Light simulation / reconstruction tutorial

Andrzej Szelc

The University of Edinburgh

Based on Patrick Green's and Fran Nicolas's UK-LArSoft light simulation tutorial:
https://indico.ph.ed.ac.uk/event/91/contributions/1410/attachments/891/1203/light_simulation_tutorial.pdf

& Alex Himmel's & Laura Paulucci's DUNE tutorials:
https://wiki.dunescience.org/wiki/Photon_simulation_tutorial

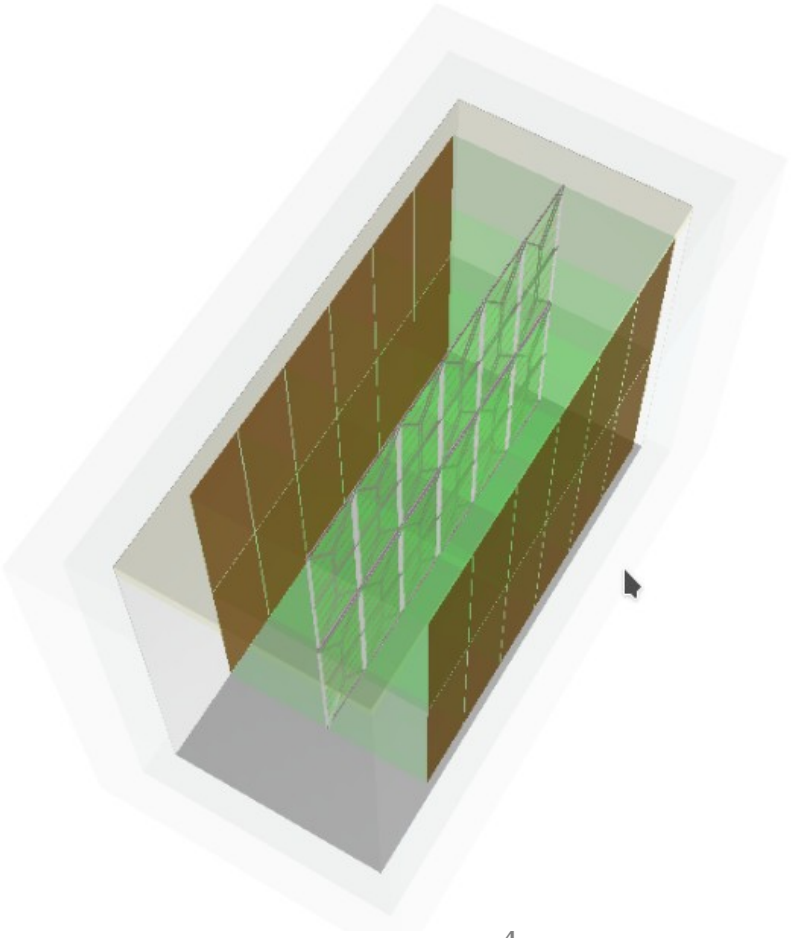
Introduction

- In this tutorial we will look at simulating and reconstructing the scintillation light
- We will take a look at how DUNE-FD fast optical simulation works and what the results look like for particles traveling through the detector at truth level
- We will then look at running the OpDetDigitizer stage and what the signals look like on each optical detector
- Then we will run the optical reconstruction, and look at the OpHits and how they can be combined into Flashes
- Finally, we will talk a little bit about flash matching

Part 1: simulating the light through the LArG4 Stage

Reminder: working with the DUNE 1x2x6 geometry

- Two TPCs, separated by an APA with double-sided light detectors (center, x=0)
- Light detectors at each APA, for ARAPUCA super-cells per APA.
  - X-Arapucas: see light from interactions occurring both TPCs



Running the light simulation in SBND

- We will be using this fhicl: *optical_tutorial_sim_muons.fcl*
  - you can find this fhicl in *Workshop/Photon/fcl/* in your local duneana install
  - This fhicl will generate 2 GeV muons at a certain position in the detector – don't run it just yet.

