GEANT4 & ROOT

Computing Workshop 9/6/2024

ericking@temple.edu

TEMPLE UNIVERSITY

About Me

- Work at Temple University
- Postdoctoral Fellow
    - PhD, Physics
    - MSci, Computation/Simulation using HTP/HPC.
- Working on MOLLER
    - Moller Polarimetry
    - Ferrous Materials Backgrounds

Outline for Today

- Brief overview of C++ nomenclature/terminology I may use
- Geant4 Overview
- ROOT Overview
- Hands-on Session

C++ Classes

- Class
  - Defines some sort of object/struct

```
... headers ...

class Person {

};
```

C++ Classes (cont'd)

- Class
  - Defines some sort of object/struct
  - Contains access/members/attributes
    - Access (public, private, protected)

```
… headers …

class Person {
    private:

          …
    public:

};
```

C++ Classes (cont'd)

- Class
    - Defines some sort of object/struct
    - Contains access/members/attributes
        - Access (public, private, protected)
        - Variables/Values

```
… headers …

class Person {
    private:
        string firstname;
        string lastname;
        int birthmonth;
        int birthday;
        int birthyear;


    public:









};
```

C++ Classes (cont'd)

- Class
  - Defines some sort of object/struct
  - Contains access/members/attributes
    - Access (public, private, protected)
    - Variables/Values
    - Constructors

```
… headers …

class Person {
    private:
        string firstname;
        string lastname;
        int birthmonth;
        int birthday;
        int birthyear;


    public:
        Person() {}
        Person(string f, string l, int m, int d, int y) :
          firstname(f), lastname(l), birthmonth(m),
          birthday(d), birthyear(y) { }




};
```

C++ Classes (cont'd)

- Class
  - Defines some sort of object/struct
  - Contains access/members/attributes
    - Access (public, private, protected)
    - Variables/Values
    - Constructors
    - Functions/methods that operate on that structure/object

```
… headers …

class Person {
    private:
        string firstname;
        string lastname;
        int birthmonth;
        int birthday;
        int birthyear;
        SetFirstName(string fname){firstname = fname;}
        SetLastName(string lname){lastname = lname;}
        …
    public:
        Person() {}
        Person(string f, string l, int m, int d, int y) :
          firstname(f), lastname(l), birthmonth(m),
          birthday(d), birthyear(y) { }

        void printbirthdate(){
          cout << birthmonth << "/"
               << birthday << "/"
               << birthyear << endl;
        };
        …
};
```

C++ Class Inheritance

➢ Critical part of C++ for any application developer and this is used extensively in Geant4 applications

- Can create a user-defined **derived class** which *inherits* from a **base class**

- Allows for **addition of members** to the base class.

```
… headers …
#include “Person.hh”

class MyPerson : public Person {
    private:
        string fathersname;
        string mothersname;

        SetMothersName(string s){mothersname = s;}
        SetFathersName(string s){fathersname = s;}
        …

    public:

    …

};
```

Geant4

What is Geant4?

- Geant4 is a software toolkit for particle/nuclear physics Monte Carlo simulations
  - **GE**ometry **AN**d **T**racking
  - Toolkit ⇒ Geant4 doesn't do anything on its own.
  - Applications ⇒ You build the geometry and specify the physics
- Geometry:
  - Specify your own – basic toy models to very complex geometries.
- Tracking:
  - Internal tracking (Geant4) and Sensitive Detector / Hits Collection output
    - Get the information that you want
  - User ability to terminate tracks of no interest [time saver]
  - User specified data output
    - Generally ROOT but can be anything you want that you can code for

Useful Geant4 Documentation

- ➢ Geant4 Developers Guide [here]
- ➢ Geant4 Installation Guide [here]
- ➢ Introduction to Geant4 [here]

Geant4 Framework

- Central functionality/classes in Geant4
    - Main application file – initialize run manager, classes, visualization, etc.
    - RunAction – starts and ends runs.
    - DetectorConstruction
        - In-line native Geant4 coding
            - Requires a recompile for changes, simple troubleshooting.
        - GDML
            - XML macro specified file read-in, very tricky troubleshooting
    - EventAction – starts and ends events, allows user specified actions.
    - SteppingAction – What to do at end of steps, allows user-specified action
    - Detectors & Hits – Assign volumes as sensitive detectors and record hit data.

Run / RunAction() – Analogous to a physics 'run'

Run

- A specified total number of events to be simulated
- Geometry for simulation is 'built'
- Physics processes are set
  [*I'll have slide that touches on this*]

➢ Once a run has started geometry can not be changed and physics cannot be changed.

RunAction

- BeginOfRunAction()
  - Start your Input-Output [IO] class
  - Create files, create data structures
- EndOfRunAction()
  - Close out your data files
    - ROOT files
    - Histograms
    - CSV files
    - etc...

Geant4: DetectorConstruction – Materials

G4Elements

- Requires “name”, “symbol”, Z, and A
- `G4Element * elC = new G4Element(“carbon”, “C”, 6, 12.01*g/mole)`
- Natural isotope ratios will be added according to Z

G4Isotopes

- Can create specific isotope mixtures
- ```
  G4Isotope * isoC13 = new G4Isotope(name=“C13”,iz=6,n=7);
  G4Element * elC13 = new G4Element(“C13”, “C13”, numisotopes=1);
  elC13->AddIsotope(isoC13,abundance=100.*percent);
  ```


  ```

Geant4: DetectorConstruction – Materials (cont’d)

G4Materials

Pre-defined list of materials available [here] or can be user-defined.

Al and Ar used in later slide example

- Pre-defined → Call NIST manager, find material, assign to G4Material object
  - ```
    G4NistManager * manager = G4NistManager::Instance();
    G4Material * Al = manager->FindOrBuildMaterial(“G4_Aluminum”);
    G4Material * Ar = manager->FindOrBuildMaterial(“G4_Argon”);
    ```

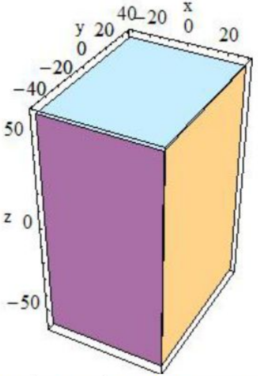
- User-defined material → Define elements, material,
  - ```
    G4Element* el_i = new G4Element("Iodine","I", 53,126.9*g/mole);
    G4Element* el_cs = new G4Element("Cesium","Cs",55,132.9*g/mole);
    G4Material* mat_csi = new G4Material("CsI",4.51*g/cm3,2);
    mat_csi->AddElement(el_i,1);
    mat_csi->AddElement(el_cs,1);
    ```

Geant4: DetectorConstruction – Solids

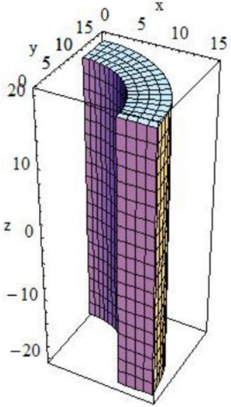
- Various solids are defined in Geant4:
  - G4Box
  - G4Tubs
  - G4Para (parallelepiped)
  - G4Sphere
    - G4Orb (solid sphere)
  - ... and more.

- See section 4.1.2 of the Geant4 Developers Guide for full list.

