ROOT Analysis of Test Beam Data

Pedestal and Noise Determination, Cluster Finding and Position Reconstruction

Gero Flucke

January 29th, 2010

Introduction

GEANT 4

- Learned how to simulate particle interaction with detectors.
- Important for planning of detectors.
- Real performance of devices to be demonstrated e.g. in test beams.

This tutorial

- Aspects of a test beam analysis for silicon sensors.
- Using the computing tool common to large HEP experiments: ROOT.
- Also using truth information to understand basic properties easily.
- Note: Not all physics effects simulated:
  charge diffusion is beyond GEANT4
  (but small effect according to T. Rohe's presentation)!

Some Notes on ROOT

ROOT

- is an interactive tool,
- also largely used in batch processing.
- We will start with a small fully interactive part.
- Then we'll work in the "compiled macro mode."

Macros

- (interprete or) compile C++ macro code

$\Rightarrow$ methods and classes available on command line

- interprete C++ (-like) code: error prone, not 100% C++, etc.
- therefore we **compile** and load C++ code
  - 'root macro.C+' == 'root, .x macro.C+'
  - .x macro.C+ == .L macro.C+ ; 'macro()'
  - i.e. calling method with name of macro (other methods from macro.C available as well)
  - 'macro.C++' first compiles the macro, then loads
  - 'macro.C+' same, **but** skips compilation if macro did not change
  - $\Rightarrow$ will use frequently

More About ROOT

Debug Help

- Working with macros and reloading them: ROOT crashes from time to time

$\Rightarrow$ usual '.q' to quit $\Rightarrow$ '.qqq' or '.qqqqq'

- if all fails: 'killall root.exe' on another shell
- sometimes compilation screwed up:
  $\Rightarrow$ 'rm *so' to get rid off old libraries
- ROOT documentation of ROOT classes:
  http://root.cern.ch/root/html522/<classname>

Even More About ROOT

Useful Features

- History of commands: using up/down keys.
- Search in history:
  CTR-r <type what you search in history>
- tab-completion for names of variables and methods:
  try out: 'new TB<TAB>' or even 'new TBr<TAB>
- help on method arguments:
  myfunction(<TAB>

C++ Standard Template Library (STL)

- Useful C++ Classes known by compiler.
- Convenient to use.
- Class names start with 'std::'.
- Will make use of (very) basic features of STL:
  - std::vector<someType>: similar to plain C-arry of <something>
    ```
    std::vector<float> vec(2);
    vec[0] = vec[1] = 1.5;
    ```
  - std::pair<aFirstType,aSecondType>: just group two things : `item`
    ```
    std::pair<int,double> aPair(1, -7.3);
    aPair.first = 17;
    aPair.second = 40.9;
    ```

  Complete definition of classes:
  http://www.sgi.com/tech/stl/table_of_contents.html

Tutorial: Preparational Steps

- **Create working directory** and 'go there':
  - cd school
  - mkdir SiTelescope
  - cd SiTelescope
- **Large test beam data samples** of GEANT tasks:
  - created centrally for analysis
  - reside /afs/desy.de/user/s/school01/public/datafiles/*root
  - analysis tasks assume files to be in working directory
  - copy locally into /scratch and link from there:

```
mkdir /scratch/files
cp /afs/desy.de/user/s/school01/public/datafiles/*.root
```

