Abstract

This past week, I spent most of my time trudging through C++ and ROOT tutorials and reference pages online, in order to get the relevant data out of my optical MC runs. I can now extract S1 and S2 light histograms and overlay them from any two selected runs. This will greatly speed up my study into which parameters affect the S2 distribution more dramatically than they do the S1 distribution. Since the MC runs take roughly a day each, I’ll spend the majority of my time next week reading through Ben Loer’s and Luca Grandi’s theses in an attempt to figure out which are the relevant parameters, while the MC chugs along in the background.

1 Literature

1.1 Summary

I’ve printed Ben Loer’s and Luca Grandi’s Ph.D. theses, in order to get a better idea of DarkSide in general, but specifically the parameters referenced in the optical MC. I’ve also printed a ROOT User’s Guide, which I’ll begin to look over soon. This makes my current reading list: Ben Loer’s Thesis, Luca Grandi’s Thesis, ROOT User’s Guide, art User’s Guide, and the LaTeX guide (of which I’m on Chapter V). It looks like I’ll be hitting the books for the next week or two.

2 Coding

2.1 C++

You may have noticed that, in the previous section, my C++ tutorials weren’t mentioned as part of my reading list—that’s because as of yesterday I finished
the 17th and final chapter of those lessons. They definitely came in handy when writing my macros in Section 3, especially the last few chapters on the Standard Template Library (STL) and std::strings.

2.2 ROOT

Using Christy’s macro as a jumping-off point, I figured out how to extract S1 as well as S2 data from the optical MC runs I created, as can be found below in Section 3. I’m now able to create eight relevant histograms for each MC run: [S1 OR S2][Max Channel 1 OR 0][Channel 1 or 0] and then overlay these histograms from two different MC runs, with relevant parameters adjusted in the interim.

The actual data *.root file has a slightly different Tree hierarchy, so I need to figure out how to work with that, as well.

As of this writing, I have ten ROOT files: five with the parameters unchanged (Runs 001-005), to get a baseline; and five with elike_quenching (“Alfredo’s quenching factor”) adjusted from its initial 0.8 to 0.7, 0.6, 0.5, 0.4, and 0.3 (Runs 006-010). I’m currently running the MC five times over to get runs where this parameter has been adjusted to 0.9, 1.0, 1.1, 1.2, and 1.3 (Runs 011-015).

Since elike_quenching is a variable which seems to affect only the electrons, I thought it would have a greater effect on the S2 rather than the S1 light, but obviously I need to do some more research to figure out which other parameters, chunks of code, etc. would be relevant. The big accomplishment this past week was being able to write these macros and get used to running them.

2.3 Optical MC

Initially, I emailed Christy in an attempt to figure out how to turn old DS10 data into *.root files. After some technical issues, we decided it would be best to just not worry about genrooting the old data into *.root files—there are some old files floating around blackhole, so I’ll just use those. Christy also remarked that most people probably aren’t even using the optical MC environment for analysis anymore, suggesting I get used to art, for DS50. As it turns out, this is partly true, though there are still a few bugs being worked out from the old MC (top-bottom asymmetry, for instance). Naturally, though, I’ll also start getting used to art and DarkArt, which are still in a state of flux.
3 Macros

3.1 extractS1S2.C

The following C language macro creates S1 and S2 light distributions from a single optical MC run.

Note, there are two instances of ellipses (...) which are used to simply shorten path names so as to not overflow the horizontal box in this $\LaTeX$ document.

S2 & S1 Light Distribution for DS-10 MC:

Reads in optical MC information and creates a root file with histograms of S2 and S1 light distributions on the top PMTs, overlayed on histograms of S2 and S1 light distributions on the top PMTs of a second *.root file, in order to see if modified parameters affect light distributions.

#include <iostream>
#include "TPrincipal.h"
#include "iomanip.h"
#include "Riostream.h"
#include "TROOT.h"
#include "TApplication.h"
#include "TCanvas.h"
#include "TH1.h"
#include "TSystem.h"
#include "TBrowser.h"
#include "TFile.h"
#include "TRandom.h"
#include "TMultiDimFit.h"
#include "TVectorD.h"
#include "TMath.h"

/* ---------------------------------- */
/* EDIT "m" for the number of events! */
/* ---------------------------------- */

// n = 7, for only top PMTs?
void extractS1S2(Int_t n = 7, Int_t m = 10000){
    //loads libraries needed for MC analysis
    gROOT->ProcessLine(".L ..libMCOpticalEvent.so");
    TChain* chains2 = new TChain("mc");
    TChain* chains1 = new TChain("mc");
TFile* f = new TFile ("...run010out.root", "RECREATE");