- 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 DUNE-FD (Legacy Version)

Legacy LArG4:

```
physics:
{

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

    # Generation
    generator: @local::microboone_singlep

    # The geant4 step
    largeant: @local::dunefd_largeant



  }

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

Prod single generator

Geant4 simulation of light AND charge.
(in re-factored G4, you can choose only one).

Configuration of light simulation (semi-analytic or library)

```
# Hits & Timing parameterization for DUNE FD, Ar scintillation
dunefd_pdfastsim_par_ar:                        @local::standard_pdfastsim_par_ar
dunefd_pdfastsim_par_ar.VUVTiming:              @local::dune_vuv_timing_parameterization
dunefd_pdfastsim_par_ar.VUVHits:                @local::dune_vuv_RS100cm_hits_parameterization

# As above, with cathode reflections included
dunefd_pdfastsim_par_ar_refl:                   @local::dunefd_pdfastsim_par_ar
dunefd_pdfastsim_par_ar_refl.DoReflectedLight: true
dunefd_pdfastsim_par_ar_refl.VISTiming:         @local::dune_vis_timing_parameterization
dunefd_pdfastsim_par_ar_refl.VISHits:           @local::dune_vis_RS100cm_hits_parameterization
```

Note, we are simulating light reflecting off-of the cathode, even though this is currently not planned for DUNE-HD

Also include an analyzer that will allow us to access the truth level information: (SBND-specific originally but works in DUNE after minor changes).

Task 1 preparation

- We will be using the same duneana installation from before
- We need to re-setup the environment, and re-compile as I've added new files.
  - connect to the viewer as before and open a new terminal (http://py-dom.lancs.ac.uk:8080/guacamole/#/)
  - next set up your local dunetpc installation:

```
source /cvmfs/dune.opensciencegrid.org/products/dune/setup_dune.sh

source source localProducts_larsoft_v09_56_00_e20_prof/setup

      cd $MRB_SOURCE/duneana;  git pull
      cd $MRB_BUILDDIR
      mrbsetenv
      mrb i -j8
      mrbslp
```

- Make a new empty directory called photon_tutorial (or whatever you like) to work in and copy *`optical_tutorial_sim_muons.fcl`* to this directory:
  - from `srcs/duneana/duneana/Workshop/Photon/fcl/`

Task 1.1: running the light simulation and looking at the results

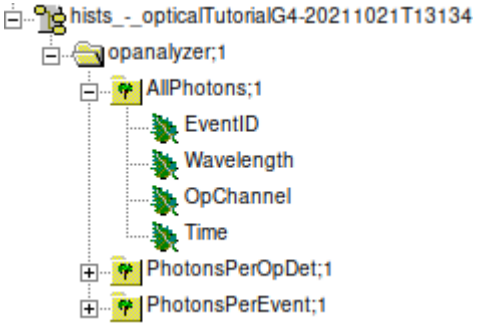
Generating 3 events

- Lets run *optical_tutorial_sim_muons.fcl*:

```
-lar -c optical_tutorial_sim_muons.fcl -n 3
```

- The OpDetAnalyzer will produce an *_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



To view the output hist file:

root -l sim_muons_G4_hist.root
new Tbrowser

and click on the file from the list

- Take a look at the AllPhotons tree (use TBrowser):
  - do the OpChannel, Wavelength and Time plots make sense? (can you try to figure out the channel mapping?)
  - try to extract the slow timing constant of argon (hint: in TBrowser, tools –> Fit panel, then fit an exponential and look at 1 / slope)

Task 1.2: lets change the muon location

