Light simulation and reconstruction tutorial

Patrick Green

UK LArSoft Workshop 2022

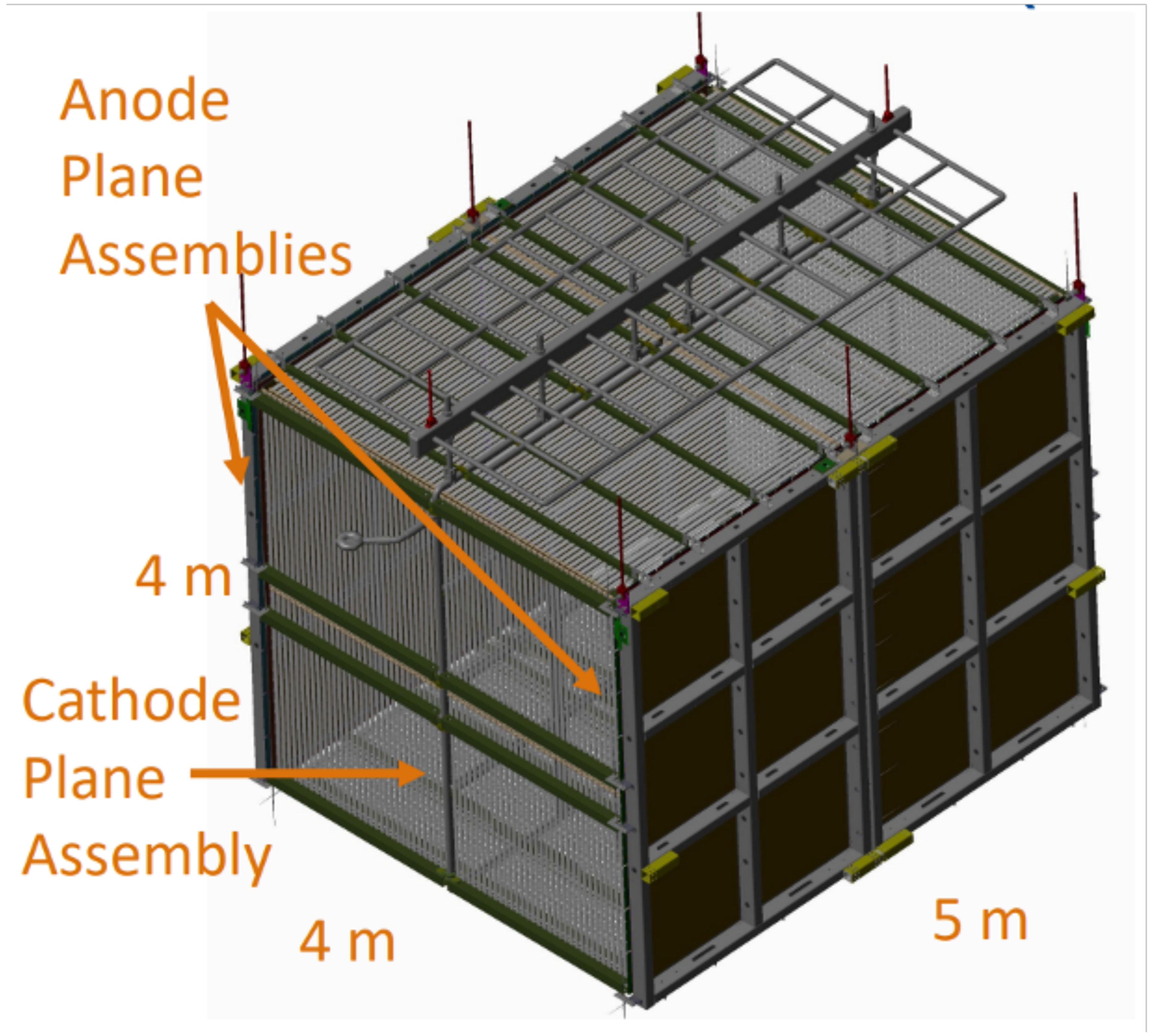
Introduction

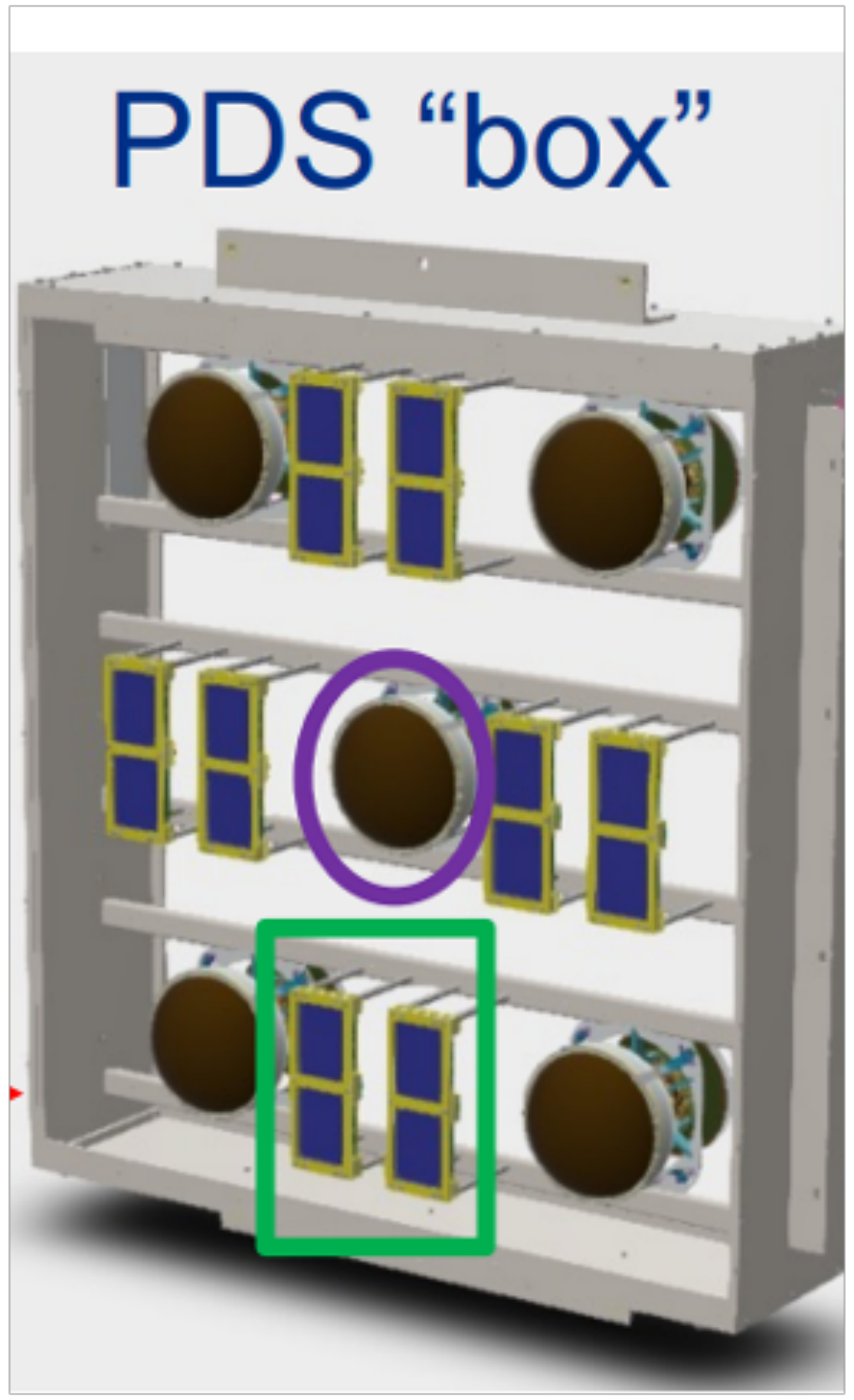
- This tutorial will cover simulation and reconstruction of the scintillation light
  - how to run + look at how the light behaves in different scenarios
- Divided into two parts:
  - part 1: running the light simulation and looking at truth-level results for different events
  - part 2: running the detector response simulation and reconstruction and looking at the resulting reconstructed objects
- For each part there will be a brief introduction then several tasks to work through at your own pace

