Simulating LEM gain in DUNE DP using legacy LArG4

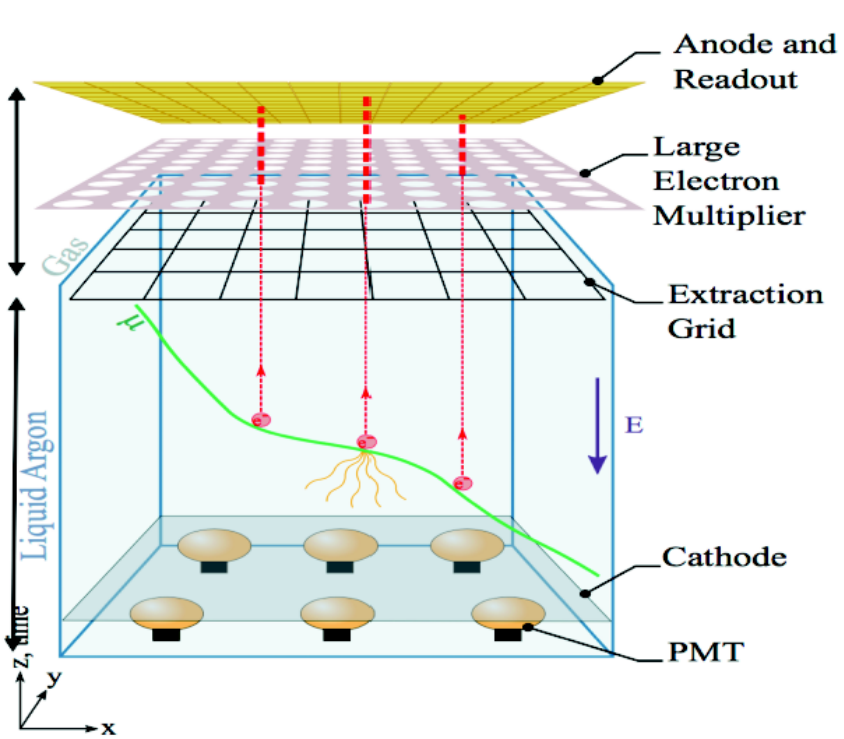
Jaime Dawson

29/06/2020

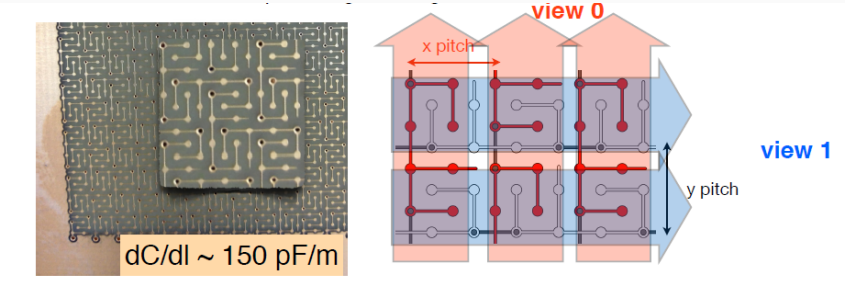
CNRS, laboratoire AstroParticule et Cosmologie, Paris