```
G4Box(const G4String& pName,
                G4double  pX,
                G4double  pY,
                G4double  pZ)
```



pX	half length in X	pY	half length in Y	pZ	half length in Z



```
G4Tubs(const G4String& pName,
                 G4double  pRMin,
                 G4double  pRMax,
                 G4double  pDz,
                 G4double  pSPhi,
                 G4double  pDPhi)
```

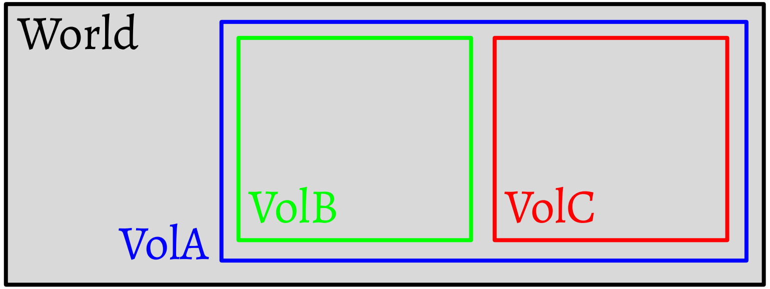
pRMin	Inner radius	pRMax	Outer radius
pDz	Half length in Z	pSPhi	Starting phi angle in radians
pDPhi	Angle of the segment in radians		

*Images, constructors and desc's adapted from Geant4 Developers Guide.

Geant4: DetectorConstruction – Volumes

- Geometries in Geant4 comprise of a number of volumes.
- The largest volume is called the **World** volume.
- Volumes are created and placed inside other volumes.
- All volumes must be fully contained in the **World** volume

- To create and place a volume we must:
  - Define or pick materials
  - Define a solid
  - Define a logical volume
  - Define a physical volume



⇒ The key concept here is proper volume nesting ⇐

$(\text{VolB} \ \& \ \text{VolC}) \subset \text{VolA} \subset \text{World}$ and $(\text{VolB} \cap \text{VolC}) = \emptyset$

Geant4: DetectorConstruction – Volumes (Cont'd)

Previously defined materials

(1) Define Shapes as previously described:
   (a) G4Box – **worldBox**
   (b) G4Tubs – **trackerTube**

(2) Define logical volumes
   (a) worldLog
   (b) trackerLog

(3) Create physical volume

```
G4LogicalVolume* worldLog
   = new G4LogicalVolume(worldBox, Ar, "World");
```

```
G4LogicalVolume* trackerLog
  = new G4LogicalVolume(trackerTube, Al, "Tracker");
```

```
G4double pos_x = -1.0*meter;
G4double pos_y =  0.0*meter;
G4double pos_z =  0.0*meter;

G4VPhysicalVolume* trackerPhys
  = new G4PVPlacement(0,
                      G4ThreeVector(pos_x, pos_y, pos_z),
                      trackerLog,
                      "Tracker",
                      worldLog,
                      false,
                      0);
```

```
G4VPhysicalVolume* physVolume = new G4PVPlacement(
    0,                          // rotation, 0 = none
    G4ThreeVector(0.,0.,0.),    // translation position
    logicalVolIdentifier,       // assoc'd logical volume
    "physVolName",              // phys vol name
    0,                          // mother volume, 0=world
    false,                      // no boolean operations
    0,                          // copy number
    true                        // overlap checker
);
```

*Code examples adapted from Geant4 Developers Guide.

Geant4: Particle Generator / PrimaryGeneratorAction()

Particle Generator

- Creates primary particles
- In practice, Geant4 can also be used to specify
- Can be simple – e.g. an electron at 10.6*GeV fired off in a particular direction
- Can be far more complicated
  - Monte carlo distributions to energy and trajectories of primaries.
  - Specific event type generator – e.g. a Moller electron generator.
  - Outside program that can provide the relevant primary particle information

PrimaryGeneratorAction()

- Runs at the beginning of each event.

Geant4: Event / EventAction()

Event

- An event processes all primary particles through the geometry
- There may be more than one primary particle
- Once the stack of primaries is empty the event is considered over.

BeginOfEventAction():

- User can define something to happen before next event takes place.

EndOfEventAction():

- Sort through HitsCollections
- Fill ROOT files
- Fill histograms
- Flush or clear variables

Geant4: Stepping / SteppingAction()

<u>Geant4 SteppingAction</u>

- Generally created as an inherited class from G4UserSteppingAction
- SteppingAction executed at the end of stepping
- Most useful application (IMHO):