Part 1: running the light simulation

Reminder: working with the SBND geometry

- Two TPCs, separated by an opaque CPA (centre)
- Light detectors at each APA:
    - PMTs and X-Arapucas
- Cathode has wavelength-shifting reflective foils on both sides:
    - we will see two components to the light: “direct” / “vuv” and “reflected” / “visible”
- Note: photo-detectors only see light from interactions occurring in the same TPC





Running the light simulation in SBND

- We will be using this fhicl file: *optical_tutorial_sim_muons.fcl*
- You can find this fhicl in *Workshop/Photon/fcl/* in your local sbndcode install
- This fhicl will generate 2 GeV muons at a certain position in the detector
- It will then run the light simulation (LArG4 stage), followed by an analyzer module that will provide 3 TTrees with truth-level information about the light

Running the light simulation in SBND

Refactored LArG4:

```
producers:
{
  rns: { module_type: "RandomNumberSaver" }

  # Generation
  generator: @local::sbnd_singlep

  # A dummy module that forces the G4 physics list to be loaded
  loader: { module_type: "PhysListLoader" }

  # The geant4 step
  largeant: @local::sbnd_larg4

  # Creation of ionization electrons and scintillation photons,
  # inside the active volume
  ionandscint: @local::sbnd_ionandscint

  # Creation of ionization electrons and scintillation photons,
  # outside the active volume
  ionandscintout: @local::sbnd_ionandscint_out

  # Light propagation inside the active volume, semi-analytic model
  pdfastsim: @local::sbnd_pdfastsim_par

  # Light propagation outside the active volume, photon library
  pdfastsimout: @local::sbnd_pdfastsim_pvs

  # Electron propogation
  simdrift: @local::sbnd_simdrift

  # Truth-level reconstruction
  mcreco: @local::sbnd_mcreco
}
```

Prod Single generator

Hybrid model light simulation

Configuration of semi-analytic and/or library models (detector-specific)

```
sbnd_pdfastsim_par {

    # Direct (VUV)
    VUVTiming:    @local::sbnd_vuv_timing_parameterization
    VUVHits:      @local::sbnd_vuv_RS100cm_hits_parameterization

    # Reflected (Visible)
    VISTiming:    @local::sbnd_vis_timing_parameterization
    VISHits:      @local::sbnd_vis_RS100cm_hits_parameterization

    DoReflectedLight: true
    IncludePropTime: true
    ...
}
```

reflected light

propagation time

Also include an analyzer that will allow us to access the truth level information: (SBND-specific but similar tools available in other detectors)

```
analyzers:
{
  # Analyzer to count number of photons arriving on photo-detectors
  opanalyzer: @local::OpDetAnalyzer
}
```

Task 1 preparation

- We will be using the same sbndcode installation from yesterday
- We need to re-setup the environment:
    - connect to the vnc viewer as before, then open a new terminal (http://py-dom.lancs.ac.uk:8080/guacamole/#/)
    - Next cd to your local sbndcode installation and set it up:

      cd $HOME/<your-working-directory>
      source /cvmfs/sbnd.opensciencegrid.org/products/sbnd/setup_sbnd.sh
      source localProducts_larsoft_v09_60_00_e20_prof/setup
      mrbslp

- Make a new empty directory called photon_tutorial (or whatever you like) in your home directory to work in and copy *optical_tutorial_sim_muons.fcl* to this directory (it can be found in $HOME/<your-working-directory>/srcs/sbndcode/sbndcode/Workshop/Photon/fcl/)

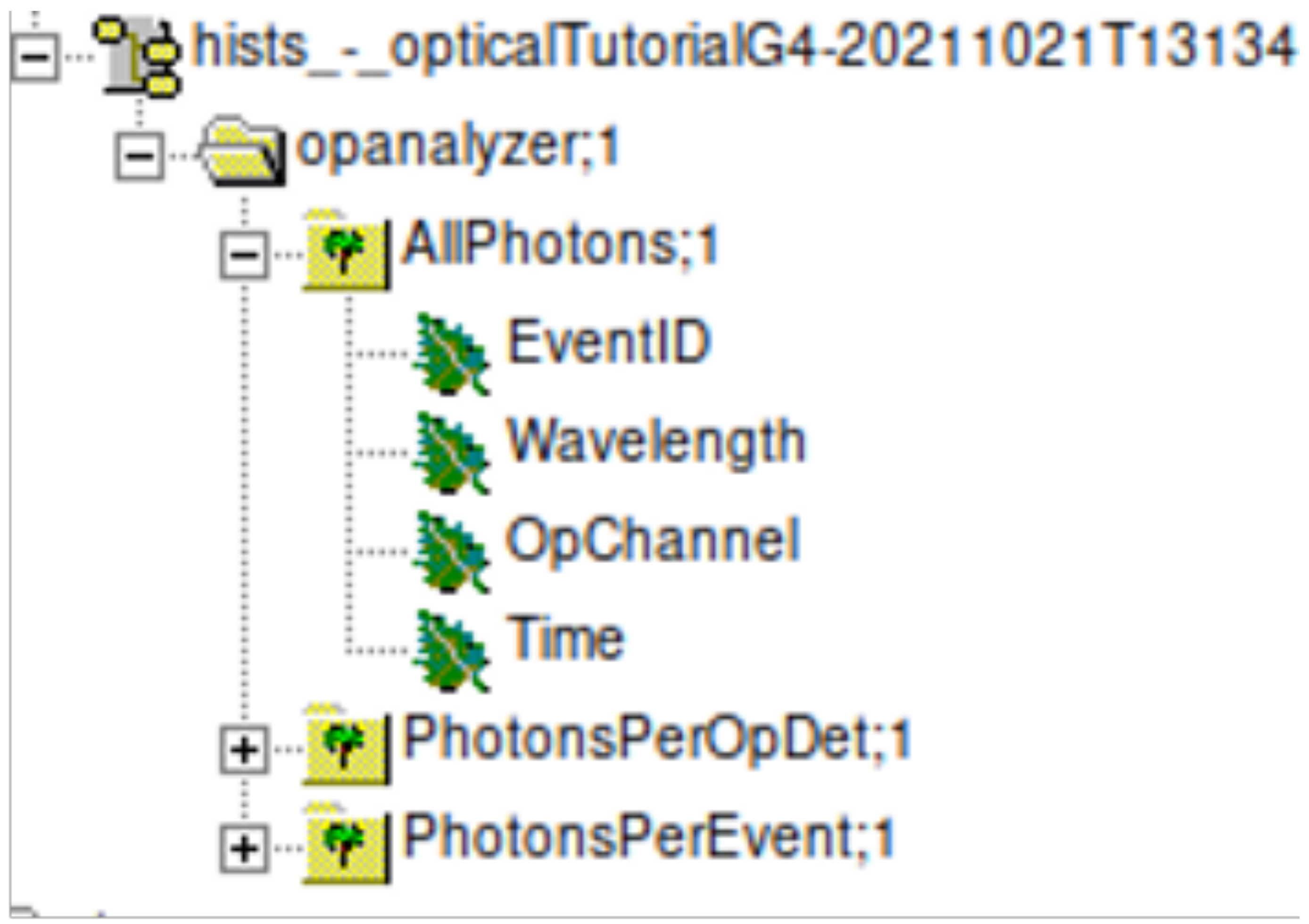
Task 1.1: running the light simulation

- In your photon_tutorial directory run optical_tutorial_sim_muons.fcl:

  lar -c optical_tutorial_sim_muons.fcl -n 1 ⟵ Generating one event

- OpDetAnalyzer will produce a _hist.root file containing three TTrees with truth-level information:

