`replay.conf`** are located in the directory **`configs`**. Either edit them there or copy them to a separate working directory.

5.2. Visual inspection of the setup

Look at these files described in Appendix A.3:

- **`detector.conf`**: Contains the geometry information of the setup, with the type, position, and orientation of your detector. The detector used in this example follows the geometry of a CMS Phase I Pixel detector, with 52×80 pixels of the size $150 \times 100\,\mu m^2$.
- **`visual.conf`**: Simplified simulation file for visualization. The important modules in this file and their functions are described below :

  `Allpix`: Contains global parameters such as file locations, number of events, ...

  `GeometryBuilderGeant4`: Translates the defined geometry into a Geant4-compatible geometry, which is needed to leverage the Geant4 energy deposition simulation and visualisation.

  `MagneticFieldReader`: Defines a magnetic field for the Geant4 simulation and the charge propagation in the sensor.

  `DepositionGeant4`: Runs Geant4 to determine the particle trajectory and the energy deposited inside the active sensor volumes.

`VisualizationGeant4`: Makes it possible to visually inspect the geometry setup.

Run Allpix[2] giving it the configuration file to be used:

```
allpix -c visual.conf
```

An error message should pop up. What does this error message tell you?

Play around with the geometry and the beam description; test out changing various parameters (e.g. the `magnetic_field` and the `source_energy`) and try to comprehend the results. Discuss with your partner and tutor what you observe. Do not forget to save the results you see.

Note: visualisation may have some problems on newer Geant4 versions. Ask a tutor if things do not work, as they can provide example images.

5.3. First simulation

Look at the following configuration file, containing the complete workflow of a detector response simulation for a single detector:

- **`start.conf`**: In this file, the modules used for the simulation and their parameters are defined.

  `Allpix`: Global parameters such as file locations, number of events, ...

  `GeometryBuilderGeant4`: Translates the defined geometry into a Geant4-compatible geometry.

  `MagneticFieldReader`: Defines a magnetic field for the Geant4 simulation and the charge propagation in the sensor.

  `DepositionGeant4`: Runs Geant4 to determine the particle trajectory and the energy deposited inside the active sensor volumes.

  `ElectricFieldReader`: Defines an electric field for the charge propagation in the sensor, either calculated from a function or read in as a more realistic, TCAD-simulated field.

  `GenericPropagation`: Handles propagation of the charge carriers in the sensor. One can switch on/off the propagation of electrons and holes, define step sizes, propagate groups of charge carriers, change mobility models, ...

  `SimpleTransfer`: Association of propagated charge carriers at the sensor surface to the corresponding pixel.

  `DefaultDigitizer`: Digitization of the charge information, emulating the pixel read-out electronics. Define noise, thresholds and the QDC parameters.

`DetectorHistogrammer`: Provides plots for an overview of the simulated detector performance.

The `[DetectorHistogrammer]` module is one of the most important ones for this lab exercise. It performs an initial data analysis, and provides numerous histograms.
Run this simulation for 2000 events (Hint: When the simulation is interrupted with **`Ctrl+c`**, the data are saved up to the currently processed event). Have a look at the generated ROOT file (use a TBrowser), which is located in **`/your/working/directory/output`**. This file contains a subfolder for each simulation module, with plots relevant to that module. Focus on the `[DetectorHistogrammer]` module. This module creates a lot of histograms for an initial evaluation of data. A few important examples are:

- A hitmap of all pixels in the pixel grid, displaying the number of times a pixel has registered a hit during the simulation run
- A cluster map indicating the cluster positions for the whole simulation run
- Distributions of the cluster size in the x- and y-directions, and the total cluster size
- Residual distribution in x and y between the center-of-gravity position of the cluster and the primary particle (i.e. how well the position can be reconstructed)

The parameter "residual" is defined as the measured hit position minus the expected hit position from the track extrapolation/Monte Carlo particle incitent position. The study of the residual distribution gives valuable information on the sensor spatial resolution after accounting for the multiple scattering and charge sharing, which is the charge leak from one pixel into the neighbouring cells. The spatial resolution is obtained from the RMS of the residual distribution for all clusters.