```
aTrack->SetTrackStatus(fStopAndKill);
```

<u>Practical Uses</u>

- Killing particle tracks
- Save valuable computing time
    - Kill off particles of no interest
    - Kill off particles under detection thresholds
    - Kill off particles not relevant to simulation study

Main Application File

- The main application file generally resides in the base directory of the application.

- What's generally included
  - Typical items found in a main() script
    - How to read passed arguments/flags when the program is executed.
    - Construction of DetectorConstruction()
    - Geant4 UserAction classes initialized
    - Physics lists intialized
    - Visualization options – UIManager setup.
    - Setup for multi-threading

Geant4: Additional Information – Manager Classes

- G4RunManager
  - Register your geometry
  - Register your physics lists
  - Register your particle generator

- G4EventManager – handles events, pre-/post-event user actions
- G4SteppingManager – handles steps and user-specified actions
- G4TrackingManager – handles tracks and trajectories
- G4DetectorManager – handles declared sensitive detectors
- G4FieldManager – handles declared fields

Geant4 System of Units

- Use G4SystemOfUnits.hh header file
  - [C++] You can

- Values coded into Geant4 should include a unit
  - Provides consistency and eliminates unit errors

- Declaration of value:
  ➢ `G4double beamEnergy = 10.6 * GeV;`

- Conversion to unit:
  ➢ `G4cout << hitP / MeV << “MeV” << G4endl;`

Geant4: Physics / Processes

- Topic is worthy of its own discussion (and I'm not the person to lead it).
- Sometimes useful to create an inherited class from G4ModularPhysics
  - Helpful in more-advanced applications
    - Enabling and disabling certain physics options via local messenger (slide coming).
  - Have scintillating detectors and light guides???
    - You might want optical physics – G4OpticalPhysics
- FTFP_BERT / QGSP_BERT are commonly used lists
  - If low-energy neutron tracking is desired FTFP_BERT_HP / QGSP_BERT_HP can be selected.

Geant4: Messengers

- Constant recompiling is a nuisance
  - Large program → Longer recompile
  - Common source of computing error
- An often implemented solution to this is to write the application to accept commands and values
  - This is often referred to as a 'macro'

Options:

(1) Create inherited class from G4UImessenger and use G4UIcommand methods

(2) Utilize G4GenericMessenger to create your own in-class messenger options.

Additional Note: C++ <iostream>

➢ Text output is a common form of debugging and sanity checking.

- The use of C++ iostream **cout, endl, and cerr** is strongly discouraged

- G4UImanager class has members which handle G4-defined “iostream” objects
  - G4cout
  - G4endl
  - G4cerr

ROOT

Data Analysis Framework

Outline

- What is ROOT?
- What are ROOT Files
- What can you do with ROOT?
- Command Line Interface
  - I'll be painting in broad strokes as you'll have a formal hands-on session after this talk.
- Macros/Scripting
  - Using ROOT via macros and scripting.

GOAL: (Hopefully) Leaving you with a feeling for what you can do with ROOT and give you a conceptual leg up for the hand-on portion of the workshop.

What is ROOT? (cont'd)

- General framework for data analysis developed for particle physics
  - Based on data structure we call a ROOT file.
- C++ based OOP for scalable data and simulation-data analysis
  - Remoll automatically outputs ROOT files.
  - Importing CSV data into root is very easy.
    - Great option for data collected from in-lab hardware or EPICS data at JLab.
  - Available ROOT libraries to connect to databases and dataframes.
  - If you're a data-junkie bored on weekends ROOT is a great tool to churn through datasets.
- Main tool that will be used for Remoll simulation analysis and MOLLER data analysis.
  - ROOT is designed to handle large amounts of data

What is ROOT? (cont'd)

- Installation is *generally* straight forward
  - Pre-compiled binaries available on ROOT website for many, but not all Linux OS
    - !!! Ubuntu 22 pre-compiled binaries have given me issues; you may have to compile from scratch; after installing dependencies I had zero problems.
  - ⭐ Already available for you in the ifarm

- When you compile remoll you'll also get reroot which includes certain remoll class definitions
  - 👉 Use reroot for remoll simulation analysis...

What is ROOT? (cont'd)

- (Opinion) It's a extraordinarily easy-to-use framework for both data analysis and data presentation

- One can convert many data types of data files into ROOT files.
  - At the end of these slides is a very simple script for converting CSV file to ROOT

- I prefer to use ROOT for data visualization and plotting for many different types of data (not just particle physics).

In other words ⇒ A VALUABLE SKILL TO HAVE ⇐

What is ROOT? (cont'd)

- (Truth) There is a learning curve.
  - Familiarity with C++ will make learning ROOT easier.
    - There is PyRoot for the python-inclined.
  - Doing things in ROOT is how you'll learn.
    - Don't be shy asking for help.
    - Plenty of online resources.
  - You'll learn what you need to know as you go and eventually you'll become a ROOT 'expert'.
  - This is your tool to interpret simulation results and extract interesting physics from experimental data.

What are ROOT files?

Hierarchical structure of data.

⇒ Base of data structure is 'tree'

⇒ Data tree broken into 'branches'

⇒ Branches further divided into 'leaves'

- ROOT File
  - ↳ Data Tree #1
    - ↳ Branch 1
      - ↳ Leaves
    - ↳ Branch 2
      - ↳ Leaves
    - ↳ Branch 3,4,5…

What are ROOT files?

Hierarchical structure of data.

⇒ Base of data structure is 'tree'
⇒ Data tree broken into 'branches'
⇒ Branches further divided into 'leaves'

ROOT files can contain multiple trees

- ROOT File
  - ↳ Data Tree #1
    - ↳ Branch 1
      - ↳ Leaves
    - ↳ Branch 2
      - ↳ Leaves
    - ↳ Branch 3,4,5…
  - ↳ Data Tree #2
    - ↳ Branches
      - ↳ Leaves

What are ROOT files?

Hierarchical structure of data.

⇒ Base of data structure is ‘tree’
⇒ Data tree broken into ‘branches’
⇒ Branches further divided into ‘leaves’

ROOT files can contain multiple trees

ROOT files can hold other objects as well:

- Histograms
- Canvases
- Graphs
- etc

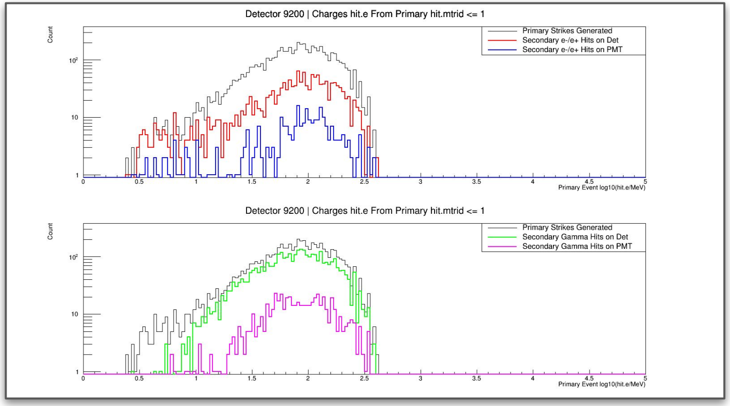
- ROOT File
  ↳ Data Tree #1
  - ↳ Branch 1
    - ↳ Leaves
  - ↳ Branch 2
    - ↳ Leaves
  - ↳ Branch 3,4,5...

  ↳ Data Tree #2
  - ↳ Branches
    - ↳ Leaves

  ↳ Object(s) ...

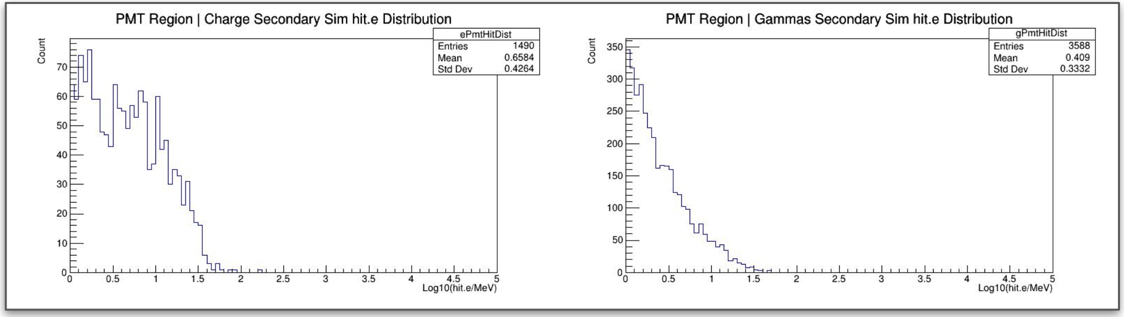
What can you do with ROOT? [Data Visualization]

- Draw histograms



remoll simulation data from ferrous materials studies.

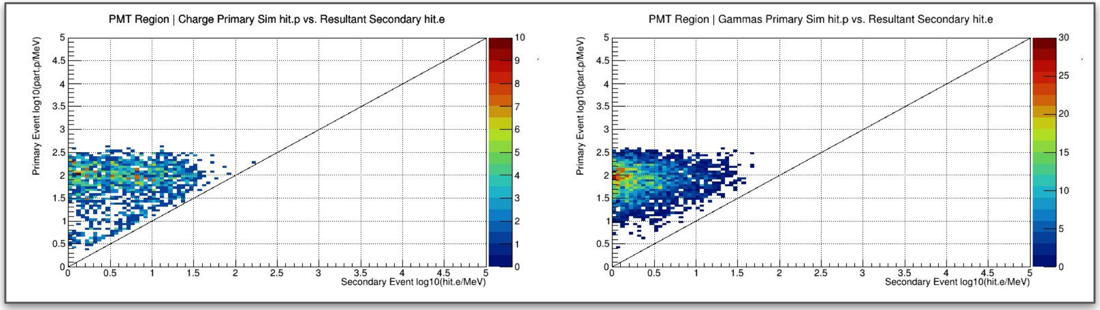
Multiple histograms can be overlaid to produce more informative plots.



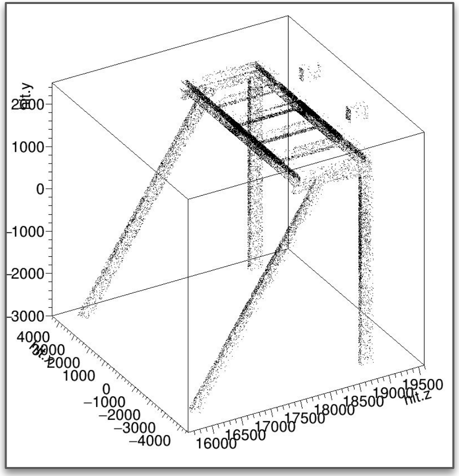
Or you can be very basic...

What can you do with ROOT? [Data Visualization]

- Draw histograms
- Draw scatter plots and heatmaps (2D hist with color!)



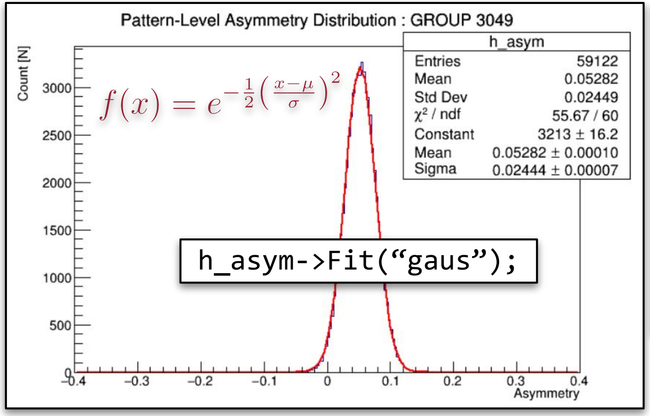
2D Heatmap of ferrous materials scattering PMT region backgrounds from collar 2 barite wall support (prelim' simulation).



3D Scatter plot of remoll simulation data of events that strike the collar 2 barite wall support structure for ferrous materials background studies.

What can you do with ROOT? [Data Visualization]

- Draw histograms
- Draw scatter plots and heatmaps (2D hist with color)
- Data Fitting
  - Predefined or custom functions



Experimental data fit; here to a gaussian.

f(x) = p0*exp(-0.5*((x-p1)/p2)^2)

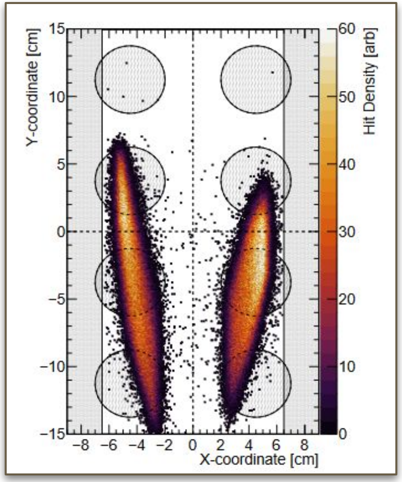
Fit returns:
$\chi 2$ / ndf
P0 ⇒ Constant: Amplitude of Gaussian
P1 ⇒ (Gaussian) Mean
P2 ⇒ (Gaussian) Sigma