  AllPhotons – contains information about each photon
  PhotonsPerOpDet – number of photons arriving at each detector
  PhotonsPerEvent – total number of photons detected per event



- Take a look at the AllPhotons tree (use TBrowser):

  - do the OpChannel, Wavelength and Time plots make sense? (Instructions for making plots + an optical channel mapping can be found in the backup)

  - try to extract the slow timing constant of argon (hint: in TBrowser, tools –> Fit panel, then fit an exponential and look at 1 / slope)

To view the output hist file:

root -l sim_muons_G4_hist.root
new TBrowser

and click on the file from the list

Task 1.2: lets change the location of the muon

- The muons we just generated were at X = 100cm, about in the middle of one of the TPCs

- Towards the end of the optical_tutorial_sim_muons.fcl you will see the parameters of the generated particles:

  - X0, Y0, Z0: start coordinates of the particle

```
# generator parameters
physics.producers.generator.PadOutVectors: true
physics.producers.generator.PDG: [13]
physics.producers.generator.P0: [2.0] # GeV
physics.producers.generator.SigmaP: [0]
physics.producers.generator.PDist: 0
physics.producers.generator.X0: [100]
physics.producers.generator.Y0: [0]
physics.producers.generator.Z0: [150]
physics.producers.generator.T0: [0]
physics.producers.generator.Theta0XZ: [0]
```

- What happens if we move the muons to X0 = 25cm (by CPA) or 175cm (by APA)?

  - how does the total amount of light change? (look at the PerEvent tree)

  - how does the amount of VUV vs visible light change at different positions? Why is this? (there are separate branches for each)

> We can use the “-o” and “-T” options to change the output root file names larsoft, e.g.:
>
> *lar -c optical_tutorial_sim_muons.fcl -n 1 -o sim_muons_25cm.root -T sim_muons_25cm_hist.root*

Task 1.3: distribution of the light

- Lets look at how the photons are distributed using this macro:
  - *Workshop/Photon/macro/PlotPhotonsYZ.cc*
- First try running this using the muons from the previous task - does the distribution of the light make sense?
- Lets try generating some lower energy electrons at different positions in the detector (copy the fcl to your working directory):

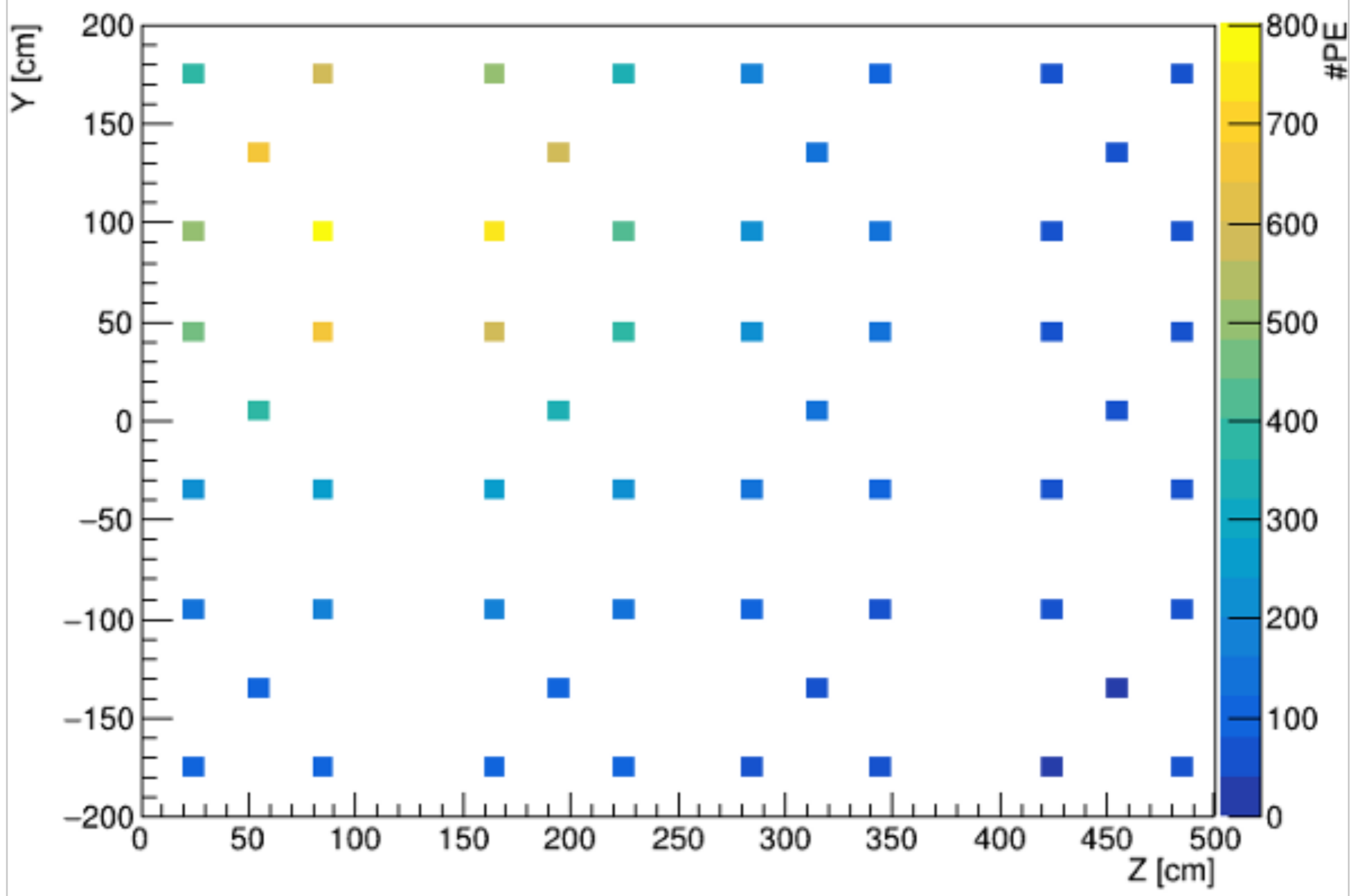
  lar -c optical_tutorial_sim_electrons.fcl -n 5

- How does the distribution of the light differ from the muons?
  - look at each event (they will be in different YZ positions)
- Bonus task: plot the direct and reflected light separately (modify the macro to plot CountDirect or CountReflected) – the reflected light is much more diffuse, why?
- Bonus task: plot the distribution of the light on the Arapucas (modify the macro to set !isPMT)