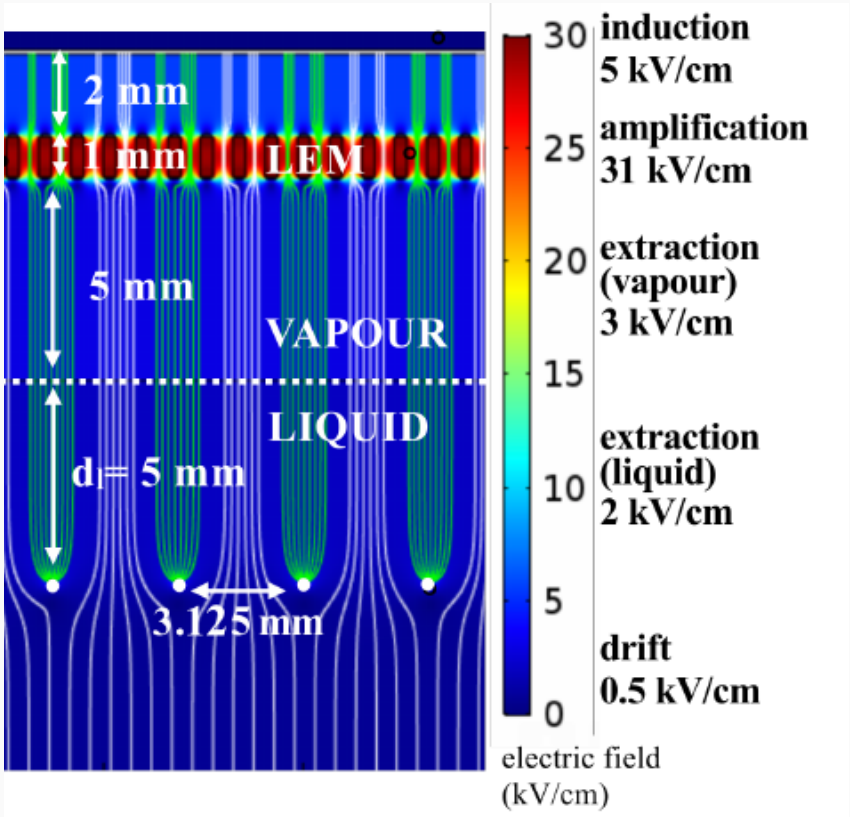
Table of contents

Dual Phase



Dual-Phase has additional steps to Single-Phase LArTPCs. Electrons are extracted into the gas. In the LEM holes, Townsend avalanches occur. Electrons are collected on two split anodes.





It would be great if we could simulate various operating voltages

Of interest:

Efficiency of electron extraction

Efficiency of entering LEM holes

LEM gain

Efficiency of anode collection

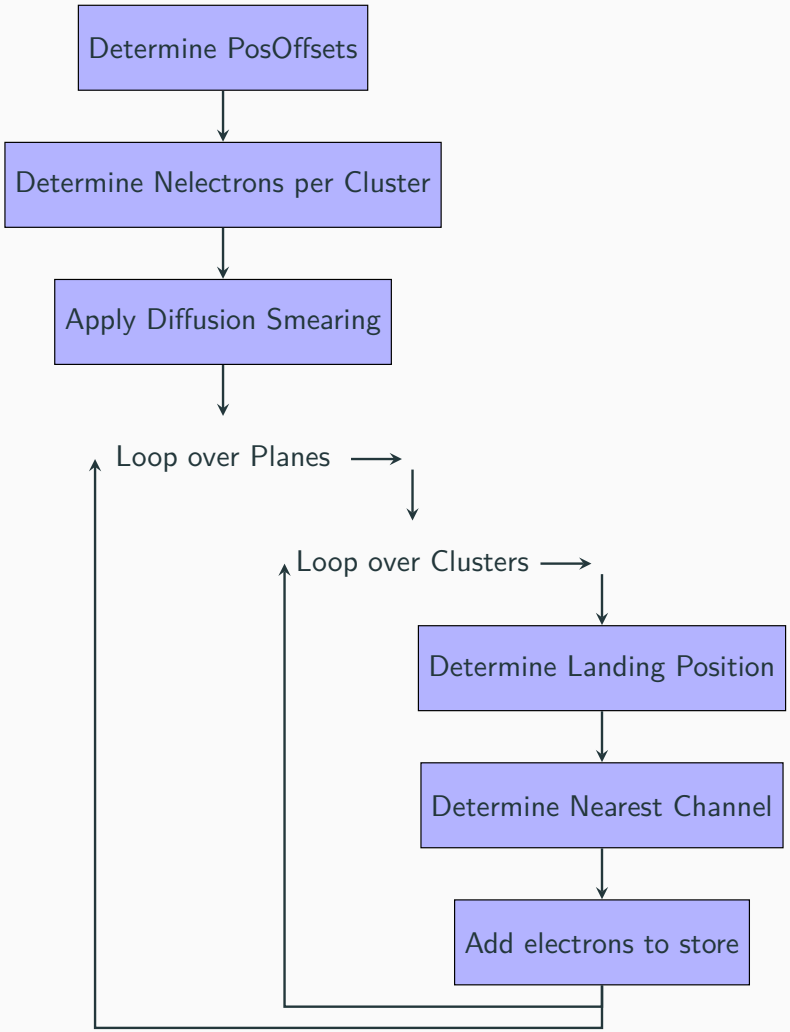
Noting:

The LEMs (10cm*10cm panels) are not all operating at the same voltages (different gains)