(continuation of previous line ...) `/scratch/files`

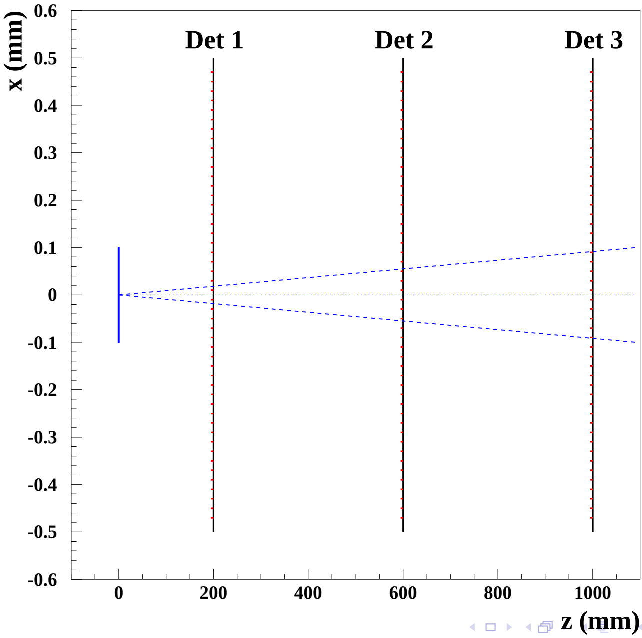
```
for i in /scratch/files/*.root ; do ln -s $i; done
```

- We want to change some default **ROOT settings**:
  - e.g. done by file called .rootlogon.C in home directory (i.e. ~)

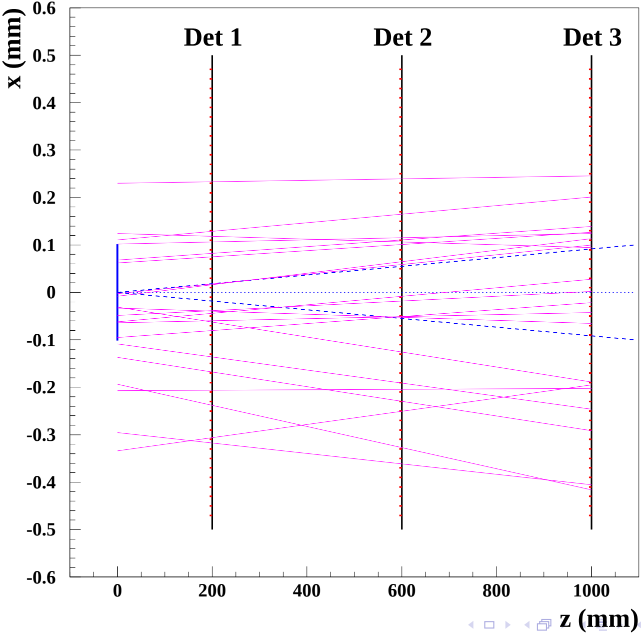
$\Rightarrow$ `cp` <path as above>`/SiTelescope/.rootlogon.C ~/.`

Questions?

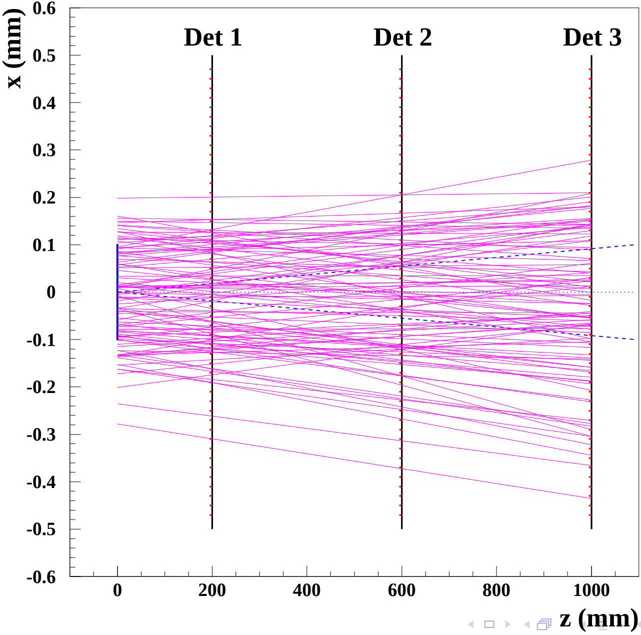
Reminder of our Setup



Reminder of our Setup with Tracks



Reminder of our Setup with More Tracks



Task 2.1a): Pedestal and Noise

Motivations

- In real life, measurements come from ADC:
  Pedestal is what you get out without signal.
- There will be noise:
  Spread around the pedestal.
- In principal, for each strip of each sensor separately.