```
# 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: [-180]
physics.producers.generator.Y0: [300]
physics.producers.generator.Z0: [50]
physics.producers.generator.T0: [0]
physics.producers.generator.Theta0XZ: [0]
physics.producers.generator.Theta0YZ: [0]
physics.producers.generator.SigmaThetaXZ: [0]
physics.producers.generator.SigmaThetaYZ: [0]
```

- The muons we just generated were at X = -180cm, 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 coords of the particle
- What happens if we move the muons to X0 = -330cm (by CPA) or -25cm (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) – the reflected light is much more diffuse, why?*

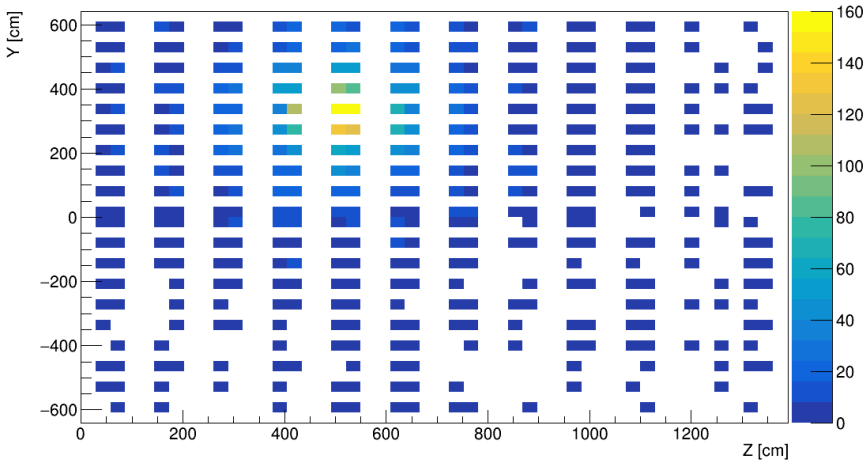
To run the root macro:

EventID to plot

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

Do this in a new terminal to use the local root install rather than cvmfs (much faster)

Example for a 50 MeV electron, each point represents an X-ARAPUCA supercell



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, which will be next
- 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: detector response simulation and light reconstruction

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: OpDetDigitizerDUNE
- 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 DUNE, along with an analyzer to let us look at the resulting waveforms

OpDetDigitizer

```
producers:
{
  rns:       { module_type: "RandomNumberSaver" }
  opdigi:          @local::dune35t_opdigi
  daq:             @local::dune_detsim
}

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

Each PE swapped for an electronics response (here constructed from parameters). Noise then added to the waveform.

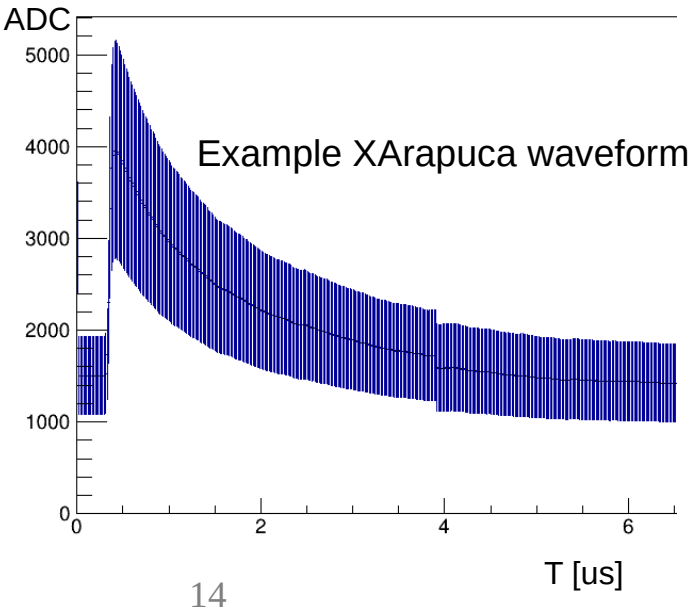
Analyzer will let us look at these waveforms