```
root [8] H->Fit("gaus")
 FCN=32.6596 FROM MIGRAD      STATUS=CONVERGED       63 CALLS         64 TOTAL
                     EDM=2.2785e-09 STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                     STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     1.78988e+02   4.07137e+00   9.47909e-03  -1.50266e-05
   2  Mean         3.50379e-02   1.82819e-04   5.11155e-07  -1.29829e-01
   3  Sigma        9.57978e-03   1.26021e-04   1.00540e-05  -1.77126e-02
```

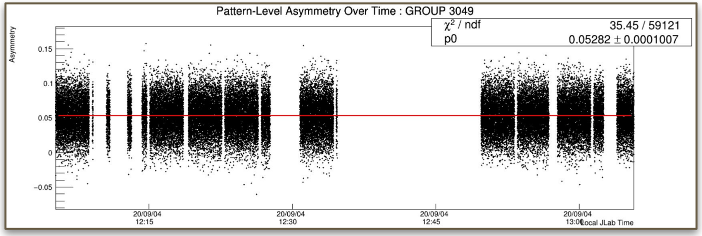
Note: Plot and text fit data not the same

What can you do with ROOT? [Data Visualization]

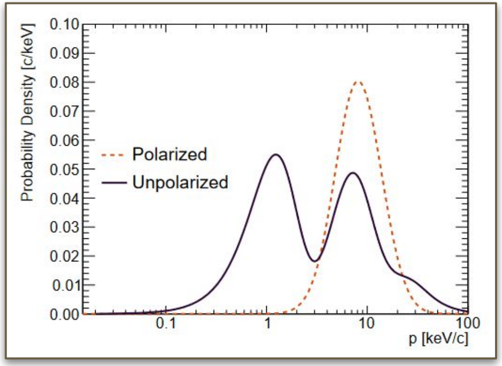
- Draw histograms
- Draw scatter plots and heatmaps
- Data Fitting
  - Predefined or custom functions
- **Data Visualization**
  - **Actual data**
  - **Simulated data**
  - **Imported data**
- **>> Publication ready plots <<**



Moller polarimeter simulated data. (almost publication plot)



Polarimetry – Moller QED asymmetry over time during CREX

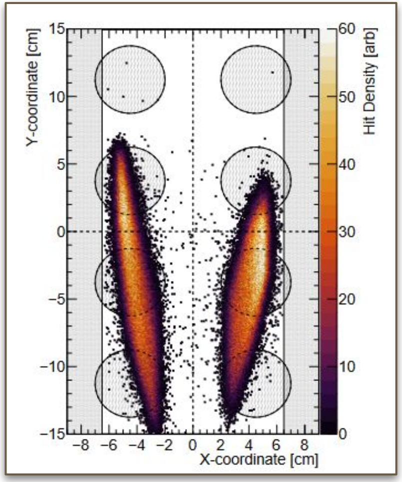


Imported CSV data of computed bulk Fe wavefunctions turned into TGraph object. (publication plot)

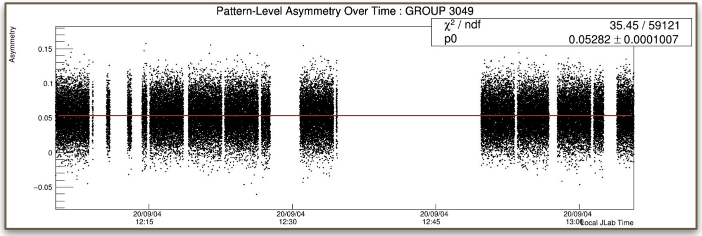
DE King, DC Jones, et al. – Moller Polarimetry for PREX-2 and CREX
10.1016/j.nima.2022.167506

What can you do with ROOT? [Data Visualization]

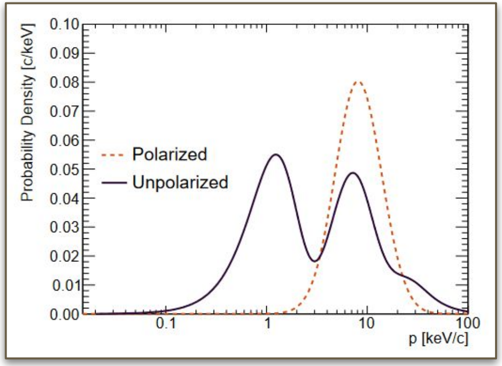
- Draw histograms
- Draw scatter plots and heatmaps
- Data Fitting
  - Predefined or custom functions
- **Data Visualization**
  - **Actual data**
  - **Simulated data**
  - **Imported data**
- **>> Publication ready plots <<**



Moller polarimeter simulated data. (almost publication plot)



Polarimetry – Moller QED asymmetry over time during CREX



Imported CSV data of computed bulk Fe wavefunctions turned into TGraph object. (publication plot)

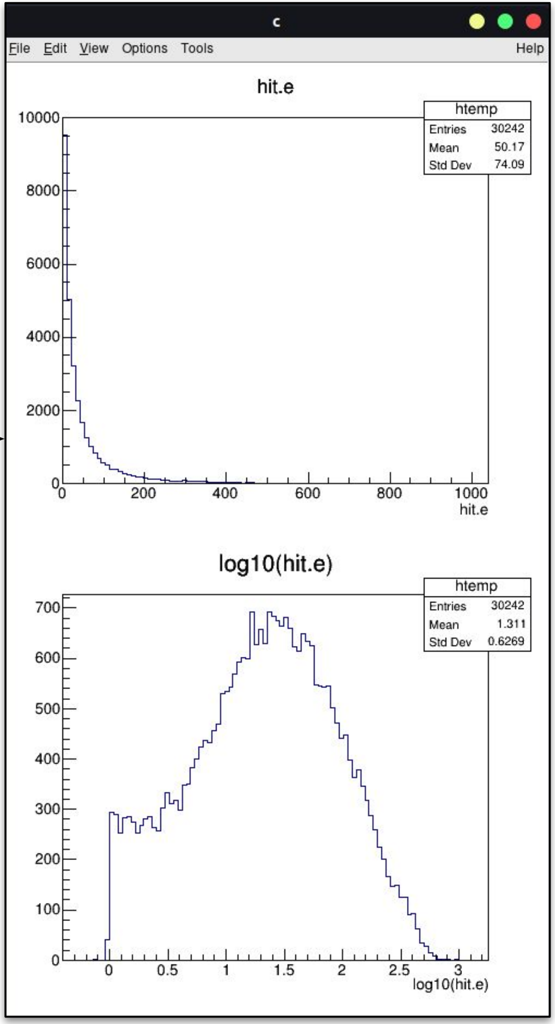
DE King, DC Jones, et al. – Moller Polarimetry for PREX-2 and CREX
10.1016/j.nima.2022.167506

Using ROOT

Three ways of using ROOT:

- Command Line Interface
  - Quick and dirty method of looking at data.
  - Make basic data cuts.
  - Make basic plots.

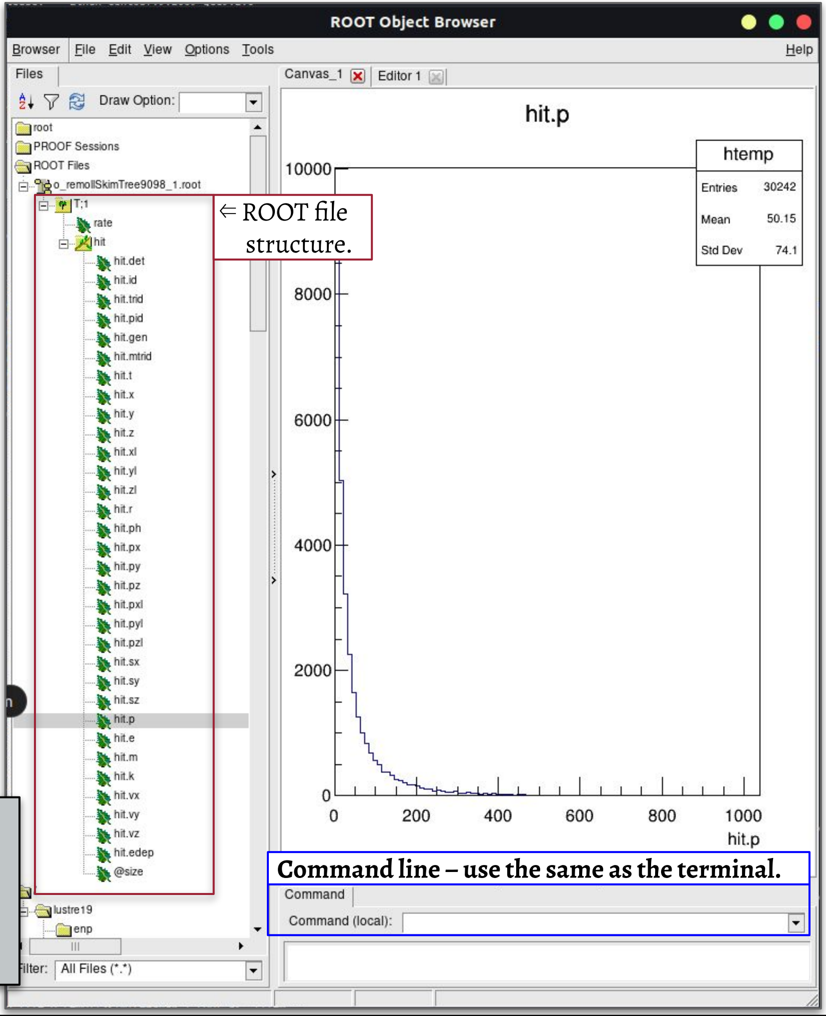
```
ifarm1802.jlab.org> ./reroot -l o_remollSkimTree9098_1.root
re-root [0]
Attaching file o_remollSkimTree9098_1.root as _file0...
(TFile *) 0x2380910
re-root [1] TCanvas* C = new TCanvas("c","c",500,900)
(TCanvas *) 0x2f15e10
re-root [2] C->Divide(1,2)
re-root [3] C->cd(1)
(TVirtualPad *) 0x3004570
re-root [4] T->Draw("hit.e")
re-root [5] C->cd(2)
(TVirtualPad *) 0x28fae00
re-root [6] T->Draw("log10(hit.e)")
re-root [7]
```



Using ROOT

Three ways of using ROOT:

- Command Line Interface
  - Quick and dirty method of looking at data.
  - Make basic data cuts.
  - Make basic plots.
- There is a GUI File Browser
  - This is perfectly fine when working locally, but a total nuisance over X11.
  - ☆ It's an easy way to see file structure and plot uncut data distributions.
  - Perhaps good for beginners.



I'll note that this is a thing that one can do.

I don't care for it much but it has its a couple perks; you may like it.

Using ROOT

Three ways of using ROOT:

- Command Line Interface
  - Quick and dirty method of looking at data.
  - Make basic data cuts.
  - Make basic plots.
- There is a GUI File Browser
  - This is perfectly fine when working locally, buta total nuisance over X11.
  - ☆ It's an easy way to see file structure and plot uncut data distributions.
  - Perhaps good for beginners.
- Macros/Scripting
  - Formal analysis – capable of complex data cuts.
  - Allows analysis work to be repeated easily.
  - Will require you to be comfortable with C++.