Please pay special attention to the plots mentioned above. Discuss what you see, and make sure to save the plots.
The output file name can be change with the **`root_file`** keyword under the global `[Allpix]` header.
Play around the the **`log_level`** parameter to change the level of output details printed on the terminal. The list of possible logging levels can be found in section 4.2 of the Allpix^2 manual [4]. To have a lower amount of log output, set it to "WARNING" or "ERROR". To have more detailed output, set it to "DEBUG".

5.3.1. A closer look at the charge propagation (optional)

Start from the file **`start.conf`** and switch on the options **`output_animations`** and **`output_linegraphs`** (module **`GenericPropagation`**) and run this simulation for one individual event. Have a look at the output root file and at your output directory.

Hint: One can adjust the parameter **timestep_max** to achieve more meaningful results.

5.4. Main task: Extracting the detector resolution

In Allpix^2 it is possible to store and recall the data of intermediate simulation steps using the ROOTObjectWriter and the ROOTObjectReader modules. For the sensor under investigation we have created files containing objects of the type DepositedCharge and PropagatedCharge for different sensor rotations and magnetic fields. This should make it faster to investigate the detector resolution as a function of these two parameters while sparing the two most time-consuming steps performed by DepositionGeant4 and GenericPropagation. The files are called **Tutorial_<rotation>deg_<BField>T_data.root** and are located in the directory **data**.
The goal of this exercise is to **optimise the resolution** of a single silicon sensor of the CMS pixel type, by varying different parameters. Think about how we can improve resolution (hint: it is strongly related to the cluster size).
Look at the configuration file:

- **replay.conf**: Configuration loading the simulated propagated charges.

  `Allpix`: Global parameters such as file locations, number of events, ...

  `ROOTObjectReader`: Read in intermediate simulation results from previous runs. Choose which objects to load.

  `SimpleTransfer`: Association of propagated charge carriers at the sensor surface to the corresponding pixel.

  `DefaultDigitizer`: Digitization of the charge information, emulating the pixel read-out electronics. Define noise, thresholds and the QDC parameters.

  `DetectorHistogrammer`: Provides plots for an overview of the simulated detector performance.

Start without rotation and without magnetic field (**Tutorial_0deg_0T_data.root**) and calculate the RMS of the residual (resolution) in both the x- and y-directions (that's something you could automate/script).

Investigate the following parameters and provide an explanation for the effect seen and detailed answers to the questions.

Digitization: By default our silicon detector gives us binary hit information (**qdc_resolution = 1(given in bits)**). For single-pixel clusters, we get a detector resolution of $\sigma = w/\sqrt{12}$. Go to typical QDC resolutions for charge readout (e.g. 4, 6, 8 bits). How does it influence the resolution? Does the threshold have an influence? Pick a QDC resolution, and vary the threshold in `[DetaultDigitizer]`. Keep an eye on the cluster size histograms as well.

Rotation: At which rotation angle does one achieve the best resolution? Why? Is this influenced by the resolution of the QDC?

Magnetic field: At which magnetic field does one achieve the best resolution? Why? Is this influenced by the resolution of the QDC?

Look at the residual distributions. Discuss what you see. Why are the shapes the way they are, adn where do the different features come from? Make sure to save the files, so that you can make comparison plots of the residual and cluster size distributions. Remember that the parameter `root_file` determines where the data are saved.
Hint: It is not necessary to create one configuration file for every set of parameters. Any parameter in the configuration file can be overwritten on the command line when calling Allpix^2 using the following syntax:

```
allpix -c some/configuration.conf
       -o generic_parameter=value
       -o CorrespondingModule.desired_parameter=value
```

To run the replay configuration with a QDC resolution of 4, with no rotation and a 0 T magnetic field, you could thus do

```
allpix -c replay.conf
       -o root_file="QDC4_rot0_0T.root"
       -o ROOTObjectReader.file_name="../data/Tutorial_0deg_0T_data.root"
       -o DefaultDigitizer.qdc_resolution=4
```

5.5. Drift-diffusion visualization of the charge carriers

Start from the file **`start.conf`** and switch on the option **`output_linegraphs`** (module **`GenericPropagation`**) and run this simulation for one individual event for different angles of incidence and different values of magnetic fields. Make sure to rename the output file to avoid overwriting the one from the previous analysis and have a look at the output ROOT file in the output directory.