Responses for XArapucas:

```
dunefd_opdigi_threegang:                                    @local::dunefd_opdigi_unganged
dunefd_opdigi_threegang.PulseLength:                  5.2
dunefd_opdigi_threegang.PeakTime:                       0.028
dunefd_opdigi_threegang.MaxAmplitude:                 0.0594 # * VoltageToADC = 9 ADC/PE
dunefd_opdigi_threegang.FrontTime:                      0.013
dunefd_opdigi_threegang.BackTime:                       0.386
dunefd_opdigi_threegang.algo_threshold.ADCThreshold: 15.000 # "2 PE" threshold = 1.7 PE

dunefd_opdigi_threegang_refactor: @local::dunefd_opdigi_threegang
dunefd_opdigi_threegang_refactor.InputModules: ["PDFastSim"]

dunefd_opdigi: @local::dunefd_opdigi_threegang
```

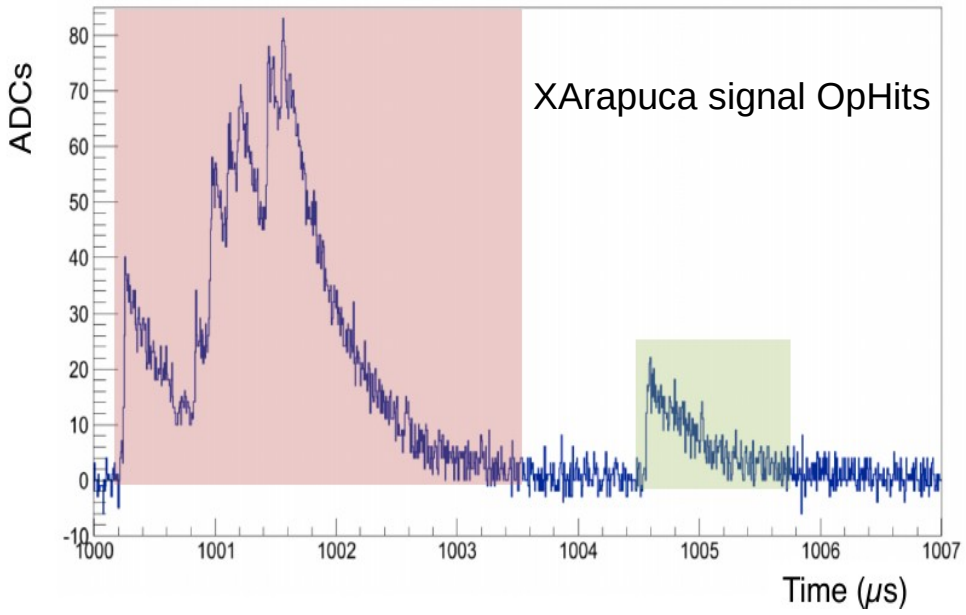


Optical reconstruction

- Our simulation is now at a stage that resembles data we would get from a real-life 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

OpHits

- First stage of reconstruction is 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 X-Arapucas)
- The OpHit Time is decided by the first arriving photon



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 from photon detectors
  producers:
  {
    ophit:     @local::dunefd_ophit
    opflash:   @local::dunefd_opflash
    rns:       { module_type: "RandomNumberSaver" }
  }

  #Load analyzers
  #  Analyzer from larana/OpticalDetector
  analyzers:
  {
    opflashana:  @local::dunefd_opflashana
    hitdumper:   @local::hitdumper
  }
```

- Produces OpHits and OpFlashes:
- Runs analyzer modules to look at OpHits and flashes in each TPC
- Dumps Hit Coordinates (mutilated SBND module – use with care).

Task 2: running the 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 the total average and the single X-Arapuca events

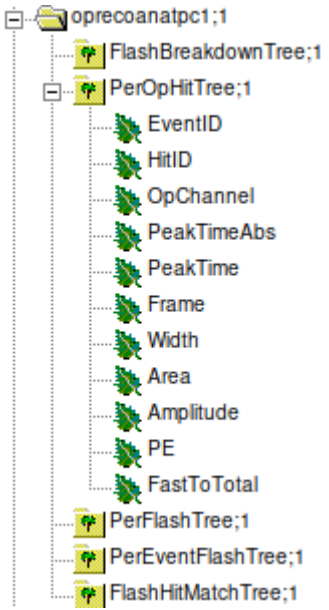
- Check that the optical detectors you expect to a lot of light do indeed (you can use the AllPhotons tree from the previous task to get an idea of the channels to look at).

Pre-made files from the previous stage can be found here if needed:
/home/share/september2022/photon/
(copy them to your directory)

Task 3.1: 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
```

