Understanding Scintillation Reconstruction (and electronics sim)

Andrzej Szelc

Using material from Alex Himmel's DUNE tutorials, found here:
https://cdcvs.fnal.gov/redmine/projects/dunetpc/wiki/Photon_Simulation_Tutorial

Intro

- In this lecture/tutorial we will look at simulating scintillation through the OpDetDigitizer Stage.
- We will then take a look at OpHits
- Then we will take a look at how OpHits can be combined into flashes.
- Finally, we will talk about flash-matching.
- This will be a lecture/tutorial depending on how quickly we will be going.

Reminder: Log in to Your Computing Account

- ssh -X -Y user01@18.231.121.254
- Password is written on the whiteboard
- Most files you will need will be in /home/andrzej/workshop_files/

Where we are now

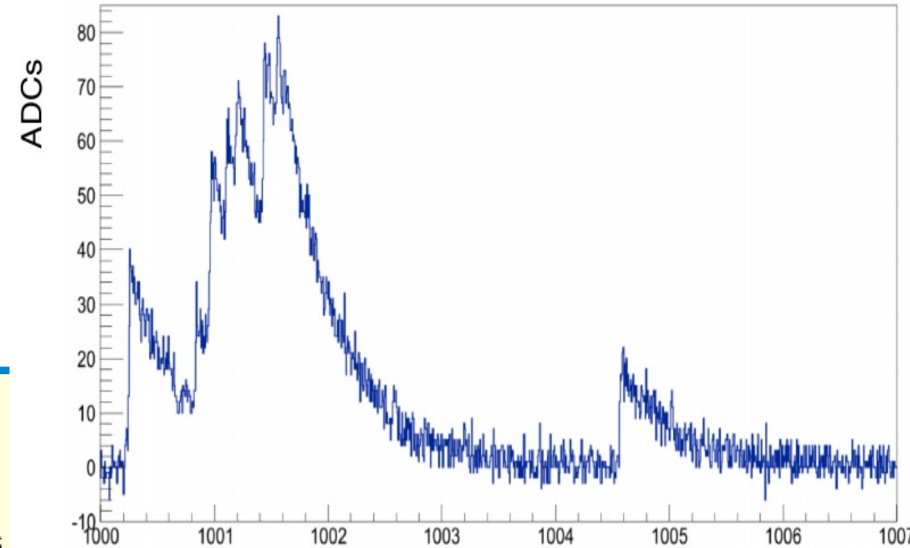
- We have understood some variants of the FastScintillation part of the simulation.
- We haven't even really brought it to the point of seeing how to simulate events to look like acquires through real electronics.
- So far we have a list of photons and their arrival times. Need to add electronics response, noise etc...

MANCHESTER 1824

The University of Manchester

OpDetDigitizer

- Module name we're interested in: `OpDetDigitizerDUNE`
- Lives in dunetpc.




```
# Assume 25 V bias with Sensl C-series SiPMs
# Gain at this voltage is 4e6 -- that this corresponds to
# the MaxAmplitude and VoltageToADC below has not been confirmed.

VoltageToADC:        151.5      # Converting mV to ADC counts (counts
LineNoiseRMS:          2.6      # Pedestal RMS in ADC counts, likely a
DarkNoiseRate:        10.0      # In Hz, Ranges 2-50 depending on Vbias
CrossTalk:             0.20     # Probability of producing 2 PE for 1 incident photon
# Afterpulsing:        0.006    # Afterpulsing is not yet simulated
Pedestal:           1500        # in ADC counts
DefaultSimWindow:   true        # Use -1*drift window as the start time and
                                # the TPC readout window end time as the end time
FullWaveformOutput: false       # Output full waveform. Be careful with this option:
                                # setting it to "true" can result in large output files

TimeBegin:             0        # In us (not used if DefaultSimWindow is set to true)
TimeEnd:            1600        # In us (not used if DefaultSimWindow is set to true)
PreTrigger:          100        # In ticks
ReadoutWindow:      1000        #
algo_threshold: @local::standard_al
```

```
dunefd_opdigi_threegang:               @local::dunefd_opdigi_unganged
dunefd_opdigi_threegang.PulseLength:   0.876
dunefd_opdigi_threegang.PeakTime:      0.028
dunefd_opdigi_threegang.MaxAmplitude:  0.0594
dunefd_opdigi_threegang.FrontTime:     0.013
dunefd_opdigi_threegang.BackTime:      0.386
```

Each PE gets swapped For an electronics response (here Constructed from Parameters)
Noise then added to waveform

Added effects like cross-talk, etc..