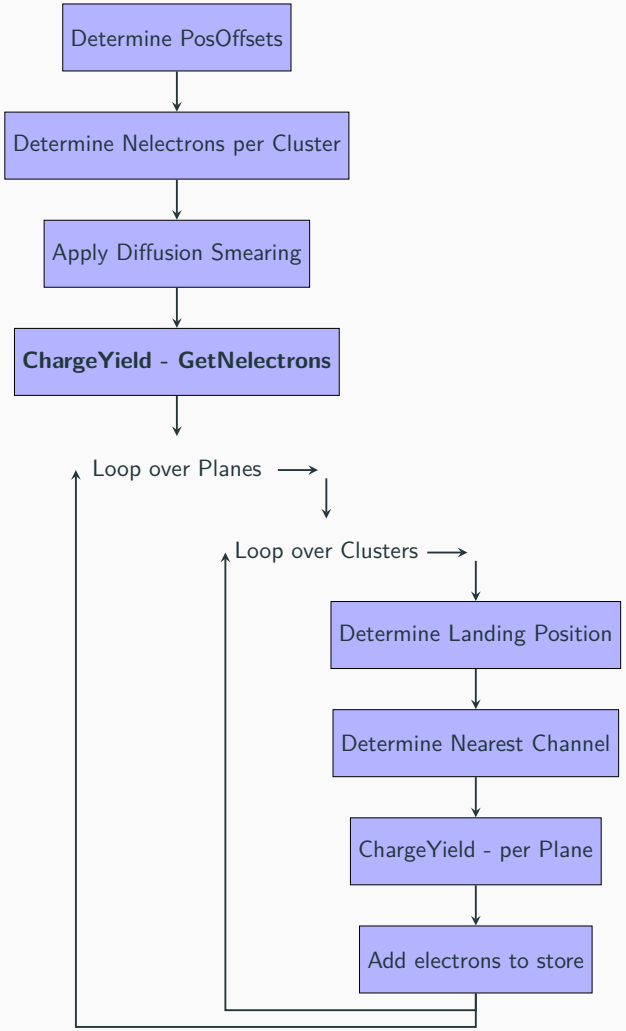
Data taken so far with protoDUNE-DP has a very non-uniform field due to a short of the field cage

LegacyLArG4

In void LArVoxelReadout::DriftIonizationElectrons(..).



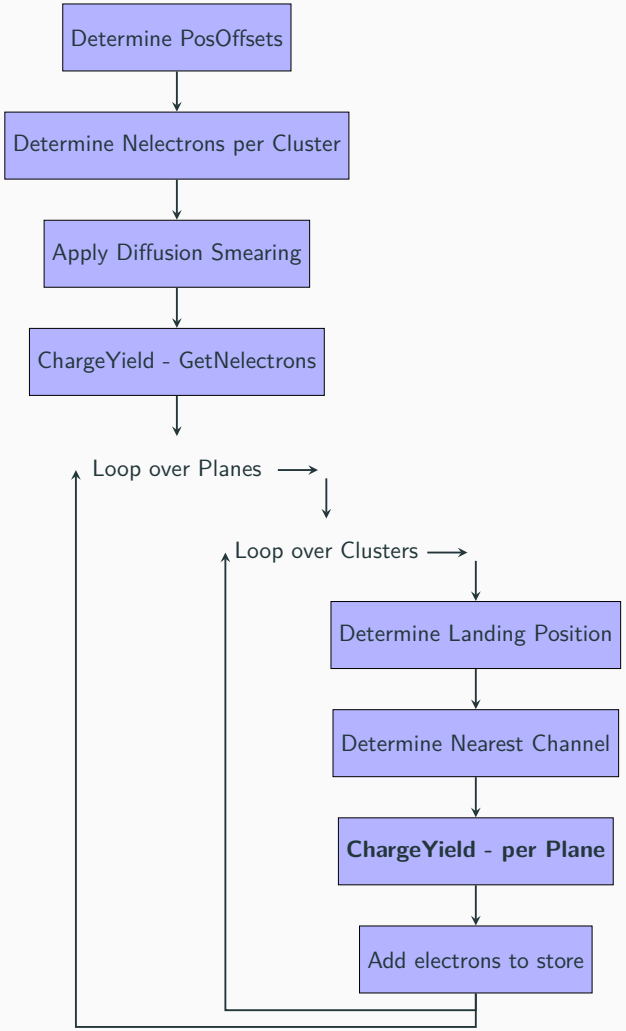
Landing positions are calculated and used to find the Nearest Channels. Electrons are collected and stored in SimChannels (per wire).
The SpaceCharge module is used to deal with our non-uniform Electric field. But, we lose the 2D information (landing position), and so we can not apply correctly any gain variations.
We also have difficulty with collection planes (correlation of number of electrons detected per cluster per plane).