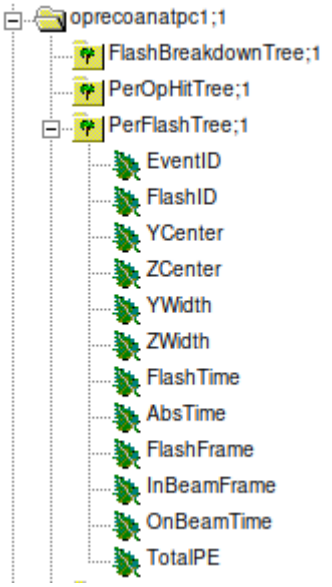


- Let's first take a look at the OpHits: (_hist file, o*pflashana/PerOpHitTree)*
  - take a look at the OpChannel and PE – do these make sense?
- Try plotting the hit Y-Z distribution:
  - a root macro to do this can be found here (copy it to your directory): /Workshop/Photon/macro/PlotOpHitYZ.cc
  - 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/september2022/photon/
(copy them to your directory)

Task 3.2: optical reconstruction - flashes

- Still using the same output hist file, lets take a look at the Flashes
- Look at the *opflashana/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 electrons and the muons?



Pre-made files from the previous stage can be found here if needed:
`/home/share/september2022/photon/`
(copy them to your directory)

Flash Matching

- The final stage is to perform matching between the reconstructed light information and the reconstructed TPC information (as you heard in the pandora tutorials!):

  - in SBND we do this with the opt0finder module after Reco2:

```
13  process_name: Reco2
14
15  physics.trigger_paths: [reco2 #, opticalt0
16                          ]
```

```
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.

- Unfortunately I haven't managed to get it working in DUNE (because ran out of time).

Conclusions

- 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, 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 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:

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

  - Run: `root -l myScript.C`

Light simulation in legacy LArG4

- Many experiments still use legacy LArG4 (uBooNE, ICARUS, DUNE-HD), this works a bit differently:

```
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   rns:       { module_type: "RandomNumberSaver" }
 }

 analyzers:
 {
   pmtresponse: @local::dunefd_simphotoncounter
```

- simphotoncounter is a useful analyzer for legacy LArG4 that has similar outputs to our analyzer

- Particle interaction, ionization, electron drift, optical simulation all performed in one stage:
  - result is similar, but much less flexibility
- Configuration of light simulation (semi-analytic or library) is equivalent:

```
# Hits & Timing parameterization for DUNE FD, Ar scintillation
dunefd_pdfastsim_par_ar:                    @local::standard_pdfastsim_par_ar
dunefd_pdfastsim_par_ar.VUVTiming:          @local::dune_vuv_timing_parameterization
dunefd_pdfastsim_par_ar.VUVHits:            @local::dune_vuv_RS100cm_hits_parameterization

# As above, with cathode reflections included
dunefd_pdfastsim_par_ar_refl:               @local::dunefd_pdfastsim_par_ar
dunefd_pdfastsim_par_ar_refl.DoReflectedLight: true
dunefd_pdfastsim_par_ar_refl.VISTiming:     @local::dune_vis_timing_parameterization
dunefd_pdfastsim_par_ar_refl.VISHits:       @local::dune_vis_RS100cm_hits_parameterization
```