The University of Manchester

MANCHESTER 1824

Task 1:

- You will need the file: `optical_tutorial_digi.fcl`

```
# modules are grouped by type.
physics:
{

   producers:
   {
      opdigi:    @local::dunefd_opdigi_threegang    # simple digitizer with no noise and high saturation
      rns:       { module_type: "RandomNumberSaver" }
   }

   analyzers:
   {
      opdigiana:   @local::dunefd_opdigiana
      #ophitana:    @local::dunefd_ophitana
   }
```

```
## DUNE geometry with foils and cryostat.
services.Geometry.GDML: "dune10kt_v4_1x2x6.gdml"
services.Geometry.ROOT: "dune10kt_v4_1x2x6_nowires.gdml"
#
#
```

Note that we don't
Need the library anymore.
It's done its job.
Not clear whether we need the
geometry at this stage

Task 1:

- Run `optical_tutorial_digi.fcl`

  use your marley and muons file as source:
- `lar -c optical_tutorial_digi.fcl -s data_marley.root -o marley.digi.root`
- `Lar -c optical_tutorial_digi.fcl -s <name_of_muonfile>.root -o muon_digi.root`
- Take a look at the _hist.root file. It contains waveforms for each events and PhotonDetector.
- Check that the photon detector you expect to have the most light does indeed.

Bonus Task:
Calculate the average waveform and see what the timing constant is. (hard given how much time we have left constraints)

Optical Reconstruction

- Our simulation is now at a stage that resembles data we would get from a real-live detector.
- This means that we need shift towards reconstructing the signals and seeing how well this reconstruction reproduces the initial step.
- First step of this reconstruction is the OpHit.

recob::OpHit Class Reference

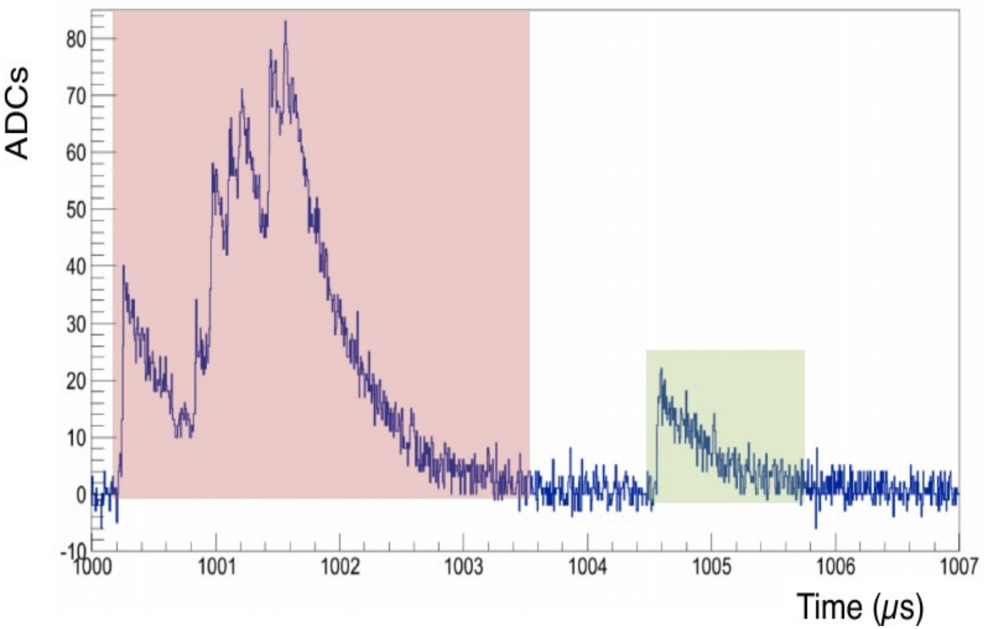
```
#include "OpHit.h"
```

Public Member Functions

	OpHit ()
	OpHit (int opchannel, double peakt fasttototal)
int	**OpChannel** () const
double	**PeakTimeAbs** () const
double	**PeakTime** () const
unsigned short	**Frame** () const
double	**Width** () const
double	**Area** () const
double	**Amplitude** () const
double	**PE** () const
double	**FastToTotal** () const