//create histograms for Max Chan 0
TH1F* S2ch0max0 = new TH1F("S2ch0max0", 
    "Histogram: S2 Light Fraction on Ch0 (S2 Max Chan 0)", 
    200, 0, 1);
TH1F* S2ch1max0 = new TH1F("S2ch1max0", 
    "Histogram: S2 Light Fraction on Ch1 (S2 Max Chan 0)", 
    200, 0, 1);
TH1F* S1ch0max0 = new TH1F("S1ch0max0", 
    "Histogram: S1 Light Fraction on Ch0 (S1 Max Chan 0)", 
    200, 0, 1);
TH1F* S1ch1max0 = new TH1F("S1ch1max0", 
    "Histogram: S1 Light Fraction on Ch1 (S1 Max Chan 0)", 
    200, 0, 1);

//create histograms for Max Chan 1
TH1F* S2ch0max1 = new TH1F("S2ch0max1", 
    "Histogram: S2 Light Fraction on Ch0 (S2 Max Chan 1)", 
    200, 0, 1);
TH1F* S2ch1max1 = new TH1F("S2ch1max1", 
    "Histogram: S2 Light Fraction on Ch1 (S2 Max Chan 1)", 
    200, 0, 1);
TH1F* S1ch0max1 = new TH1F("S1ch0max1", 
    "Histogram: S1 Light Fraction on Ch0 (S1 Max Chan 1)", 
    200, 0, 1);
TH1F* S1ch1max1 = new TH1F("S1ch1max1", 
    "Histogram: S1 Light Fraction on Ch1 (S1 Max Chan 1)", 
    200, 0, 1);

TH1F* PCA_PMThist_S2 = new TH1F("PCA_PMThist_S2", 
    /* ---------------------------- */
    /* EDIT "f" to the output file! */
    /* ---------------------------- */
"PCA_PMThist_S2", n,1,n+1);
TH1F* PCA_PMThist_S1 = new TH1F("PCA_PMThist_S1",
"PCA_PMThist_S1", n,1,n+1);

// input m data points to the fit
// for the first argument and x as second

for(int i = 0; i < m; i++){
  for (int j = 0; j < n; j++){
    datas2[j] = 0.0;
    datas1[j] = 0.0;
  }

  totals2 = totals1 = 0.0;
  maxchans2 = maxchans1 = 0;

  stringstream args2;
  args2 << "mc.MCOpticalEvent.NEvent == " << i+1;
  TCut eventnums2 = args2.str().c_str();

  stringstream args1;
  args1 << "mc.MCOpticalEvent.NEvent == " << i+1;
  TCut eventnums1 = args1.str().c_str();

  chains2->Draw("mc.MCOpticalEvent.PMHits2.detector_index
  >>PCA_PMThist_S2", eventnums2);
  chains1->Draw("mc.MCOpticalEvent.PMHits1.detector_index
  >>PCA_PMThist_S1", eventnums1);

  for (int j = 0; j < n; j++){
    datas2[j] = PCA_PMThist_S2->GetBinContent(j+1);
    totals2 = totals2 + datas2[j];
    datas1[j] = PCA_PMThist_S1->GetBinContent(j+1);
    totals1 = totals1 + datas1[j];
  }

  for (int j = 0; j < n; j++){
    datas2[j] = datas2[j]/totals2;
    datas1[j] = datas1[j]/totals1;
    cout << j << "	" << datas2[j] << endl;
  }

  for (int j = 1; j < n; j++){
    if (datas2[j] > datas2[maxchans2]) maxchans2 = j;
    if (datas1[j] > datas1[maxchans1]) maxchans1 = j;
  }
if (maxchans2 == 0){
    S2ch0max0->Fill(datas2[0]);
    S2ch1max0->Fill(datas2[1]);
}if(maxchans1 == 0){
    S1ch0max0->Fill(datas1[0]);
    S1ch1max0->Fill(datas1[1]);
}if(maxchans2 == 1){
    S2ch0max1->Fill(datas2[0]);
    S2ch1max1->Fill(datas2[1]);
}if(maxchans1 == 1){
    S1ch0max1->Fill(datas1[0]);
    S1ch1max1->Fill(datas1[1]);
}
}

// WRITE HISTOGRAMS TO ROOT FILE
S2ch0max0->Write();
S2ch1max0->Write();
S1ch0max0->Write();
S1ch1max0->Write();
S2ch0max1->Write();
S2ch1max1->Write();
S1ch0max1->Write();
S1ch1max1->Write();
f->Close();
}

3.2 compare.C

The following C language macro overlays matching S1/S2 light distribution histograms from two runs of the user’s choosing. Histograms and output *.gifs are dynamically named, based on the runs chosen for comparison by the user.