```
void skimTreeMulti(string fileList, string DetNums, Int_t gencut=0, int beamGen=1, int test=0){
  startTime = std::clock();
  std::ofstream fout;
  fout.open("ferrous_skimTree_results.txt");

  testRun    = test;
  generation = gencut;

  //////////////////////////////////////////////////////////////
  //// Detectors to be looked at
  std::stringstream ss(DetNums);
  while(ss.good()){
    string ss_parse;
    getline(ss,ss_parse,',');
    detectorNumbers.push_back( std::atoi(ss_parse.c_str()) );
    vector<Int_t> tempVec;
    for(Int_t g=0; g<=generation; g++) tempVec.push_back(0);
    detectorHitN.push_back( tempVec );
  }
  cout << "Detectors to be examined: ";
  for(Int_t k=0; k<detectorNumbers.size(); k++){
    if(k==0 && detectorNumbers.size()==1){
      cout << "{" << detectorNumbers[k] << "}";
    }else if(k==0){
      cout << "{" << detectorNumbers[k];
    }else if(k==(detectorNumbers.size()-1)){
      cout << "," << detectorNumbers[k] << "}" << endl;
    }else{
      cout << "," << detectorNumbers[k];
    }
  }
  cout << "Recording all particles whose mother is <=" << generation << endl;

  long nTotHits(0);
  int nFiles(0);

  if(fileList==""){
    cout<<"\t did not find input file. Quitting!"<<endl;
    return;
  }

  //////////////////////////////////////////////////////////////
  //// Create output files and trees
  otree     = new TTree("T", "ferrous skim tree");
  b_rate    = otree->Branch("rate", &newrate);
  b_hit     = otree->Branch("hit",  &newhit );
  for(Int_t i = 0; i < detectorNumbers.size(); i++){
    cout << "Creating pointer to TFile named " << Form("o_remollSkimTree%d_%i.root",detectorNumbers[i],gencut) << endl;
    outputFiles.push_back( new TFile(Form("o_remollSkimTree%d_%i.root",detectorNumbers[i],gencut),"RECREATE") );
    otree->SetObject("T",Form("ferrous skim tree %i",detectorNumbers[i]));
    //otree->SetObject(Form("T%i",detectorNumbers[i]),Form("ferrous skim tree %i",detectorNumbers[i])); //for debugging
    outputTrees.push_back( otree->CloneTree(0) ); //Create a clone of tree and copy 0 entries
  }
  otree->SetObject("T","ferrous skim tree");

  //////////////////////////////////////////////////////////////
  //// Keep code from original script
  if( fileList.find(".root") < fileList.size() ){
    //cout<<"Processing single file:\n\t"<<fileList<<endl;
    if(beamGen){
      scaleRate = getEvents(fileList);
    }
    nTotHits+=processOne(fileList);
    nFiles=1;
  }else{
```

Showing example with skimmed simulated hit data from my work.
remoll files contain much more information

Using ROOT – Command Line Interface

Command line interface uses CLING (a C++ interpreter)

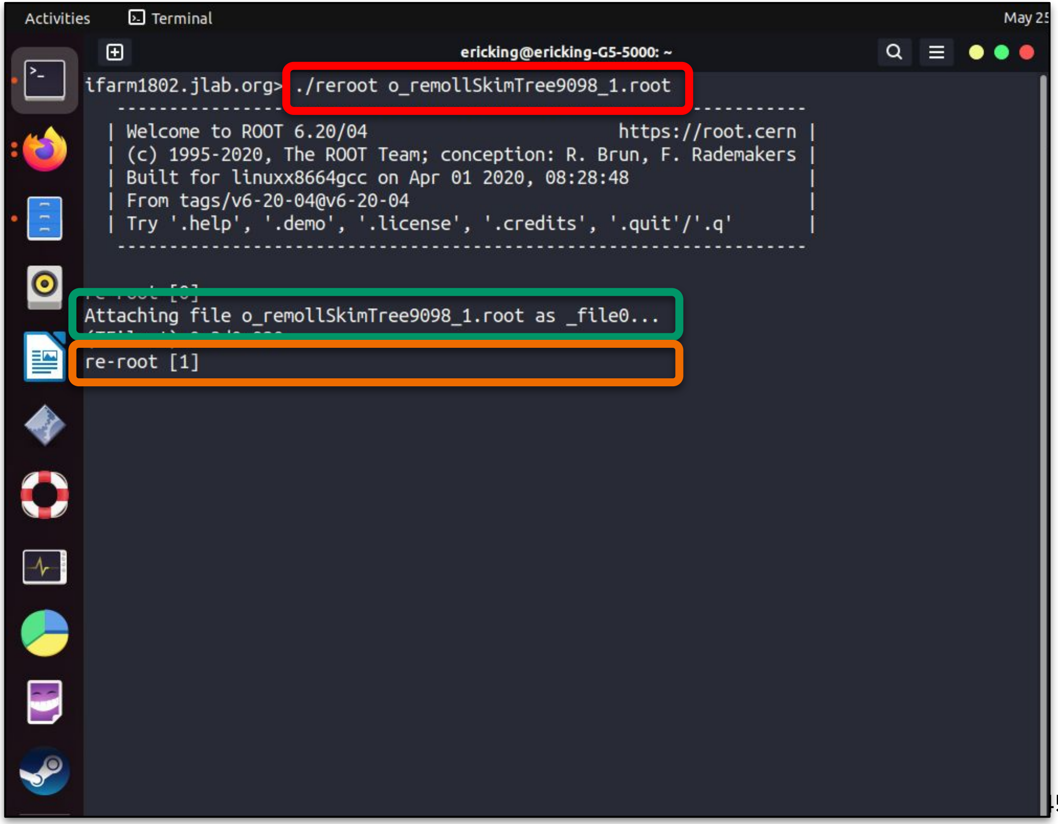
From the terminal command line you can open a ROOT session

We execute the command *root* and pass it a filename as an argument.