To run the root macro:

```
root -l
.L PlotPhotonsYZ.cc
PlotPhotonsYZ("sim_muons_G4_hist.root", 1)
```

Example result for a 50 MeV electron, each point represents a PMT in SBND



Part 1 summary

- You are now able to run simple light simulation jobs and have gained some understanding of what is happening in them
- There is of course a lot more that can be done with light, but that needs us to start looking at reconstruction of events
- One thing I did not cover is how the semi-analytic simulation is trained and how optical libraries are constructed. This is a bit more complicated, but tutorials/details can be found here:
    - https://cdcvs.fnal.gov/redmine/projects/sbn-analysis-group/wiki/Tutorial_3_Semi-Analytic_mode_How_to_generate_the_correction_curves
    - https://cdcvs.fnal.gov/redmine/projects/dunetpc/wiki/How_to_make_a_photon_library

Part 2: photo-detector response simulation and light reconstruction

Photo-detector response simulation

- We have determined the number of photons at truth level, now we need to model what a realistic photo-detector response would look like:
  - need to add electronics response, noise, etc.
  - module we're interested in: OpDetDigitizerSBND
- For this part of the tutorial we will need this fhicl: *optical_tutorial_detsim.fcl*
  - you can find this in the Workshop/Photon/fcl/ directory as before, copy this to your working directory
- This fhicl runs the standard detsim in SBND, along with an analyzer to let us look at the resulting waveforms

OpDetDigitizer module

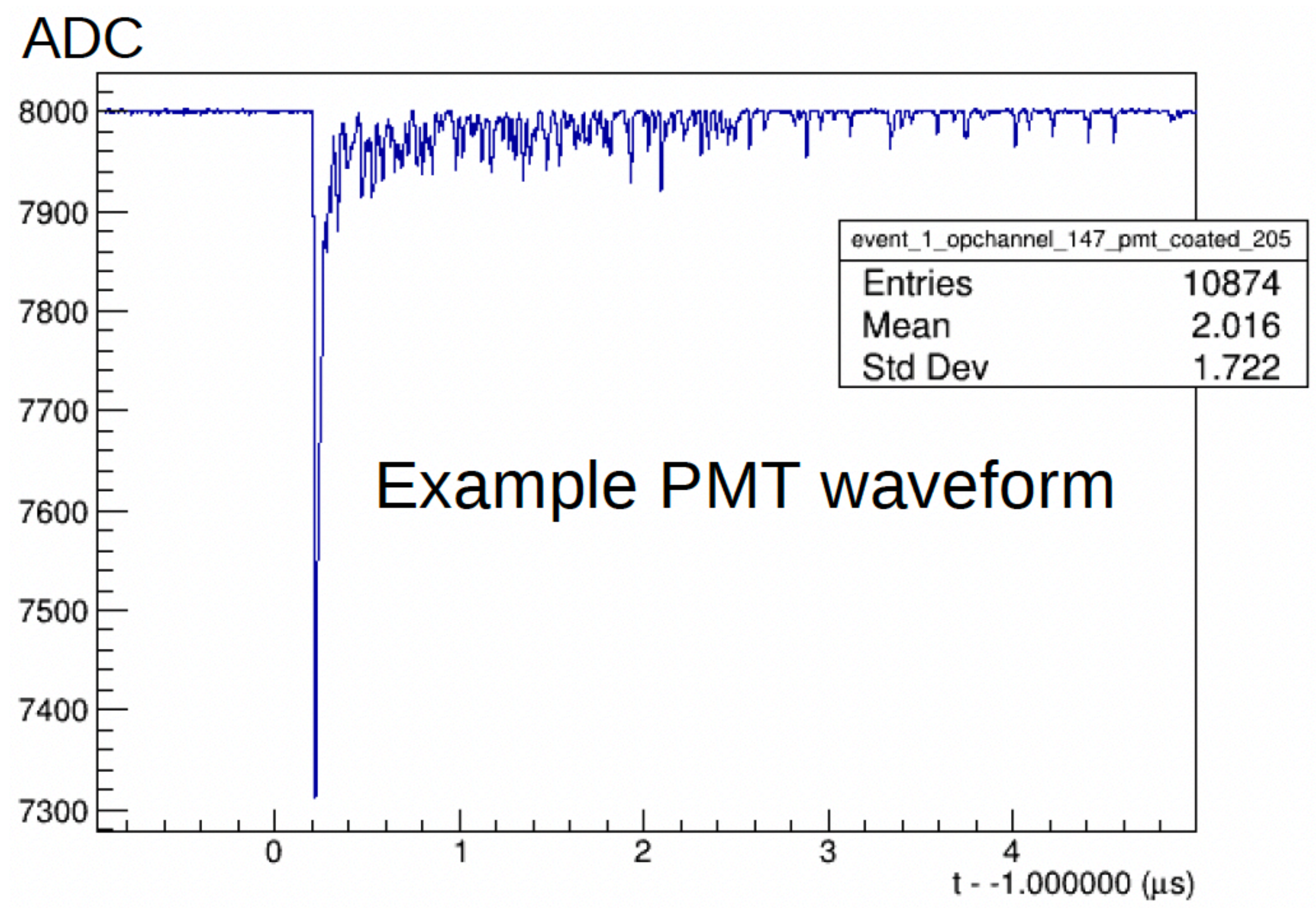
```
producers:
{
  rns:        { module_type: "RandomNumberSaver" }
  daq:        @local::sbnd_simwire
  opdaq:      @local::sbnd_opdetdigitizer
}

analyzers:
{
  wvfana: @local::wvf_ana
}
```