# include "TFile.h"
# include "TChain.h"
# include "TH1F.h"
# include "TH2F.h"
#include "TProfile.h"
#include "TCanvas.h"
#include "TROOT.h"
#include "TStyle.h"
#include "TF1.h"
#include "TMath.h"
#include <iomanip> // for cout.setw() function
#include <iostream>
#include <fstream>
#include "math.h"
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>
#include <map> // for maps

template<typename T>
inline std::string ToString(T tX){
    std::ostringstream oStream; oStream << tX;
    return oStream.str();
}

template<typename T>
inline bool FromString(const std::string& sString, T &tX){
    std::istringstream iStream(sString);
    return (iStream >> tX) ? true : false;
}

void compare(std::string titleone = "001",
             std::string titletwo = "002"){
    std::string path = "/data/test_processing/watson/FINISHED/run";
    std::ostringstream osone; osone.str(""");
    std::ostringstream ostwo; ostwo.str(""");
    std::ostringstream outS1; outS1.str(""");
    std::ostringstream outS2; outS2.str(""");
    osone << path << titleone << "out.root";
    ostwo << path << titletwo << "out.root";
    outS1 << "Run " << titleone << " (black) v "
         << "Run " << titletwo << " (red) [S1]";
    outS2 << "Run " << titleone << " (black) v "
         << "Run " << titletwo << " (red) [S2]";

    TFile *fileone = new TFile(osone.str().c_str());
TFile *filetwo = new TFile(ostwo.str().c_str());

/* When split into a 2x2 canvas, the below dimensions give
 four pads of 496x372, with an exact 4:3 ratio (standard TV) */

TCanvas* c0 = new TCanvas("c0", outS1.str().c_str(), 968, 726); // S1 Light
TCanvas* c1 = new TCanvas("c1", outS2.str().c_str(), 968, 726); // S2 Light

c0->Divide(2, 2); c1->Divide(2, 2);
outS1.str(""); outS2.str("");

outS1 << titleone << "v" << titletwo << "S1.gif";
outS2 << titleone << "v" << titletwo << "S2.gif";

//
// DYNAMIC CREATION OF HISTOGRAMS
//

map <int, TH1F*> Hists;
std::ostringstream indx; // Index of Histogram in map
std::ostringstream labl; // Label of Histogram
int ndex = 0; // Integer index of Histogram in map
int frst = 0;
int plt1 = 0; // Counters to keep track of canvas plots
int plt2 = 0;

int file = 1; // 1 or 2
int lite = 1; // 1 or 2 (S1 or S2)
int ch = 0; // 0 or 1 (Channel)
int max = 0; // 0 or 1 (Max Channel)

for( ch; ch < 2; ch++) { max = 0;
for( max; max < 2; max++) { lite = 1;
for(lite; lite < 3; lite++) { file = 1;
if(lite < 2) { plt1++; } else { plt2++; }
for(file; file < 3; file++){

labl.str("");
labl << ToString("S") << ToString(lite)
    << ToString("ch") << ToString(ch)
    << ToString("max") << ToString(max);

indx.str("");
indx << ToString(file) << ToString(lite)
<< ToString(ch) << ToString(max);

FromString(index.str().c_str(), ndex);
std::string strlabl = labl.str();

if(file < 2){ Hists[ndex] =
    (TH1F*)fileone->Get(strlabl.data());
    frst = ndex; }
else{ Hists[ndex] =
    (TH1F*)filetwo->Get(strlabl.data()); }
}

if(lite < 2){ c0->cd(plt1); } else { c1->cd(plt2); }

Hists[ndex]->SetLineColor(kRed);
if(Hists[frst]->GetMaximum() >= Hists[ndex]->GetMaximum()){
    Hists[frst]->Draw(); Hists[ndex]->Draw("same");
} else{ Hists[ndex]->Draw(); Hists[frst]->Draw("same"); }
}}}