⇒ We simplify: Treat all 48 strips per sensor together.

Steps: Interactive ROOT session (in SiTelescope directory)

- Open file with TTree from noise run: `root tree_Noise.root`
- Type `new TBrowser`
- double click 'ROOT files'
- double click 'tree_Noise.root'
- double click 'SiTelescope', see variables of the TTree
- click browser button with tooltip 'Details'

Tree Variables

```
Float_t signal1[48]; // signal in strips of sensor 1 [e]
Float_t signal2[48]; // signal in strips of sensor 2 [e]
Float_t signal3[48]; // signal in strips of sensor 3 [e]
Float_t truthPos1; // true x position of particle in sensor 1 [mm]
Float_t truthPos2; // true x position of particle in sensor 2 [mm]
Float_t truthPos3; // true x position of particle in sensor 3 [mm]
Float_t truthE1; // true energy deposited in sensor 1 [MeV]
Float_t truthE2; // true energy deposited in sensor 2 [MeV]
Float_t truthE3; // true energy deposited in sensor 3 [MeV]
Float_t truthPos0; // true particle position in x at z=0 [mm]
Float_t truthAngle0;// true angle in xz-plane z=0 [mrad]]
```

Remark:

truthPos<1|2|3> is in global coordinates,
it does **not** rotate with the DUT!

Steps: Interactive ROOT Session (ctd.)

- double click on `signal1`: draw signal (=noise)

⇒ Noise around pedestal: **Mean and RMS**
(Why 480000 entries - we have 100000 events?)

- Interactively fit to Gaus function:
  - right click on histogram line ⇒ FitPanel
  - choose function `gausn`
  - fit and see fitted results for Mean and σ
- Note down numbers and repeat for signal2 and signal3.
- **Bonus:**
  To see noise of a single strip number 6, type on command line
  `SiTelescope->Draw("signal1[5]")`

Preparation of Analysis Skeleton

- Start root with file: `root tree_SiTelescope_2GeV.root`
- TTree object accessible on command line via its name.
- Use command line to create analysis skeleton:
  - `SiTelescope<press RETURN>` (to see that it exists and its type)
  - `SiTelescope->MakeSelector()`
- Quit ROOT and read comments in SiTelescope.C (ignore lines on PROOF).

Program Flow

- Add print statements to SiTelescope::Begin, SiTelescope::Process and SiTelescope::Terminate, e.g.

```
std::cout « "Begin " « GetOption() « std::endl;
std::cout « "Process " « entry « std::endl;
```

- Again start ROOT with a TTree file:

```
root tree_SiTelescope_2GeV.root
```

- See how the skeleton is called by typing in ROOT:

```
SiTelescope->Process("SiTelescope.C",
                     "myoption", 10);//10 events
```

- try also with "SiTelescope.C+" ⇒ probably you 'forgot' to add `#include <iostream>` needed for `cout` in compiled code...
- **NOTE** This error tells you that your C++ was wrong:

```
Error in <TSelector::GetSelector>:  file
SiTelescope.C+ does not have a valid class deriving
from TSelector
```

  ⇒ Look at the output above this error to diagnose.

Use of SiTelescope Class

- We will
  - book histograms in Begin(),
  - fill them in Process(),
  - draw/store/fit them in Terminate()
- Access to variables of the events:
  - available as generated data members of the `SiTelescope` class, e.g. see `truthPos1` in SiTelescope.hs.
  - Try to print it (add `std::`):
    `cout«"Entry"«entry«",pos1 is"« truthPos1 « endl;`
  - **But:** All the same? Yes, but wrong!
  - In Process, we need
    `this->GetEntry(entry);`

