Simulation Project: Part 2

Dr. Rakitha Beminiwattha
Louisiana Tech University

Goals

- Use simulation output root files of part 1 of the project
- Use `RootScript.C` as a template script and create a root script relevant for above study
- Use the output of this script to study the effects of slit scattering from collimator
    - We will look at Photons (pid=22) hitting the sensitive detector (det id 47) we created previously at GDML overview session

Prerequisites

- Files required for the tutorial are available at Hands-On-Remoll in

  https://drive.google.com/drive/folders/1Bcoe3hPBwS1MCXiwCdD0ghjgKnkjRsfB?usp=sharing

- Download the file `RootScript.C` into ~/softwares/remoll/analysis
- the root files `remollout_Moller_gen_2k.root` and `remollout_Moller_gen_Krypto_2k.root`
  - Both of these files are also available in Hands-On/Rootfiles directory in shared Google Drive
  - Make sure these rootfiles are in ~/softwares/remoll/

Root Scripting: Histogram Declarations

RootScript.C is our template script, we will create histograms of hit radius, xy 2D distribution and source vertex of these hits on Det-47

1. Declare 1D histograms for radius and source vertex

```
TH1D *r

TH1D *sourceZ

TH1D *rRate //for rate weighted radial distribution
```

2. Define 2D histograms for XY distribution

```
TH2D *hXY

TH2D *hXYrate //for rate weighted XY distribution
```

Root Scripting: Histogram definitions

- Let's define their parameters and create them inside initHisto() routine

```
  r = new TH1D("r","Det47 radial distribution;r[mm]",200,0,600);

  rRate = new TH1D("rRate","Det 47 rate weighted
distribution;r[mm]",200,0,600);

  hXY = new TH2D("hXY","2D hit distribution;x [mm];y
[mm]",200,-600,600,200,-600,600);

  hXYrate = new TH2D("hXYrate","rate weighted 2D hit
ditribution;x [mm];y [mm]",200,-600,600,200,-600,600);

  sourceZ = new TH1D("sourceZ","initial vertex for hit ;z
position [mm]",10000,-5300,8000);
```

Root Scripting: Proper Cuts

- Let’s set the proper cuts to match our analysis, cuts are applied in the `processOne(...)` routine
  - Select only photons

```
if(hit->at(j).pid!=22) continue;
```

  - Select hits only on detector id 47

```
if(hit->at(j).det != 47) continue;
```

- Next we fill histograms

Root Scripting: Filling Histograms

- Let's fill these histograms with data from the Tree in at the `processOne(...)` routine

```
r->Fill(hit->at(j).r);

sourceZ->Fill(hit->at(j).vz);

hXY->Fill(hit->at(j).x,hit->at(j).y);

rRate->Fill(hit->at(j).r,rate);

hXYrate->Fill(hit->at(j).x,hit->at(j).y,rate);
```

Root Scripting: Post Processing

- Scale rate weighted histograms if we have used chain of root files (more than one root filed linked) in the `void scale()` routine

```
rRate->Scale(1./nFiles);

hXYrate->Scale(1./nFiles);
```

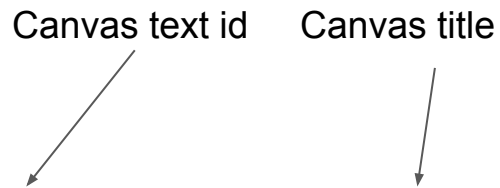
Make Histograms in Canvases: Create a Canvas

- In routine `void plot()`
- Let's create a canvas

```
Double_t w = 600;//width px

Double_t h = 600;//height px

TCanvas *p1 = new TCanvas("TCan_sourceZ","Source Z
Canvas",w,h);
```



Format Histogram and Draw on the Canvas

- Use the function: `gStyle->SetOptStat("nemr");` to format histogram stat box information: n-name, e-events, m-mean, r-rms
- Draw the vertex histogram on the p1 Canvas : `sourceZ->DrawCopy();`
- You can save the canvas as an image to formats including pdf or png: `p1->SaveAs("TCan_sourceZ.png");`
- This command will save the canvas p1 with the histogram `sourceZ` as an image file.