// // MANUAL CREATION OF HISTOGRAMS //
/*
// Naming format H[FileNumber{1/2}][S1/S2][ch0/ch1][max0\max1]
// ex: P1201 = Histogram: File 1, S2 light, Channel 0, Max Channel 1
    TH1F* H1100 = (TH1F*)fileone->Get("S1ch0max0"); // Max Ch 0
    TH1F* H2100 = (TH1F*)filetwo->Get("S1ch0max0"); // Ch0 Hists
    TH1F* H1200 = (TH1F*)fileone->Get("S2ch0max0"); //
    TH1F* H2200 = (TH1F*)filetwo->Get("S2ch0max0"); //
    TH1F* H1101 = (TH1F*)fileone->Get("S1ch0max1"); // Max Ch 1
    TH1F* H2101 = (TH1F*)filetwo->Get("S1ch0max1"); // Ch0 Hists
    TH1F* H1201 = (TH1F*)fileone->Get("S2ch0max1"); //
    TH1F* H2201 = (TH1F*)filetwo->Get("S2ch0max1"); //
    TH1F* H1110 = (TH1F*)fileone->Get("S1ch1max0"); // Max Ch 0
    TH1F* H2110 = (TH1F*)filetwo->Get("S1ch1max0"); // Ch1 Hists
    TH1F* H1210 = (TH1F*)fileone->Get("S2ch1max0"); //
    TH1F* H2210 = (TH1F*)filetwo->Get("S2ch1max0"); //
    TH1F* H1111 = (TH1F*)fileone->Get("S1ch1max1"); // Max Ch 1
    TH1F* H2111 = (TH1F*)filetwo->Get("S1ch1max1"); // Ch1 Hists
*/
TH1F* H1211 = (TH1F*)fileone->Get("S2ch1max1"); //
TH1F* H2211 = (TH1F*)filetwo->Get("S2ch1max1"); //

H2100->SetLineColor(kRed); c0->cd(1);
if(H1100->GetMaximum() >= H2100->GetMaximum()){
  H1100->Draw(); H2100->Draw("same");
} else { H2100->Draw(); H1100->Draw("same"); }

H2200->SetLineColor(kRed); c1->cd(1);
if(H1200->GetMaximum() >= H2200->GetMaximum()){
  H1200->Draw(); H2200->Draw("same");
} else { H2200->Draw(); H1200->Draw("same"); }

H2101->SetLineColor(kRed); c0->cd(2);
if(H1101->GetMaximum() >= H2101->GetMaximum()){
  H1101->Draw(); H2101->Draw("same");
} else { H2101->Draw(); H1101->Draw("same"); }

H2201->SetLineColor(kRed); c1->cd(2);
if(H1201->GetMaximum() >= H2201->GetMaximum()){
  H1201->Draw(); H2201->Draw("same");
} else { H2201->Draw(); H1201->Draw("same"); }

H2110->SetLineColor(kRed); c0->cd(3);
if(H1110->GetMaximum() >= H2110->GetMaximum()){
  H1110->Draw(); H2110->Draw("same");
} else { H2110->Draw(); H1110->Draw("same"); }

H2210->SetLineColor(kRed); c1->cd(3);
if(H1210->GetMaximum() >= H2210->GetMaximum()){
  H1210->Draw(); H2210->Draw("same");
} else { H2210->Draw(); H1210->Draw("same"); }

H2111->SetLineColor(kRed); c0->cd(4);
if(H1111->GetMaximum() >= H2111->GetMaximum()){
  H1111->Draw(); H2111->Draw("same");
} else { H2111->Draw(); H1111->Draw("same"); }

H2211->SetLineColor(kRed); c1->cd(4);
if(H1211->GetMaximum() >= H2211->GetMaximum()){
  H1211->Draw(); H2211->Draw("same");
} else { H2211->Draw(); H1211->Draw("same"); }
*/

c0->Update();
c0->SaveAs(outS1.str().c_str());
c1->Update();
c1->SaveAs(outS2.str().c_str());
return;
}

4 Histograms

4.1 Baseline

The following are all "baseline" runs, where no parameters have been changed. We should therefore see very few differences between the overlaid histograms.

4.1.1 S1 Light

Figure 1: Run 001 v Run 002 S1 light
Figure 2: Run 001 v Run 003 S1 light

Figure 3: Run 001 v Run 004 S1 light
4.1.2 S2 Light

Figure 4: Run 001 v Run 005 S1 light

Figure 5: Run 001 v Run 002 S2 light
Figure 6: Run 001 v Run 003 S2 light

Figure 7: Run 001 v Run 004 S2 light
4.2 “Alfredo’s Quenching Factor”

The following histograms show how elike quenching can affect the histograms—as it turns out, not much (at least when adjusted 10% at a time). Note, random runs are chosen for comparison from 001-005.
4.2.1 S1 Light

Figure 9: Run 004 v Run 006 S1 light

Figure 10: Run 003 v Run 007 S1 light
Figure 11: Run 005 v Run 008 S1 light

Figure 12: Run 001 v Run 009 S1 light
4.2.2 S2 Light

Figure 13: Run 002 v Run 010 S1 light

Figure 14: Run 004 v Run 006 S2 light
Figure 15: Run 003 v Run 007 S2 light

Figure 16: Run 005 v Run 008 S2 light
Figure 17: Run 001 v Run 009 S2 light

Figure 18: Run 002 v Run 010 S2 light