As a second step, switch on the option **`propagation_holes`** and look at the visualisations by varying magnetic fields and angles of incidence. Discuss with your partner what you see and make sure to save all the plots.

5.6. Extra task (optional)

Can you set up and simulate a typical test beam experiment with a six-plane EUDET-type (Mimosa26 sensors) beam telescope and a DUT (Device under test)?
Hint: Have a look at the list of predefined sensor geometries in the Allpix^2 repository at https://gitlab.cern.ch/allpix-squared/allpix-squared/tree/master/models

A. Appendix

A.1. Useful Linux commands

The Linux command line, often referred to as the shell, terminal, console or prompt is a program that takes commands from the keyboard and gives them to the operating system to perform. In the old days, it was the only user interface available on a Unix-like system such as Linux. Nowadays, we have graphical user interfaces (GUIs) in addition to command line interfaces (CLIs) such as the shell.

On most Linux systems a program called bash (which stands for Bourne Again SHell, an enhanced version of the original Unix shell program, sh, written by Steve Bourne) acts as the shell program. Besides bash, there are other shell programs available for Linux systems. These include: ksh, tcsh and zsh.

An overview of some of the most useful Linux terminal commands are listed in the table below.

Command	Description
Ctrl+Alt+T	Open a terminal
pwd	Gives the present working directory
ls	Lists all the files in the current directory
cd< *some_directory* >	Enter < *some_directory* >
cd ..	Go to parent directory (one directory up)
cd	Takes you to home directory
mkdir	Creates a folder or a directory
rm< *some_file* >	Deletes < *some_file* >
rm -r < *some_directory* >	Recursively removes all the contents in a directory (use with caution)

Table A.1.: List of some useful Linux terminal commands

A.2. Introduction to ROOT

ROOT is a framework widely used in high-energy physics for data analysis, visualization, and storage. It provides a comprehensive set of tools and libraries for manipulating and analyzing large datasets efficiently. In this introduction, we will cover some basic operations using ROOT through the terminal. Table A.2 shows a small list of ROOT commands which would be helpful for this lab exercise.

Command	Description
root root -l	open ROOT opens ROOT, avoids a splash screen with a software version displayed on the start-up
.q	Exit from the ROOT environment
.?	Show help in interactive ROOT
root -l file.root	Open ROOT and load a file
new TBrowser	Inspect the graphs/histograms in a file in an interactive way
.ls	list the present ROOT directory
$histo->GetRMS()$	Provides the RMS of the histogram

Table A.2.: Some useful ROOT commands

In newer ROOT versions installed using modern C++, a new kind of TBrowser is enabled by default. This opens a web browser window and uses that at the interface, which often has anegative performance impact. To use the old TBrowser, one can start ROOT with a flag, e.g. via `root --web=off`.

Opening a ROOT File

To open a ROOT file using the terminal, you can use the following command:

```
root -l filename.root
```

Replace `filename.root` with the name of the ROOT file you want to open. The `-l` option launches ROOT in interactive mode.

Browsing in a ROOT File

Once you have opened a ROOT file, you can navigate its contents using the terminal. The basic command to browse through the file is:

```
TFile* file = new TFile("filename.root");
file->ls();
```

This creates a pointer to the ROOT file and displays a list of objects contained within the file, including histograms, trees, and other data structures.
To move within the ROOT file and access objects in subfolders, you can use the `cd()` function. Here's an example of how to move into a subfolder and list its contents:

```
TDirectory* subfolder = (TDirectory*)file->Get("path/to/subfolder");
subfolder->cd();
subfolder->ls();
```

Replace `path/to/subfolder` with the actual path to the subfolder within the ROOT file. The `Get()` function is used to retrieve the subfolder, and `cd()` is used to change the current directory to the subfolder. Finally, `ls()` lists the objects within the subfolder. By using these commands, you can explore and navigate through the directory structure of the ROOT file, including accessing objects within subfolders. This allows you to interact with specific parts of the file for further analysis and visualization.