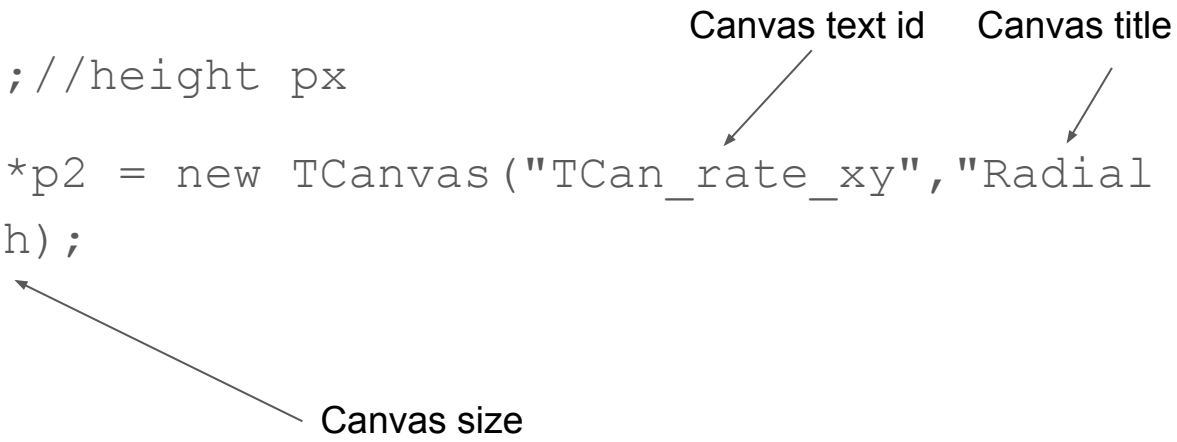
Make Histograms in Canvases: Create a Canvas

- Let's create the second canvas

```
w = 1000;//width px

h = 1000;//height px

TCanvas *p2 = new TCanvas("TCan_rate_xy","Radial and XY
hits",w,h);
```



Make Histograms in Canvases: Divide the Canvas

- Let's create the canvas p2 with multiple pads : `p2->Divide(2,2);//nx,ny`
  - Above command creates 2 by 2 = 4 pads in the canvas
- Let's add four histograms into this canvas

```
p2->cd(1);

r->DrawCopy();

p2->cd(2);

rRate->DrawCopy();

p2->cd(3);

hXY->DrawCopy();

p2->cd(4);

hXYrate->DrawCopy();
```

Save Output into a Root File for Later access

- Output written in this step can be accessed later in a root file “`RootScript.root`”
- In the routine `void writeOutput()`

```
r->Write();

sourceZ->Write();

hXY->Write();

rRate->Write();

hXYrate->Write();
```

Save Output into a Root File for Later access

- Output written in this step can be accessed later in a root file “`RootScript.root`”
- This file name is set in the routine `void initHisto()`

```
string foutNm = Form("RootScript.root");
```

- You can access the saved histograms using the command

```
root RootScript.root or ./build/reroot RootScript.root
```

Analysis Steps

1. We will first run the script `RootScript.C` with root file `remollout_Moller_gen_2k.root`
2. Give an unique name at `string foutNm = Form("RootScript_real.root");`
3. Load the script RootScript.C

   ```
   .L analysis/RootScript.C
   ```

4. Execute the script

   ```
   RootScript(“remollout_Moller_gen_2k.root”)
   ```

Analysis Steps

1. We will then run the script RootScript.C with root file `remollout_Moller_gen_Krypto_2k.root`
2. Give an unique name at `string foutNm = Form("RootScript_krypto.root");`
3. Load the script RootScript.C

   ```
   .L analysis/RootScript.C
   ```

4. Execute the script

   ```
   RootScript(“remollout_Moller_gen_Krypto_2k.root”)
   ```