```
./reroot o_remollSkimTree.root
```

ROOT starts and we see that the file has successfully opened.

Now, the ROOT command line is waiting for instructions. :)



Using ROOT – Command Line Interface

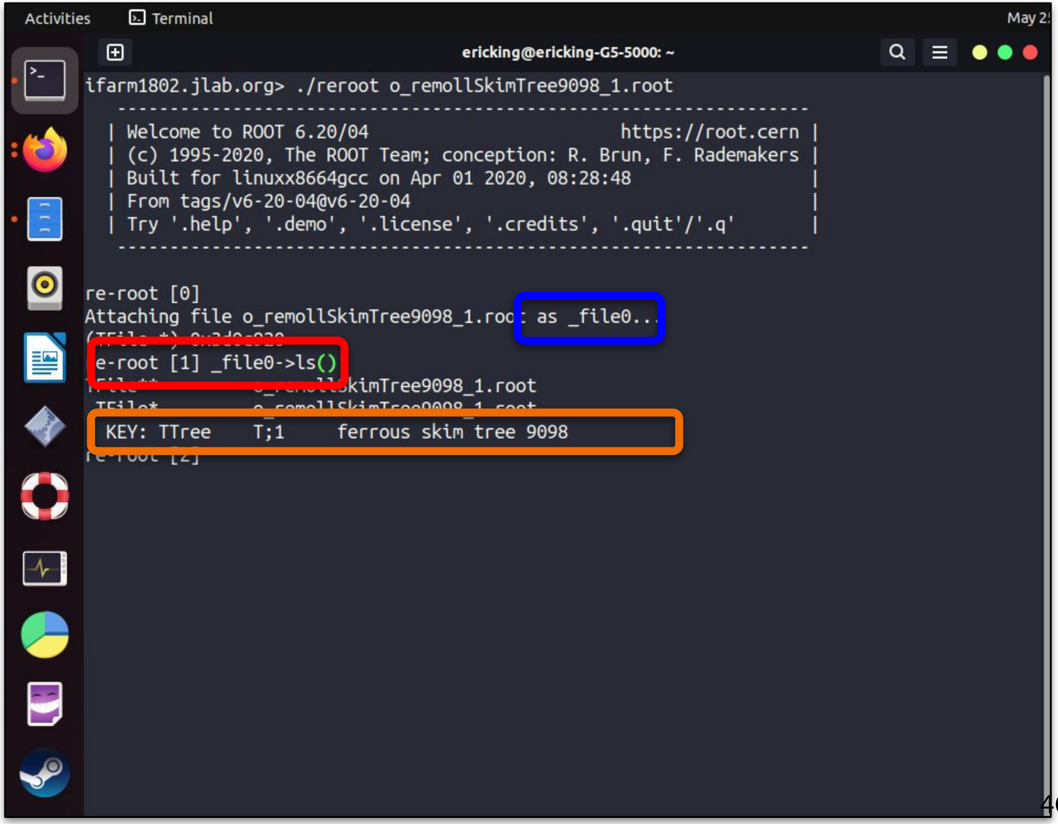
We can look at the contents of the ROOT file…

ROOT opened up the file and has auto-named the object `_file0`

Similar to the basic Linux command we can list the file contents using the ls() method of TFile.

```
_file0->ls()
```

☆ We have a data tree named “T” with the description “ferrous skim tree 9098”



Using ROOT – Command Line Interface

We can examine the structure of the data tree

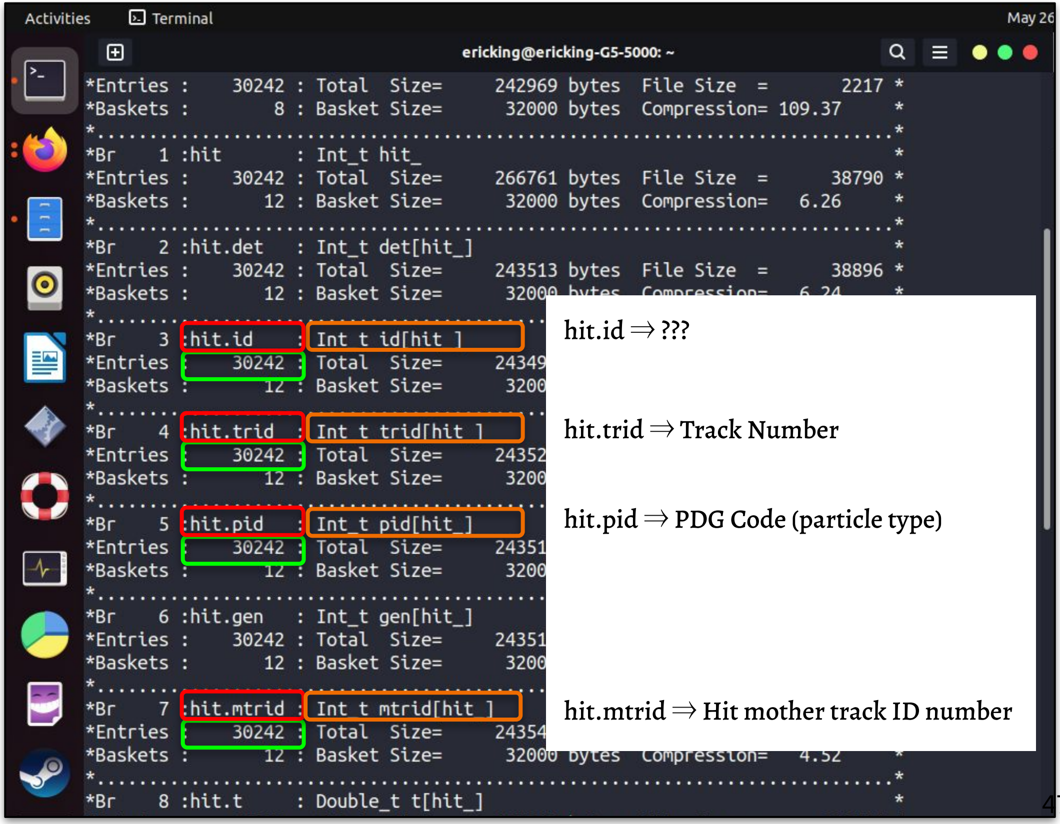
```
T->Print()
```

We can see branch names in red squares.

Total number of entries in green.

Details about the data structure in orange.