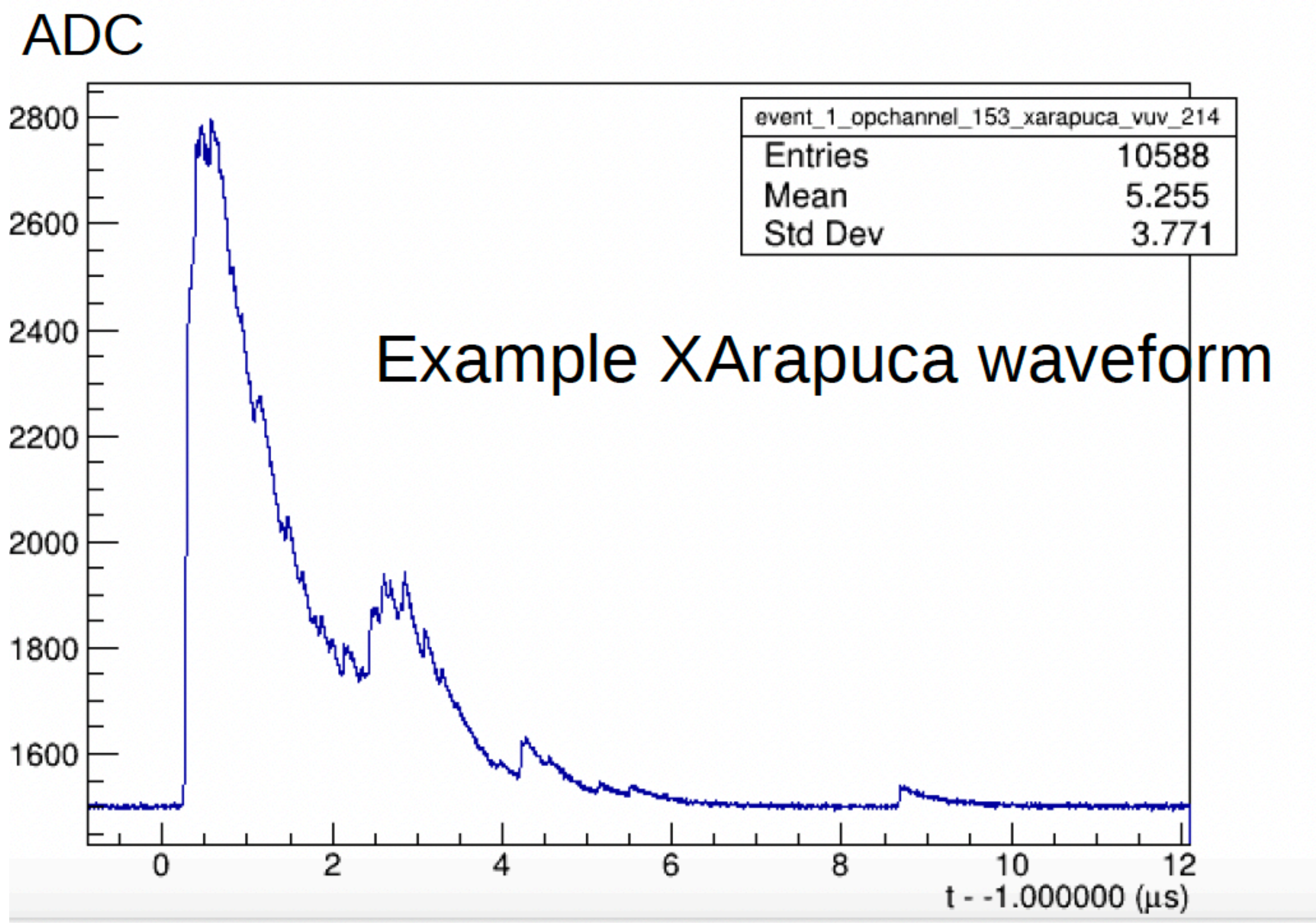
- Each PE swapped for an electronics response (here constructed from parameters). Noise then added to the waveform
- Analyzer will let us look at these waveforms

Different responses for PMTs / XArapucas:

```
sbnd_digipmt_alg:
{
  PMTRiseTime:             3.8         #ns
  PMTFallTime:             13.7        #ns
  PMTMeanAmplitude:        0.9         #in pC
  PMTBaselineRMS:          1.0         #in ADC
  PMTDarkNoiseRate:        1000.0      #in Hz
  TransitTime:             55.1        #ns
  TTS:                     2.4         #Transit
  CableTime:               135         #time de
  PMTChargeToADC:          -11.1927    #charge
  PMTSaturation:           300         #in numb
  PMTBaseline:             8000.0      #in ADC
```

```
sbnd_digiarapuca_alg:
{
  ArapucaVoltageToADC:          151.5    #mV to ADC
  ArapucaBaselineRMS:           2.6      #in ADC cou
  ArapucaDarkNoiseRate:         10.0     #in Hz
  CrossTalk:             0.2      #20% probability
  ArapucaBaseline:              1500     #ADC counts
  ArapucaPulseLength:           4000.0   #ns
  ArapucaPeakTime:              260.0    #ns
  ArapucaMeanAmplitude:         0.12     #mV
  ArapucaRiseTime:              9.0      #ns
  ArapucaFallTime:          476.0    #ns
  ArapucaSaturation:            300      #in number
```





Optical reconstruction

- Once we run OpDetDigitizer, our simulation will now be at a stage that resembles data we would get from a real-life photo-detector

- This means that we need to shift towards reconstructing the signals (and seeing how well this reconstruction reproduces the initial truth information)

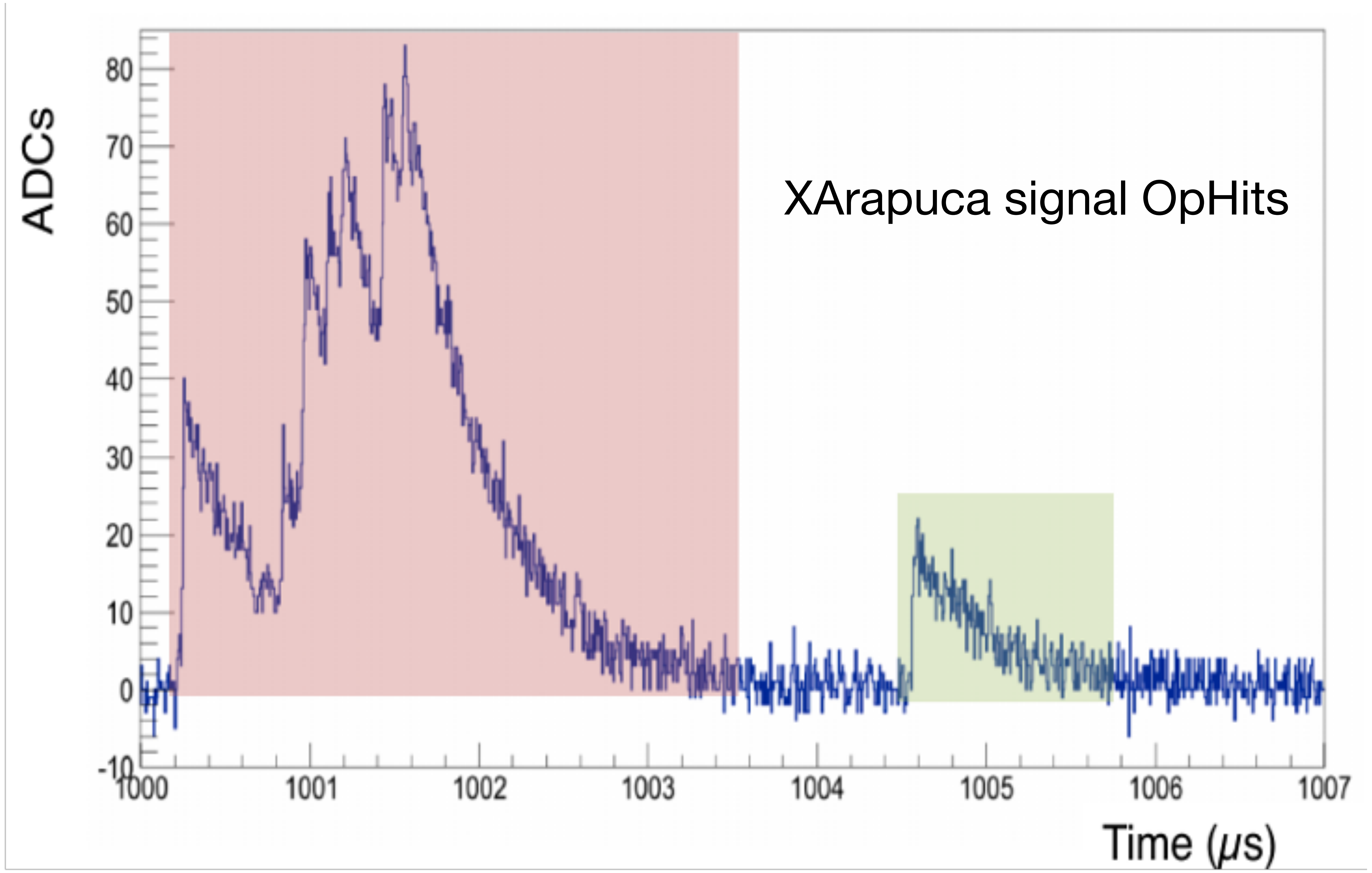
- For this part of the tutorial we will need this fhicl: optical_tutorial_reco.fcl