OpHits

- OpHits are found when the waveform is above certain threshold and held until continues to be so.
- Especially for SiPM signals this can lead to merging of visibly separate optical signals.
- OpHit Time is decided on the first arriving photon.



Flashes

- OpHits from different photon detectors are combined in to flashes. They would be like clusters, except currently they are matched in time rather than space.
- Having a Flash allows to try to reconstruct the position of a particle that generated the light (very roughly)
- In principle can use this to match the light signals to original TPC tracks.

Task 2

- You will need the file:
  `optical_tutorial_reco.fcl`
  run it on the _digi.root files you just generated earlier (do it for both the marley_digi.root file and muon_digi.root file).
- This file will run both OpHits as well as flashes.
- Plus an ana module.

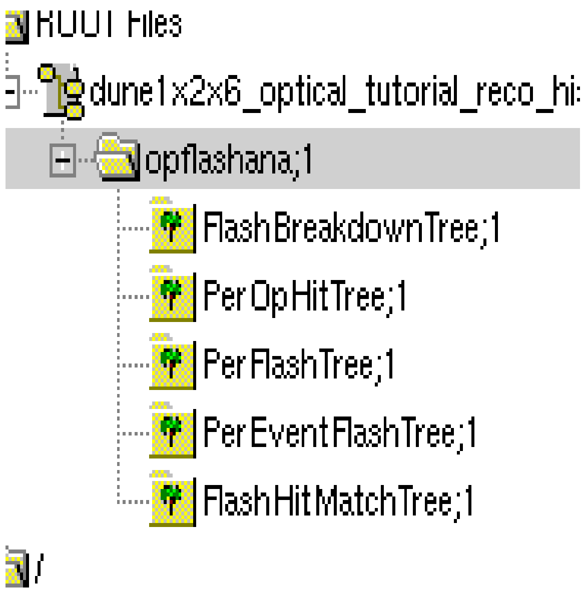
```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

   producers:
   {
      # photon detector reconstruction
      ophit:      @local::dunefd_ophit
      opflash:    @local::dunefd_opflash
      rns:        { module_type: "RandomNumberSaver" }
   }

   analyzers:
   {
      opflashana:  @local::dunefd_opflashana
   }
```

Task 2 cont'd

- Let's first take a look at PerOpHitTree.
- Make plots of the PeakTime, OpChannel and PE.
- Overlay them with the MCTruth results we had earlier.
- How is the OpHitFinder performing?