```
hit.trid    ⇒ Integer data type array
            ⇒ 30242 Entries
```



hit.id ⇒ ???

hit.trid ⇒ Track Number

hit.pid ⇒ PDG Code (particle type)

hit.mtrid ⇒ Hit mother track ID number

Using ROOT – Command Line Interface

We can perform a sampling (scan) of the data:

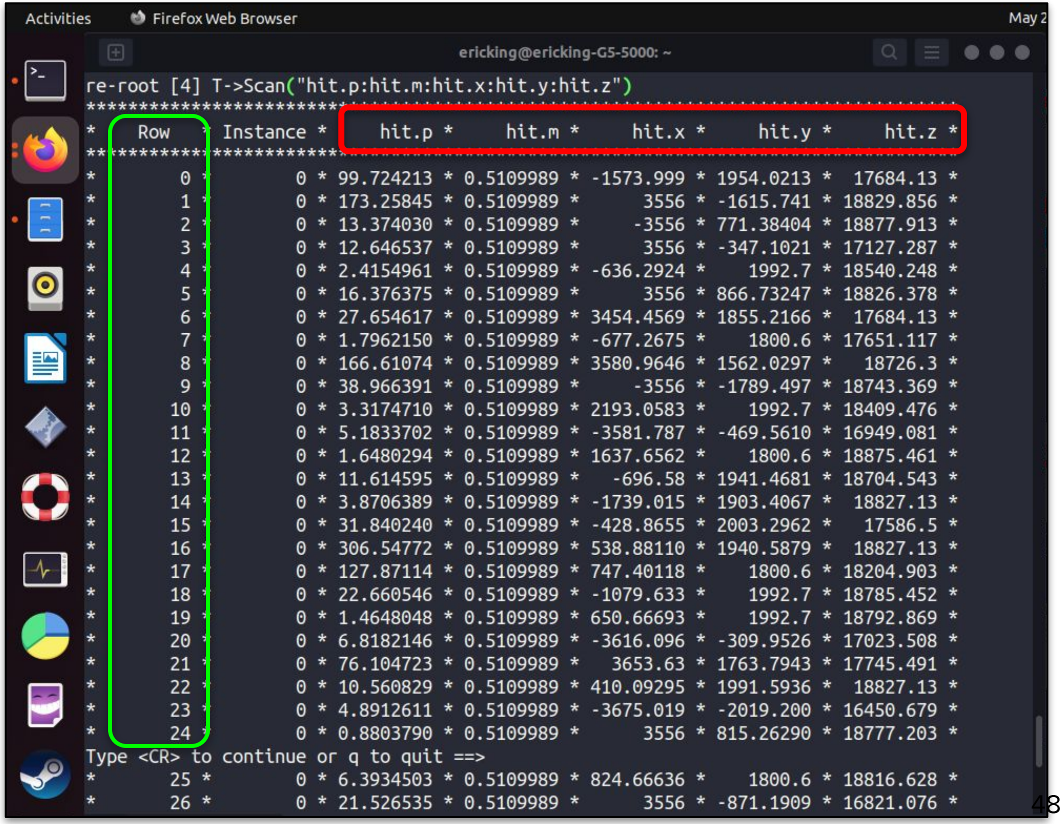
```
T->Scan(“hit.p:hit.m:hit.x …”)
```

We see our branches:

```
hit.p
hit.m
hit.x
hit.y
hit.z
```

Entries are the row numbers

Not seen here is the fact that you can have multiple values per entry.



re-root [4] T->Scan("hit.p:hit.m:hit.x:hit.y:hit.z")

Row	Instance	hit.p	hit.m	hit.x	hit.y	hit.z
0	0	99.724213	0.5109989	-1573.999	1954.0213	17684.13
1	0	173.25845	0.5109989	3556	-1615.741	18829.856
2	0	13.374030	0.5109989	-3556	771.38404	18877.913
3	0	12.646537	0.5109989	3556	-347.1021	17127.287
4	0	2.4154961	0.5109989	-636.2924	1992.7	18540.248
5	0	16.376375	0.5109989	3556	866.73247	18826.378
6	0	27.654617	0.5109989	3454.4569	1855.2166	17684.13
7	0	1.7962150	0.5109989	-677.2675	1800.6	17651.117
8	0	166.61074	0.5109989	3580.9646	1562.0297	18726.3
9	0	38.966391	0.5109989	-3556	-1789.497	18743.369
10	0	3.3174710	0.5109989	2193.0583	1992.7	18409.476
11	0	5.1833702	0.5109989	-3581.787	-469.5610	16949.081
12	0	1.6480294	0.5109989	1637.6562	1800.6	18875.461
13	0	11.614595	0.5109989	-696.58	1941.4681	18704.543
14	0	3.8706389	0.5109989	-1739.015	1903.4067	18827.13
15	0	31.840240	0.5109989	-428.8655	2003.2962	17586.5
16	0	306.54772	0.5109989	538.88110	1940.5879	18827.13
17	0	127.87114	0.5109989	747.40118	1800.6	18204.903
18	0	22.660546	0.5109989	-1079.633	1992.7	18785.452
19	0	1.4648048	0.5109989	650.66693	1992.7	18792.869
20	0	6.8182146	0.5109989	-3616.096	-309.9526	17023.508
21	0	76.104723	0.5109989	3653.63	1763.7943	17745.491
22	0	10.560829	0.5109989	410.09295	1991.5936	18827.13
23	0	4.8912611	0.5109989	-3675.019	-2019.200	16450.679
24	0	0.8803790	0.5109989	3556	815.26290	18777.203

Type <CR> to continue or q to quit ==>

Row	Instance	hit.p	hit.m	hit.x	hit.y	hit.z
25	0	6.3934503	0.5109989	824.66636	1800.6	18816.628
26	0	21.526535	0.5109989	3556	-871.1909	16821.076

Using ROOT – Command Line Interface

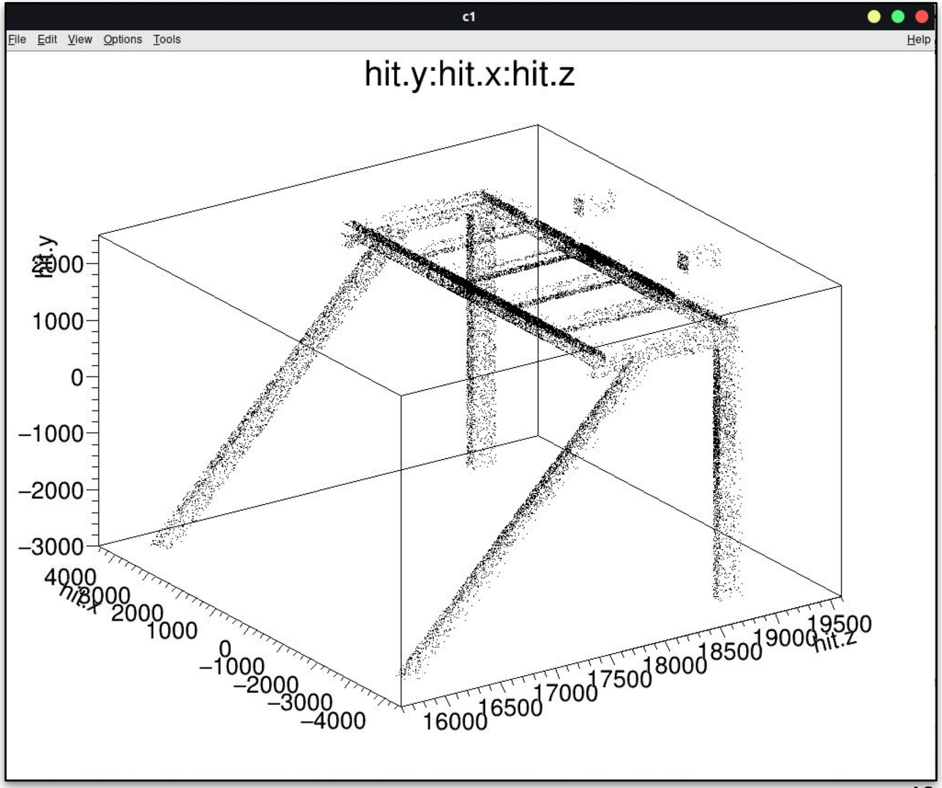
We can draw a sample scatter plot from the tree

```
T->Draw("hit.y:hit.x:hit.z")
```

Here we are drawing the locations of the hits on the sensitive detector in the simulation.

This scatterplot output is a TGraph object; it'll look nice and clean.

In a previous slide I created a TCanvas so I could specify a size and divide it. If you don't do that you'll get a default canvas object c1.



Using ROOT – Command Line Interface

Previous plot was TGraph which is a point-by-point plotting.

Below are two overlaid histograms; you can see how the bins 'fuzz' things out.

Specifying fine binning will fix this at a memory/drive space cost.

We can do a little more:

```
T->Draw("hit.y:hit.x:hit.z>>H","hit.e > 100")
```

We store the contents of the draw in an object called "H"