  - you can find this in the Workshop/Photon/fcl/ directory as before, copy this to your working directory

- This fhicl runs the standard optical reconstruction in SBND, along with a couple of analyzers to let us look at the resulting information

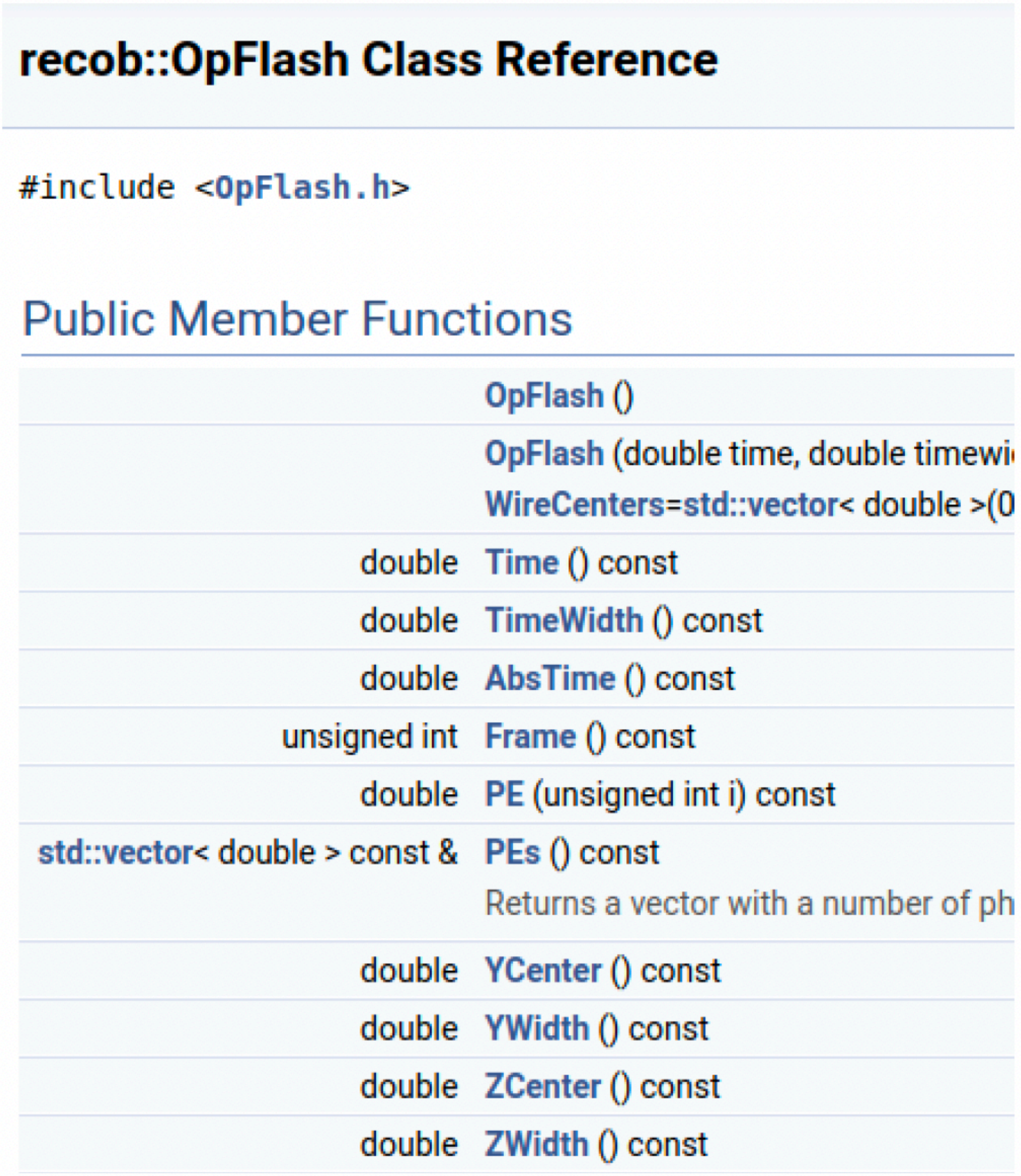
Optical hits

- First stage of reconstruction is finding optical hits (OpHits)
- OpHits are found when the waveform is above a certain threshold and held while it continues to be so.
- This can lead to the merging of visibly separate optical signals, especially in the case of SiPMs (in the XArapucas)
- The OpHit Time is decided by the first arriving photon



Optical flashes

- OpHits from different photon detectors are combined into Flashes
  - These are analogous to clusters in the charge reconstruction, but matched in time rather than space
- Having a flash allows us to try to reconstruct the position of the particle that generated the light (roughly)
- This can then be used to match the light signals to the reconstructed TPC tracks – Flash Matching



recob::OpFlash Class Reference

`#include <OpFlash.h>`

Public Member Functions

	OpFlash ()
	OpFlash (double time, double timewi WireCenters=std::vector< double >(0
double	Time () const
double	TimeWidth () const
double	AbsTime () const
unsigned int	Frame () const
double	PE (unsigned int i) const
std::vector< double > const &	PEs () const Returns a vector with a number of ph
double	YCenter () const
double	YWidth () const
double	ZCenter () const
double	ZWidth () const

optical_tutorial_reco.fcl

```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

  #Reconstruction @OpHit level for PMTs
  #Reconstruction @OpFlash level, TPC0 and TPC1
  producers:
  {
   ophitpmt:      @local::sbnd_ophit_finder_pmt
   opflashtpc0:   @local::SBNDSimpleFlashTPC0
   opflashtpc1:   @local::SBNDSimpleFlashTPC1
  }

  #Load analyzers
  #hitdumpertree from sbndcode/Commissioning/HitDumper_module
  #  Analyzer from larana/OpticalDetector
  analyzers:
  {
   oprecoanatpc0: @local::standard_opflashana
   oprecoanatpc1: @local::standard_opflashana
   hitdumpertree: @local::hitdumper
  }
```

- Produces OpHits and OpFlashes:
    - flashes produced separately for each TPC (recall SBND has two TPCs)
- Runs analyzer modules to look at OpHits and flashes in each TPC
- Note: only looking at the PMTs here for simplicity, XArapucas hit-finding is defined analogously

Flash matching

- The final stage is to perform matching between the reconstructed light information and the reconstructed TPC information (the next tutorials will cover that part!)

- In SBND we do this with the opt0finder module during Reco2:

```
reco2: [ rns
       , pandora, pandoraTrack, pandoraShower
       , pandoraCalo, pandoraPid
       , crthit, crttrack
       , opt0finder
```

```
sbnd_opt0_finder:
{
  module_type:      "SBNDOpT0Finder"
  OpFlashProducers: ["opflashtpc0", "opflashtpc1"]
  TPCs: [0, 1]
  SliceProducer:    "pandora"
```

- This module makes a prediction of the light based on the TPC track using the same simulation method as the LArG4 stage. This prediction is then compared with each OpFlash to find the best match.