General Structure

- Subdirectory **template21b** with files prepared for task 2.1b) will be at /afs/desy.de/user/s/school05/public/SiTelescope
- Copy them to your SiTelescope directory, enter the directory:
  cd SiTelescope
  cp -r /afs/desy.de/user/s/school05/public/SiTelescope/template21b .
  cd template21b
- Comments inside what to do/add.
- Search for '**FILL ME**' to see what and where to add there.
- Play around, ask questions.
- After some time, **solution21b** will appear.
- We will discuss the resulting distributions.
- template21c will be based on solution21b

Task 2.1b): Signal Distributions of First Five Events

Comments

After editing SiTelescope.C:

- `root ../tree_SiTelescope_2GeV.root`
- `SiTelescope->Process("SiTelescope.C+", "", 10)`

Task 2.1c): Cluster Finding

- A cluster is a number of subsequent strips that “fired”.
- The signal-to-noise ratio governs which strips have fired and form a cluster.
- A common algorithm with three thresholds θ
  - Any strip part of the cluster must fulfil $S/N > \theta_{strip} = 2.$
  - At least one strip is a seed with $S/N > \theta_{seed} = 3.$
  - The full cluster must fulfill $S/N = \frac{\sum_{strips} S_i}{\sqrt{\sum_{strips} N_i^2}} > \theta_{cluster} = 5.$

Comments

After editing SiTelescope.C:

- `root ../tree_SiTelescope_2GeV.root`
- `SiTelescope->Process("SiTelescope.C+")`

See number of clusters and their number of strips.

Task 2.1d): Cluster Position Reconstruction

- Simple algorithm: centre of gravity of cluster charges.
- Makes use of charge sharing.
- Other algorithms superior for e.g. incident angles.

Comments

After editing SiTelescope.C:

- `root ../tree_SiTelescope_2GeV.root`
- `SiTelescope->Process("SiTelescope.C+")`

See position in numbers of strips: Nothing on last strip!
Bug fix for cluster algorithm in solution21d...

Task 2.2a): Cluster Charge Distribution vs Momentum

- Task split:
  - a1) Implement charge distribution: template22a1 and solution22a1
  - a2) New macro testBeam.C to combine results of different momenta: template22a2 and solution22a2

Comments to a1)

After editing SiTelescope.C:

- `root ../tree_SiTelescope_2GeV.root`
- `SiTelescope->Process("SiTelescope.C+")`

Look at charge distribution in log-scale and compare to what happens if you remove the cut on events with > 1 clusters.

Task 2.2a): Cluster Charge Distribution vs Momentum

Comments to a2)

- Look what changed in SiTelescope to store all histograms in files with names like 'hists_2GeV.root' for
  - `root ../tree_SiTelescope_2GeV.root`
  - `SiTelescope->Process("SiTelescope.C+", "2GeV")`
- You do not need to further edit this.

We have a new macro testBeam.C:

- New working horse to combine beam energies.
- Look at the structure and its main components, i.e. the methods singleTree(..), histFromFile(..) and testBeam().
- You can even run the full macro 'root testBeam.C+', . . .
  - but it will crash in the end,
  - so fix the method 'chargeDraw(..)'!
- When re-running: comment out calls to singleTree(..) in testBeam().

Task 2.2b): “Landau” Fit to Charge Distribution

- Very thin sensors do not have Landau distributions of the deposited charge.
- But also thicker sensors as ours are not perfect Landau: We will fit the convolution of Gaus and Landau.

Comments

After editing testBeam.C:

- `root testBeam.C+`
- When re-running: comment out calls to singleTree(..) in testBeam().

- Charge deposition (i.e. energy loss): roughly Landau-distributed.
- How does the "average" energy loss as function of the momentum look like?
- What is the "average"? Here try
  - mean,
  - most probable value (MPV from Landau fit),
  - median.

Comments

After editing testBeam.C:

- `root testBeam.C+`
- When re-running: comment out calls to singleTree(..) in testBeam().

Why is our median a step function?

Thanks for attention!

Have fun at the Dinner!

See you tomorrow morning:
Multiple Scattering and Resolutions