```
T->Draw("hit.y:hit.x:hit.z>>H2","hit.e < 100")
```

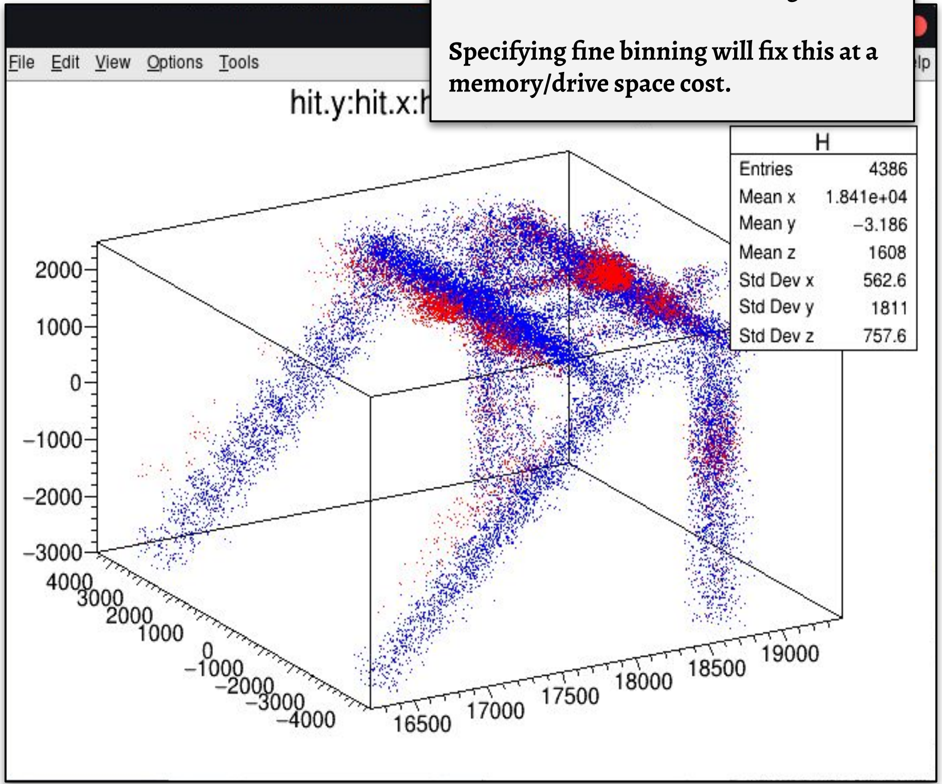
We store the contents of the draw in a histogram object called "H2"

```
H->SetMarkerColor(kRed)
H2->SetMarkerColor(kBlue)
```

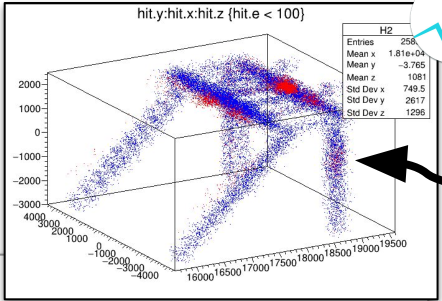
We set marker colors.

```
H2->Draw()
H->Draw("SAME")
```

And we Draw() – the second one we pass the argument "SAME"



Using ROOT – Command Line Interface



Quick Note:

Done something in the command line that you've found useful? You can turn it into a macro command by looking at ~/.root_hist

At the end of that file you'll find your latest commands.

- Copy these to a new text file
- Enclose in curly braces
- Add semicolons to the line ends.
- ROOT/CLING may be cranky about some other minor things.

```
{
  T->Draw("hit.y:hit.x:hit.z>>H","hit.e > 100");
  T->Draw("hit.y:hit.x:hit.z>>H2","hit.e < 100");
  H->SetMarkerColor(kRed);
  H2->SetMarkerColor(kBlue);
  H2->Draw();
  H->Draw("SAME");
}
```

Quick notes:

kRed and kBlue are variables in ROOT that are Int_t values. ROOT won't like those in the macro itself.

kRed = 2 ; kBlue = 4

```
./reroot  -l  root-file.root
(TFile *) 0x0000000
re-root [1] .x macro.txt
```

Using ROOT – Macros/Scripting

We can also write macros with more ‘complicated’ rules for data selection.

- This can be done for drawing data.
  - Perhaps data you pull from database.
- This can be done for performing more complicated calculations on raw data and creating a new ROOT file with calculated data.
- This can be done to data skim the information you want to move from one ROOT file into a separate smaller ROOT file.

```
#define molpol_plot_for_paper_cxx
#include "molpol_plot_for_paper.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void molpol_plot_for_paper::Loop(){

  TH2F * hCoin = new TH2F("hcoin",";X-coordinate [cm];Y-coordinate

  if (fChain == 0) return;

  Long64_t nentries = fChain->GetEntriesFast();

  Long64_t nbytes = 0, nb = 0;
  for (Long64_t jentry=0; jentry<nentries;jentry++){
    Long64_t ientry = LoadTree(jentry);
    if (ientry < 0) break;
    nb = fChain->GetEntry(jentry);   nbytes += nb;

    Bool_t trkCoin(false);
    Bool_t trk1hit(false);
    Bool_t trk2hit(false);
    Bool_t trkCoinA(false);
    Bool_t trk1hitA(false); //All coincidence on detector face
    Bool_t trk2hitA(false); //All coincidence on detector face
    for(Int_t i=0;i<hitN;i++){
      if(hitDet[i]==9 && hitTrid[i]==1) trk1hitA=true;
      if(hitDet[i]==9 && hitTrid[i]==2) trk2hitA=true;
    }
    if(trk1hitA&&trk2hitA) trkCoinA=true;
    //Fill Coincidence
    for(Int_t j=0; j<hitN; j++){
      if( hitDet[j]==9 && hitTrid[j]==1 ){
        if(trkCoinA==true)  hCoin->Fill(100*hitX[j],100*hitLy[j]);
      }
    }
  }

  gStyle->SetPalette(kSunset);
  gStyle->SetNdivisions(28,"z");

  Float_t topMargin = 0.025;
  Float_t rightMargin = 0.175;
  Float_t bottomMargin = 0.075;
  Float_t leftMargin = 0.125;

  gStyle->SetPadTopMargin(topMargin);
  gStyle->SetPadRightMargin(rightMargin);
  gStyle->SetPadBottomMargin(bottomMargin);
  gStyle->SetPadLeftMargin(leftMargin);
```

⇒ Specific code here is unimportant (not remoll).

What’s important to note is:

1. Data doesn’t contain all the immediate information we may need.
   a. Need to know if for any recorded Entry$ if each of the two generated electrons makes it to the detector.
2. Command line plotting is limited if you need compound data selection rules.

Using ROOT – Macros/Scripting

Creating macros by hand:

Be sure to include remolltypes.hh, this defines the hit, part, etc. data types.

- Open your ROOT file and your tree.

```
TFile * f = new TFile("yourFile.root","<Read/Write Option>");
TTree * t = new TTree("YourTree","Some Name");
```

- Declare variables to hold branch data and set your branch addresses:

```
Float_t someValue;
T->SetBranchAddress("branchName",&someValue);
... ... ...
```

- Proceed with your data selection, histogram filling, and canvas building.

Resources: Extensive Documentation By CERN

ROOT Manual: https://root.cern/manual/

ROOT Reference Documentation: https://root.cern/doc/master/
⇒ Although, it's just as easy to Google "cern root <insert-class> class reference"

ROOT Tutorials: https://root.cern/doc/master/group__Tutorials.html
⇒ Abundance of examples on histograms, graphs, data fitting, SQL-interfacing, and (for the Python-inclined) examples using PyROOT.
⇒ And more… [plenty of stuff from beginners to advanced]

ROOT Forum: https://root-forum.cern.ch
⇒ Someone has very likely asked your question before…

Additional Functionality

- Plenty of available extended functionality with ROOT
  - Machine Learning libraries [TMVA]
    - https://root.cern/manual/tmva/
  - PyRoot (Use ROOT with Python)
    - https://root.cern/manual/python/
  - JSroot (A Javascript Framework for looking at ROOT files)
    - https://root.cern.ch/js/

Simple script to read CSV into ROOT file

```
#include<TROOT.h>
#include<TFile.h>
#include<TTree.h>
#include<TString.h>
#include<iostream>

Int_t read_csv(TString infile, TString desc, TString output){
  TFile * f = new TFile(Form("%s.root",infile.Data()),"RECREATE");
  TTree * T = new TTree("T","dataTree");
  Long64_t nlines = T->ReadFile(infile,"",',');
  cout << "Number of lines read: " << nlines << endl;
  f->Write();
  f->Close();
  return 0;
}
```

⇒ You can run the following ROOT script on the command line with:

root -l read_csv.C+'("*datafile.csv*")'

- Header information in CSV must contain data type information followed by data:

```
Event/I,Value1/F,Value2/F,Value3/I, …
0,9.27577,0.12836,11, …
1,4.91736,-0.98736,8, …
```

- ROOT can be picky reading in csv data but is useful.
    - Data output from hardware
    - EPICS archive output
    - etc...
- *If this was written well it would just replace the substring .csv with .root :)*