- You will run the flash matching and make use of the flash timing information in the reconstruct/analysis tutorial (today/tomorrow)

Task 2.1: detector response simulation

- Run optical_tutorial_detsim.fcl using your muon from Task 1 as the input:

  lar -c optical_tutorial_detsim.fcl -s sim_muons_G4.root

- Take a look at the _hist.root file. The wvfana tree should contain waveforms for each photo-detector (there will be a lot of them!)

  - have a look at a few from PMTs and from XArapucas

  - try find some that see a lot of light and some that see very little (you can use the AllPhotons tree from previous task to get an idea of the channels to look at)

Pre-made files from the previous stage can be found here if needed:

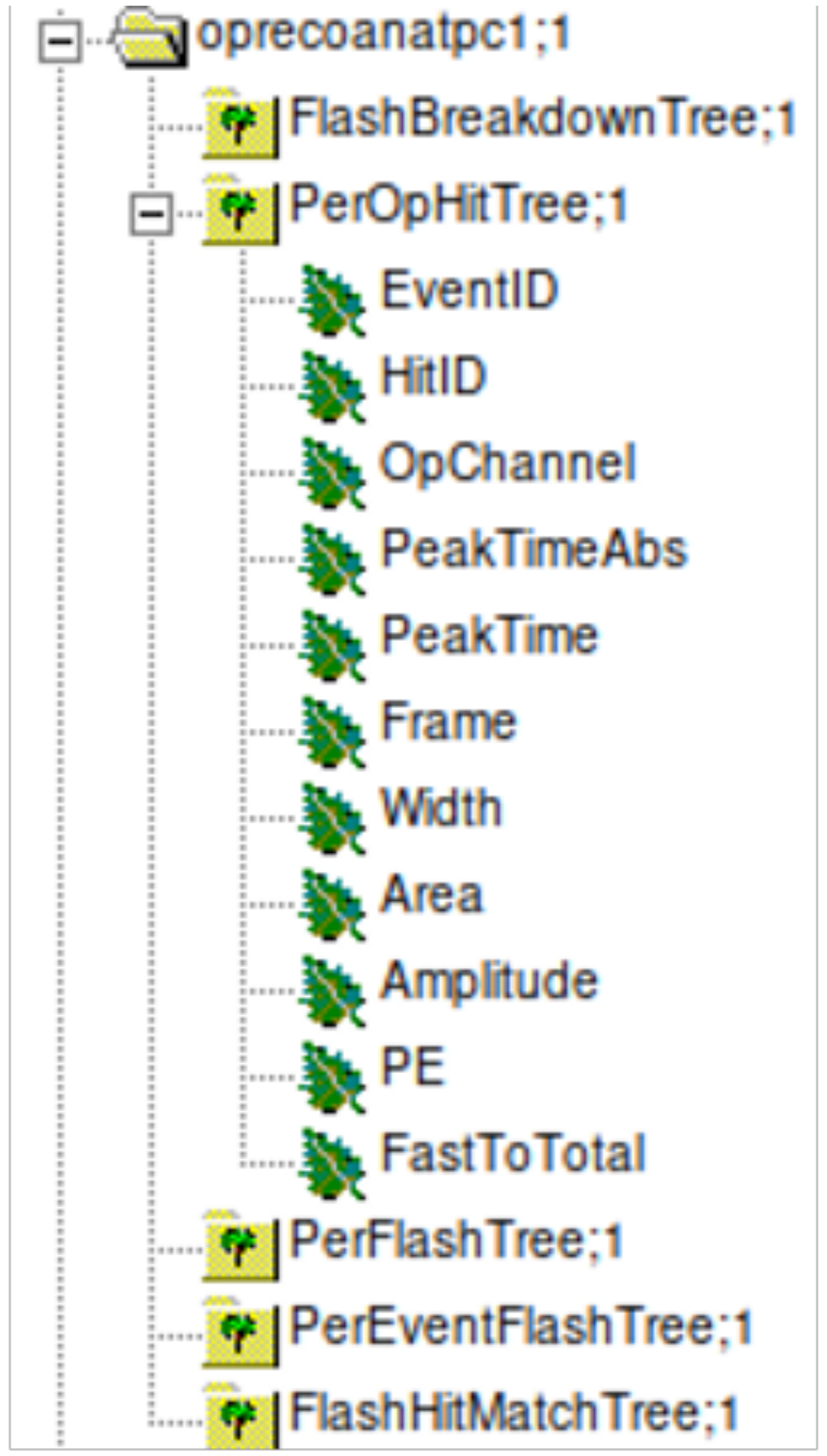
/home/share/november2022/photon/ (copy them to your directory!)

Task 2.2: optical reconstruction - hits

- Run optical_tutorial_reco.fcl using the output from the previous stage as the source:

  lar -c optical_tutorial_reco.fcl -s sim_muons_G4_Detsim.root

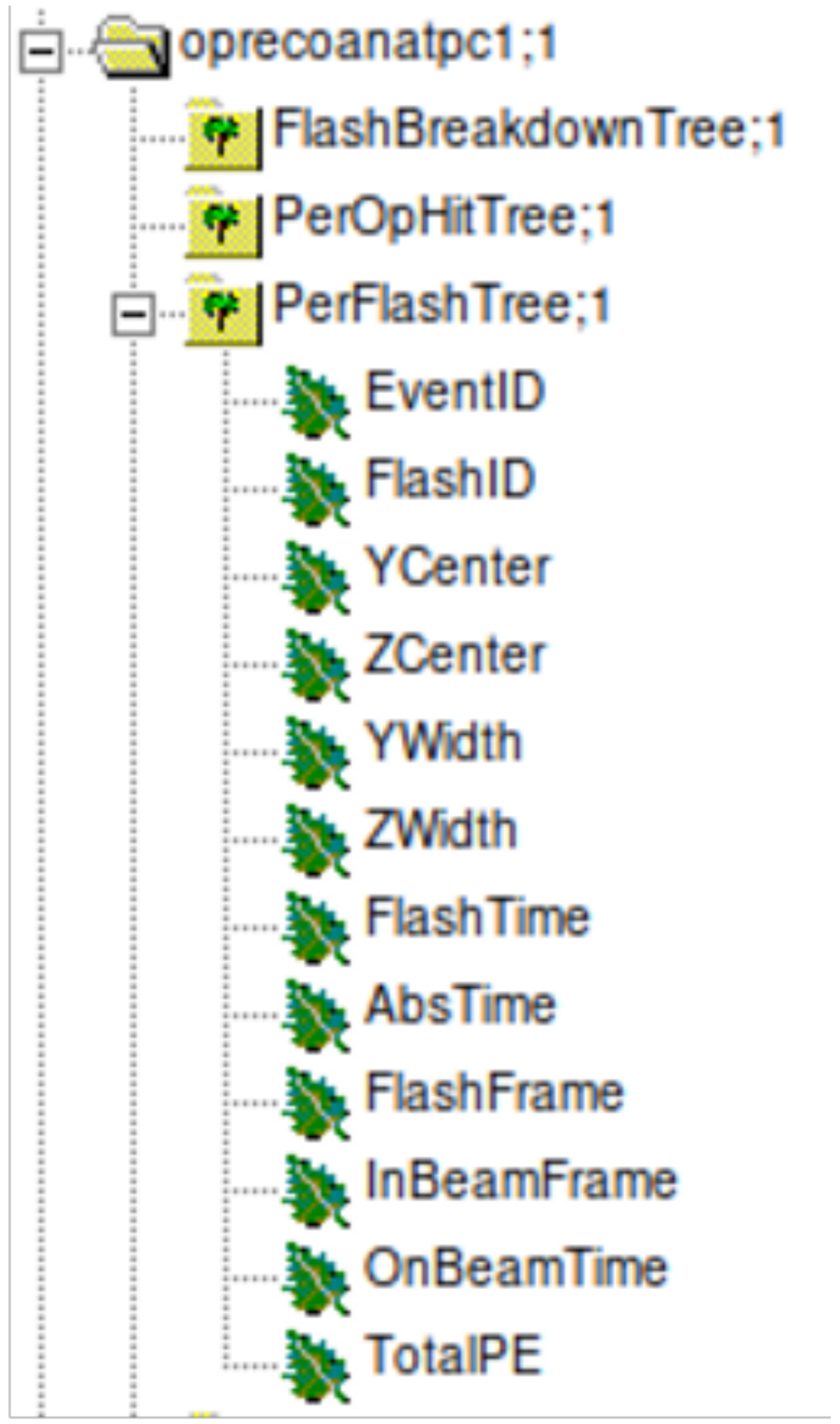
- Let's first take a look at the OpHits: (_hist file, oprecoanatpc1/PerOpHitTree)
  - take a look at the OpChannel and PE – do these make sense?
- Try plotting the hit Y-Z distribution:
  - a root macro to do this can be found here (copy it to your directory): /Workshop/Photon/macro/PlotOpHitYZ.cc and is run in the same way as PlotPhotonsYZ.cc
  - how does this compare with the equivalent plot at truth level? Is the OpHitFinder performing well?