MANCHESTER 1824

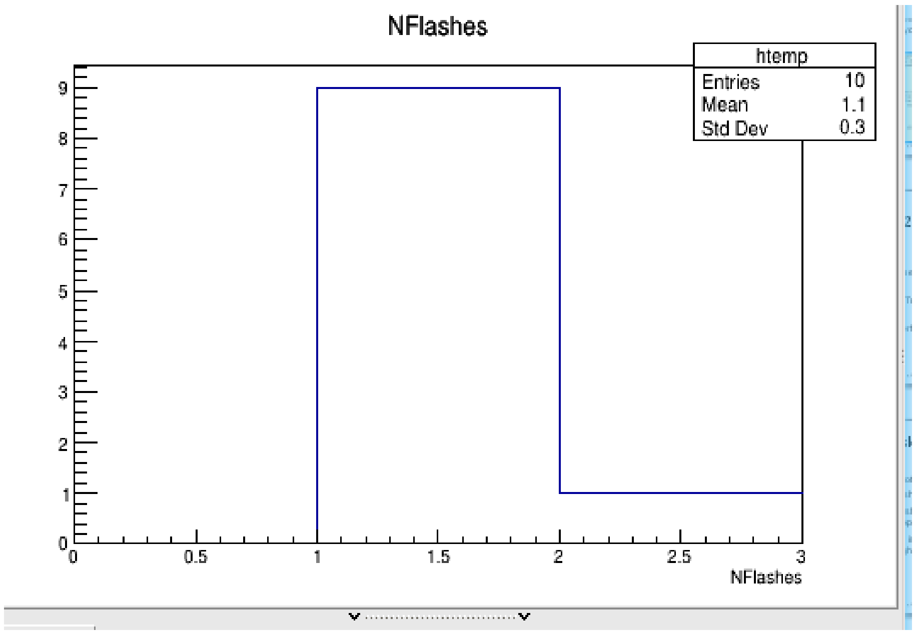
The University of Manchester

Task 2.1

- Still looking at this new root TTree.
- Take a look at the PerFlashTree.
- Check where the flashes show up in the Y-Z plane. Is this where we expect them to be?
  - The muons file may be the better one to try to find something.
- Look at the flash widths – is it wider in Y or Z? Why do you think that might be?

Task 2.2

- Take a look at the PerEventFlashTree.
- This takes into account the fact that there may have been multiple Flashes in an event.
- Check whether this happened – if yes, what could be the cause?
  - Can you diagnose it using the other parameters of the light?

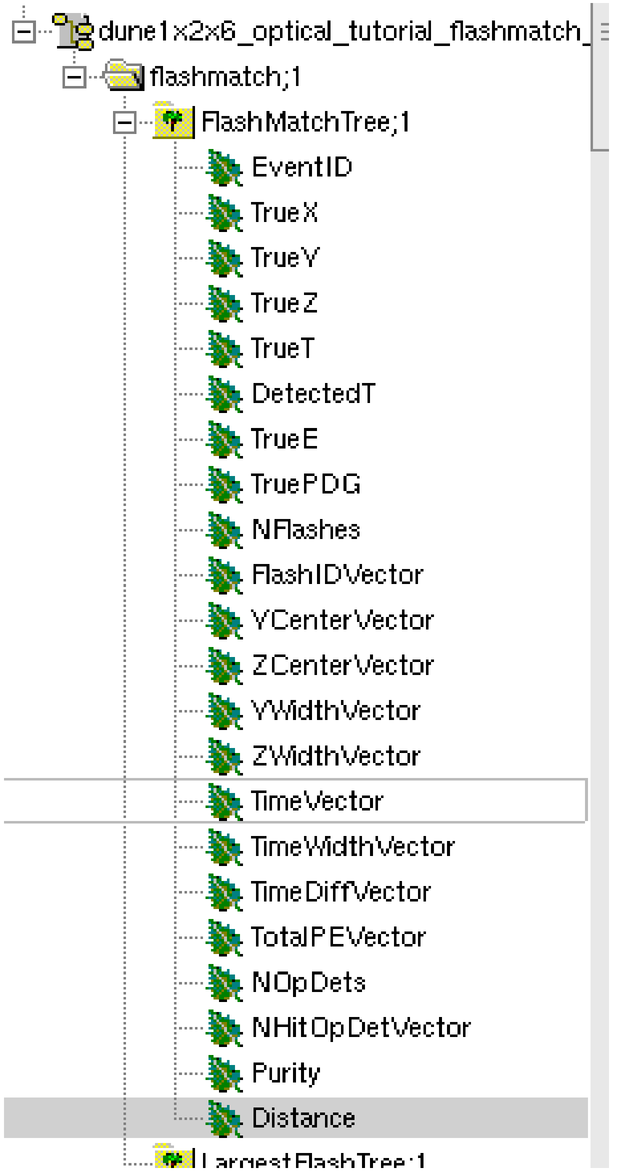


FlashMatching

- At the end of the day what we'd like to be doing is to understand where our flashes came from and how good are we at finding them.
- There is a module that helps us do that: FlashMatchAna.
- This takes the flash coordinates from the flashes and matches them to the true information.

Task 3

- You will need the file: `optical_tutorial_flashmatch.fcl` run it on the .root files you just generated after running reco. I tested it on marley. I don't guarantee that it will work on the muons.
- Open the resulting _hist.file



Task 3 Cont'd.