Using TBrowser

TBrowser is a graphical tool in ROOT that allows you to explore the contents of a ROOT file interactively, and execute commands directly within the browser. To open TBrowser and access the command prompt, you can use the following commands:

```
TFile* file = new TFile("filename.root");
TBrowser* browser = new TBrowser("browser", file);
```

Replace `filename.root` with the name of your ROOT file. These commands create a pointer to the ROOT file and open a TBrowser, displaying the file's contents in a graphical interface.
Once a TBrowser is open, you can navigate through the file's directory structure by expanding folders and double-clicking objects to view them. Additionally, you can use the command prompt within the TBrowser to execute commands.
To access the command prompt, scroll down to the bottom of the TBrowser window. There, you'll find the "Command (local)" prompt. You can type in ROOT commands directly in this prompt and execute them by pressing Enter.
For example, you can execute commands like:

```
TH1F* histogram = (TH1F*)file->Get("histogramName");
histogram->Draw();
```

In this example, we retrieve a histogram named `histogramName` from the ROOT file and display it using the `Draw()` function.
Using the command prompt in a TBrowser provides a convenient way to explore the file's contents, visualize objects, and execute commands within the same interface.
Remember to close the TBrowser and free up memory when you're done by executing the following command:

```
delete browser;
```

This will ensure that the resources used by the TBrowser are properly released. The TBrowser is automatically closed when closing ROOT from the terminal window with `.q`.

Getting the RMS of a Histogram

To calculate the RMS (Root Mean Square) of a histogram located in a subfolder of a ROOT file using the terminal, you can use the following commands:

```
TFile* file = new TFile("filename.root");
TH1F* histogram = (TH1F*)file->Get("subfolder/histogramName");
Double_t rms = histogram->GetRMS();
```

Replace `filename.root` with the name of your ROOT file, `subfolder` with the name of the subfolder containing the histogram, and `histogramName` with the name of the histogram you want to analyze. The `Get()` function is used to access the histogram object within the specified subfolder, and `GetRMS()` calculates the RMS value of the histogram.
By providing the appropriate subfolder path along with the histogram name, you can access and analyze histograms located in specific directories within the ROOT file.

A.3. Files used in this exercise

detector.conf

```
[mydetector]
type = "cmsp1"
position = 0mm 0mm 0mm
orientation = 0deg 0deg 0deg
```

visual.conf

```
[Allpix]
detectors_file = "detector.conf"
log_level = "INFO"
multithreading = true
output_directory = "output"
root_file = "visual"
number_of_events = 1

# Generate the geometry of the detector setup for GEant4:
[GeometryBuilderGeant4]

# Define a magnetic field:
#[Ignore]
[MagneticFieldReader]
model="constant"
magnetic_field = 0T 0T 0T
```

```
# Define the beam and physics parameters for Geant4:
[DepositionGeant4]
physics_list = QGSP_BERT
number_of_particles = 1

particle_type = "e-"

source_energy = 5GeV
source_type = "beam"
source_position = 0 0 -10mm

beam_size = 1.5mm
beam_divergence = 1mrad 1mrad
beam_direction = 0 0 1

max_step_length = 1um
output_plots = true

# Run the visualization
[VisualizationGeant4]
mode = "terminal"
driver = "OGL"
```

start.conf

```
[Allpix]
detectors_file = "detector.conf"
log_level = "INFO"
multithreading = true
output_directory = "output"
root_file = "QDC0bit"
number_of_events = 5000

# Generate the geometry of the detector setup for GEant4:
[GeometryBuilderGeant4]

# Define a magnetic field:
[MagneticFieldReader]
model="constant"
magnetic_field = 0T 0T 0T

# Define the beam and physics parameters for Geant4:
[DepositionGeant4]
```

```
physics_list = QGSP_BERT
number_of_particles = 1

particle_type = "e-"

source_energy = 5GeV
source_type = "beam"
source_position = 0 0 -10mm

beam_size = 1.5mm
beam_divergence = 1mrad 1mrad
beam_direction = 0 0 1

max_step_length = 1um
output_plots = true

# Read or create an electric field in the sensor:
[ElectricFieldReader]
model = "linear"
voltage = -150V
depletion_voltage = -50V
deplete_from_implants = false
output_plots = true

# Propagation of the charge carriers in the sensor:
[GenericPropagation]
temperature = 293K
charge_per_step = 20

timestep_min = 0.05ns
timestep_max = 0.8ns

propagate_electrons = true
propagate_holes = false

output_plots = true
output_linegraphs = false
output_animations = false


# Transfer the propagated charges to the pixels:
[SimpleTransfer]
# maximum distance of a propagated charge from the surface of the
↪  sensor to be collected:
```

```
max_depth_distance = 10um
output_plots = true

# Simulate the front-end electronics of the readout chip:
[DefaultDigitizer]
output_plots = true
electronics_noise = 110e
threshold = 500e
threshold_smearing = 30e
qdc_smearing = 200e
qdc_slope = 200

# Play around with the QDC resolution
qdc_resolution = 1

# Create histograms for the detector and save them to file
[DetectorHistogrammer]
matching_cut = 200um 200um
track_resolution = 0 0
```