> Pre-made files from the previous stage can be found here if needed:
> *\/home/share/november2022/photon/* (copy them to your directory!)

Task 2.3: optical reconstruction - flashes

- Still using the same output hist file, lets take a look at the Flashes
- Look at the oprecoanatpc1/PerFlashTree:
  - check where the flashes show up in the Y-Z plane. Is this where we expect them to be?
  - look at the flash widths – are they wider in Y or Z? Why?
- Bonus task: try doing the same for the electrons (you will need to run them through the detsim and reco stages too!):
  - is there any difference between the electron and the muon flashes?



Pre-made files from the previous stage can be found here if needed:
/home/share/november2022/photon/ (copy them to your directory!)

Part 2 summary

- You are now able to run simple light reconstruction in LArSoft and have hopefully gained some intuition for how the light behaves in LArTPCs
- There are a lot of things we can use this light information for to complement and enhance the TPC information (triggering and t0, event selection/background rejection, calorimetry, etc.).
- Hopefully this information / tools will help you to incorporate the light into your own analyses.
- Thanks!

Backups

Making plots

- The visual way:
  - root -l <my_file>_hist.root
  - new TBrowser()
  - Find the name of your .root file in the list
  - Select opanalyzer, select AllPhotons, right click on AllPhotons and select StartViewer.
  - You can plot any of the branches and apply cuts

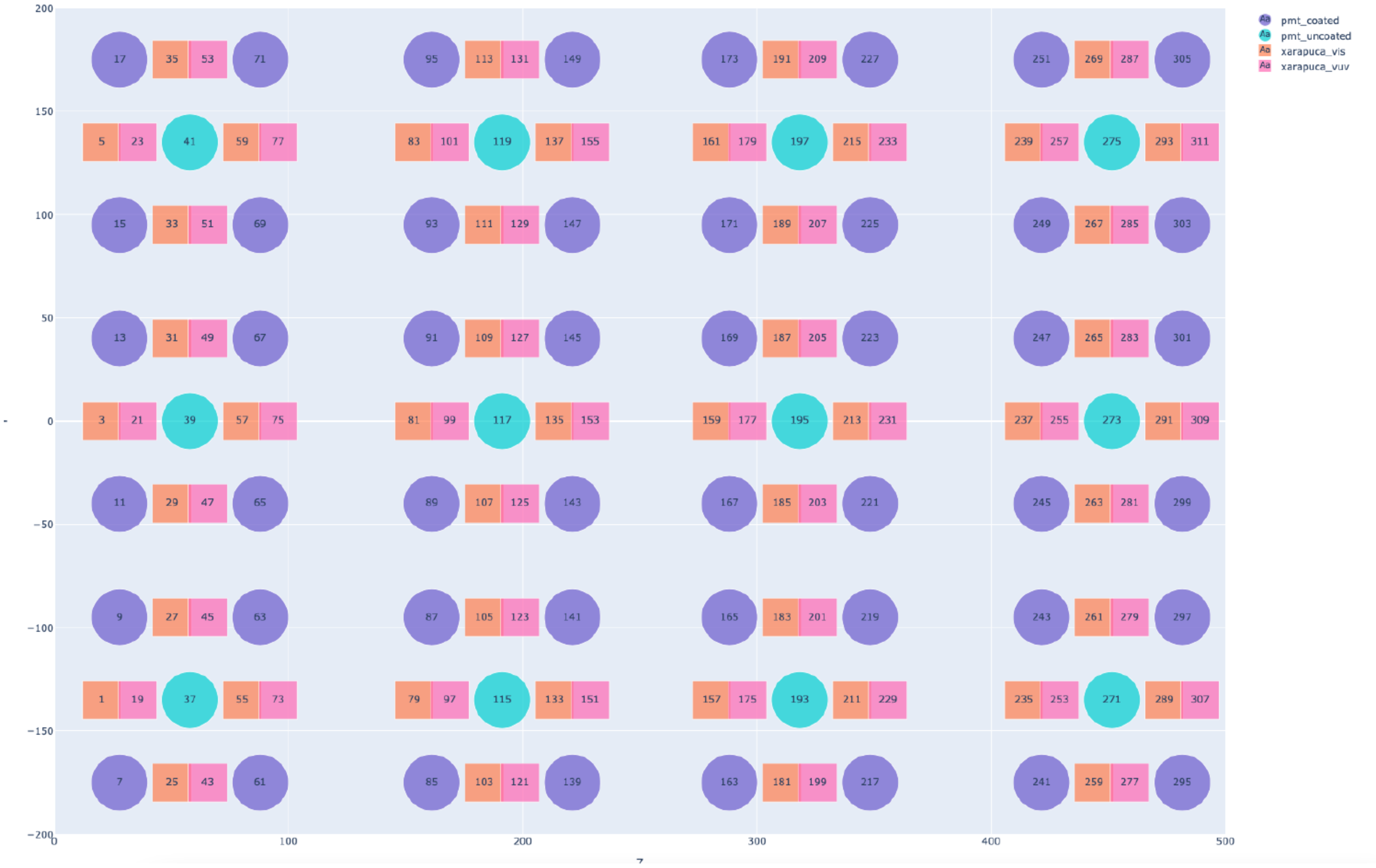
Making plots

- The script way:
  - Create a new file called myScript.C
  - In it write:

```
void myScript()
{
   TFile * fin = new TFile("<myfile>_hist.root","READ");
   TTree * mytree = (TTree *)fin->Get(“opanalyzer/AllPhotons");
   mytree->Draw("Time","");
}
```

  - Then to run: root -l myScript.C

Photon detector mapping: TPC 1, x > 0 cm



Photon detector mapping: TPC 0, x < 0 cm