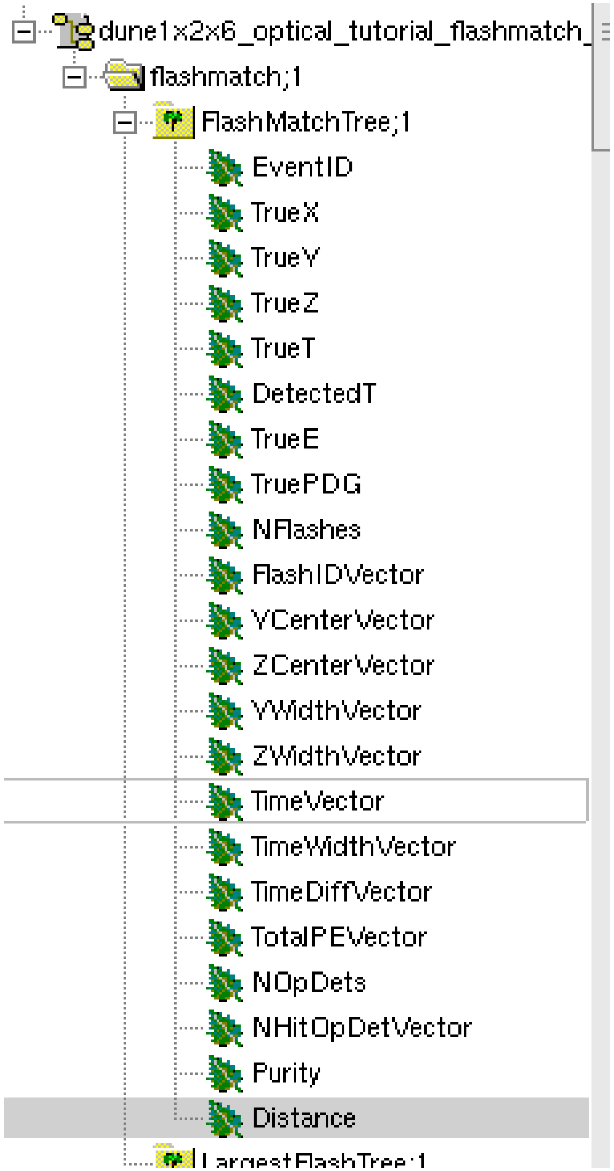
```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{
 analyzers:
 {
    # Configuration specific to supernova events.
    # Switch to standard_flashmatchana for beam or NDK
    flashmatch:      @local::marley_flashmatchana
 }
 ana:  [ flashmatch ]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [ana]
}
```

```
## DUNE geometry with foils and cryostat.
services.Geometry.GDML: "dune10kt_v4_1x2x6.gdml"
services.Geometry.ROOT: "dune10kt_v4_1x2x6_nowires.gdml"
#
#
```

Task 3 cont'd

- We now have access to a truth variables of where the events were generated, as well as the reconstructed quantities.
- Take a look at the NFlashes.
- We can make a TrueY-YCenterVector[0] plot, and the same for Z.
- How well are we doing? What if we limit to events with one flash?
- Look at Flash Widths – how big are they?
- How different is the detected time from the original time – is that reasonable?
- Does the flash efficiency vary with distance?



The University of Manchester

Conclusions

- At this point you should know how to run simple scintillation light reconstruction.
- And how to try to see whether it works or not.
- There is a lot more things that you should be aware of still, but you should hopefully know where to start looking for hints.
- Use this info to make your own awesome analyses.

Save the date,
Details coming soon.

LIDINE 2019

Manchester, UK

28-30, Aug, 19

Light Detection in Noble Elements

Backup/Tips

Making some plots

- Visual way:
  - root -l <my_file>_hist.root
  - `new TBrowser()`
  - Find the name of your .root file in the list
  - Select pmtresponse, select DetectedPhotons, right click and select treeviewer.
  - You can plot any of the branches and apply cuts.

Making some plots (2)

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

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

  - Run: `root -l myScript.C`