replay.conf

```
[Allpix]
detectors_file = "detector.conf"
log_level = "WARNING"
multithreading = true
output_directory = "output"
root_file = "QDC0bit"
number_of_events = 50000

# Read in previously simulated particles and energy deposits:
[ROOTObjectReader]
ignore_seed_mismatch = true
file_name = "../data/Tutorial_0deg_0T_data.root"
include = "PixelCharge", "MCParticle"

# Simulate the front-end electronics of the readout chip:
[DefaultDigitizer]
output_plots = true
electronics_noise = 110e
threshold = 500e
threshold_smearing = 30e
qdc_smearing = 200e
```

```
qdc_slope = 200
output_plots = true

# Play around with the QDC resolution
qdc_resolution = 1

# Create histograms for the detector and save them to file
[DetectorHistogrammer]
matching_cut = 200um 200um
track_resolution = 0 0
```

Bibliography

1. Brun, R. & Rademakers, F. ROOT — An object oriented data analysis framework. *Nucl. Instrum. Methods Phys. Res., Sect. A* **389.** New Computing Techniques in Physics Research V, 81–86. ISSN: 0168-9002. http://www.sciencedirect.com/science/article/pii/S016890029700048X (1997).
2. Spannagel, S. *et al.* Allpix^2: A modular simulation framework for silicon detectors. *Nucl. Instrum. Methods Phys. Res., Sect. A* **901,** 164–172. ISSN: 0168-9002. http://dx.doi.org/10.1016/j.nima.2018.06.020 (Sept. 2018).
3. http://www.physics.ucla.edu/~hauser/classes/180F/ROOTCheat.pdf, viewed on August 18, 2023.
4. Schütze, P., Spannagel, S., Wolters, K. & Lachnit, S. *Allpix^2 User Manual* v3.0.1 (Oct. 2020). https://project-allpix-squared.web.cern.ch/project-allpix-squared/usermanual/allpix-manual-3.0.1.pdf.
5. Kolanoski, H. & Wermes, N. *Teilchendetektoren - Grundlagen und Anwendungen* ISBN: 978-3-662-45350-6 (Springer Spektrum, Berlin, Heidelberg, 2016).
6. Kolanoski, H. & Wermes, N. *Particle Detectors, fundamentals and applications* ISBN: 9780198858362 (Oxford University Press, Sept. 2020).
7. L.Rossi *et al. Pixel Detectors,From Fundamentals to Applications* ISBN: 978-3-540-28333-1 (Springer-Verlag Berlin Heidelberg, 2006).
8. Agostinelli, S. *et al.* Geant4 — a simulation toolkit. *Nucl. Instrum. Methods Phys. Res., Sect. A* **506,** 250–303. ISSN: 0168-9002. https://www.sciencedirect.com/science/article/pii/S0168900203013688 (2003).
9. Allison, J. *et al.* Geant4 developments and applications. *IEEE Transactions on Nuclear Science* **53,** 270–278 (2006).
10. Allison, J. *et al.* Recent developments in Geant4. *Nucl. Instrum. Methods Phys. Res., Sect. A* **835,** 186–225. ISSN: 0168-9002. https://www.sciencedirect.com/science/article/pii/S0168900216306957 (2016).
11. https://ubuntu.com/tutorials/command-line-for-beginners, viewed on August 18, 2023.