Following structure of the SpaceCharge service.. Create a ChargeYieldService, that will do the amplification process

```
auto const* CYE = lar::providerFrom<chargeyield::
      ChargeYieldService>();
  if (CYE->EnableChargeYield() == true)
    {
      for(int k = 0; k < nClus; ++k){
        //calculate the number of electrons after
        //charge multiplication at first plane
        nElDiff[k]= CYE->GetNElectrons(cryostat,
              tpc,
        { (tpcg.PlaneLocation(0)[0] - tpcg.
              PlanePitch(0)), YDiff[k], ZDiff[k]
               },
        TDrift, nElDiff[k]);
      }
    }
```

and charge sharing before electrons are stored in SimChannels.



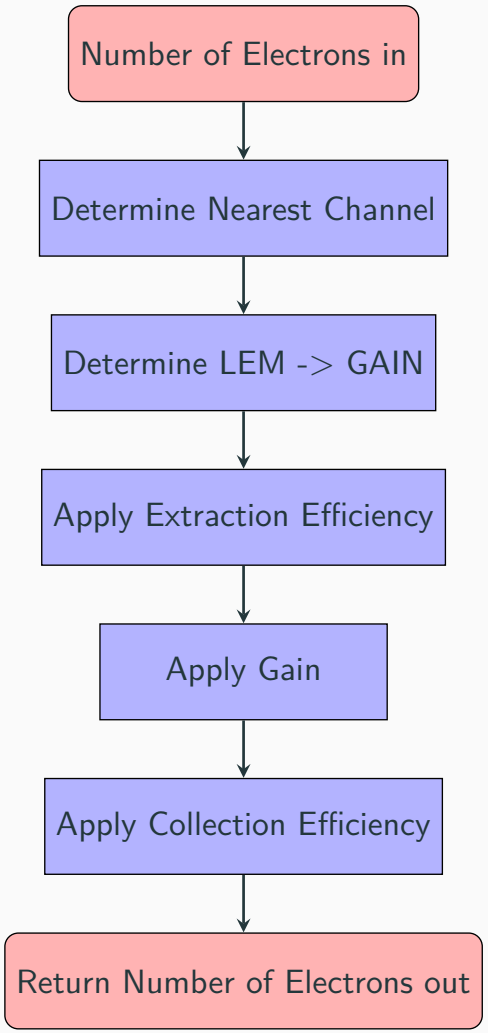

```
..
// Drift nClus electron clusters to the
     induction plane
   for(int k = 0; k < nClus; ++k){
..

     //if using ChargeYield module --> calculate
          electron signals per plane
     double nel=nElDiff[k];
     if (CYE->EnableChargeYield() == true)
         {//calculate how many electrons on each
              plane
           nel= CYE->GetNElectronsPlane(cryostat
                ,tpc, p, nElDiff[k]);
           //planes can be collection or
                induction
           nElDiff[k] = CYE->
                GetRemainingElectrons(nElDiff[
                k], nel);
           if(nel<=0) continue;
         }
       // Add electrons produced by each cluster
            to the map
       DepositsToStore[channel][tdc].add(nEnDiff
            [k], nel);
   }
   }
```

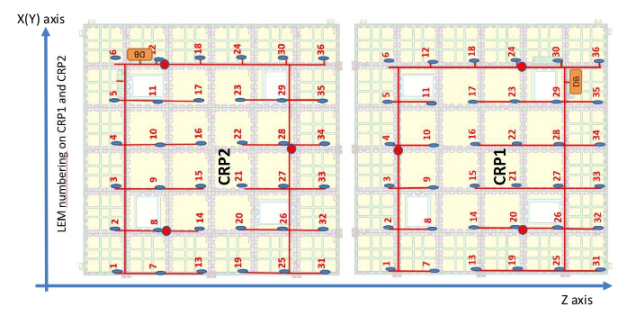
larevt/ChargeYieldStandard - does nothing. No change to normal working.
dunetpc/ChargeYieldProtoDUNEdp - implements:

- Configurable voltages (for all 144 LEMs and grids) or Effective Gain settings
- Statistics on LEM gain (if required)
- Configurable efficiencies on electron extraction and collection

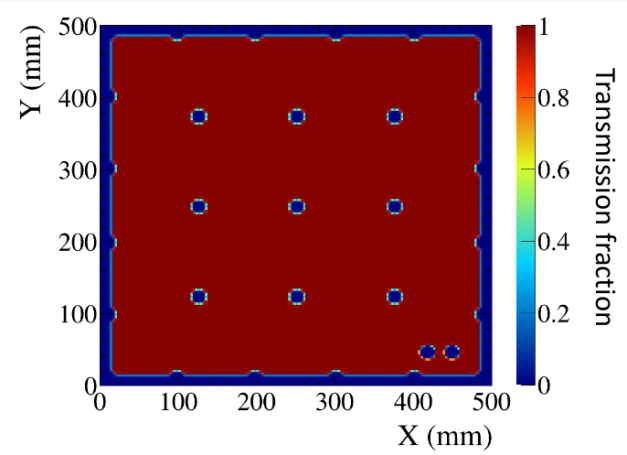
double chargeyield::ChargeYieldProtoDUNEdp::GetNElectrons(unsigned short int cryostat, unsigned short int tpc, geo::Point_t const& point, double tdrift, double Nelec_in)



LEM mapping



LEM dead areas



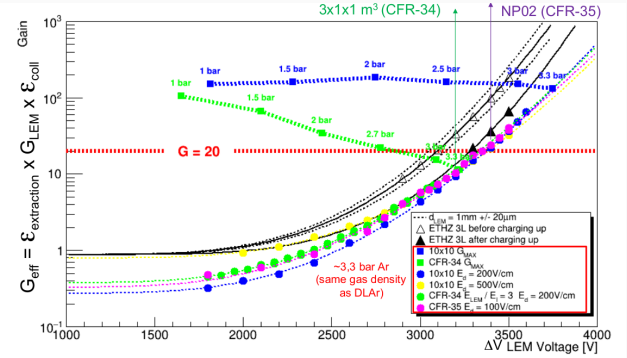
Two possibilities for the GAIN (fcl)

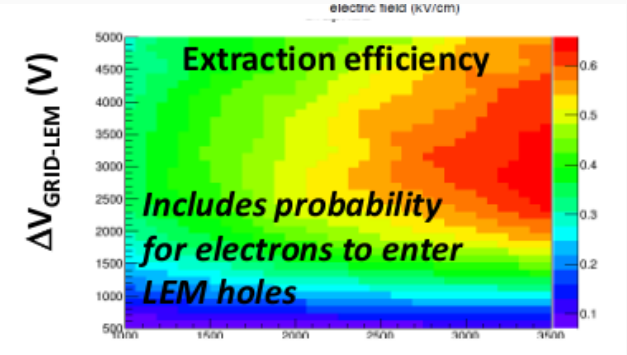
```
UseEffectiveGain: true
EffectiveGain: [0,0,0,1.22,0,0, ...144
      entries..]
```

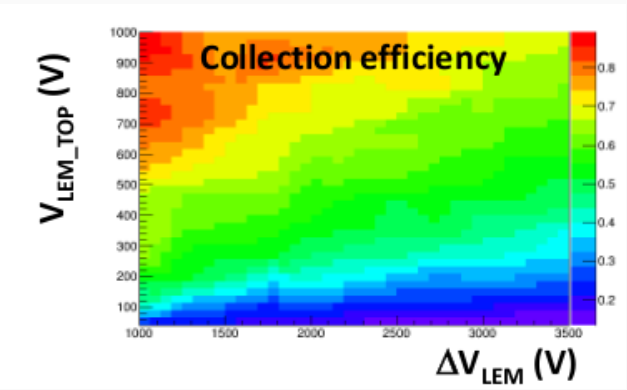
or via Voltages

```
CRPVoltFilename: "DB\_run1415.output.
      CRPVoltages" # file holding all
      voltages
defaultCollectionEfficiency: 0.7
defaultExtractionEfficiency: 1.0
CRPsimFilename: ""  # to come
Lem_A: 1646.99
Lem_B:  131.96
```

LEM lab gain measurements



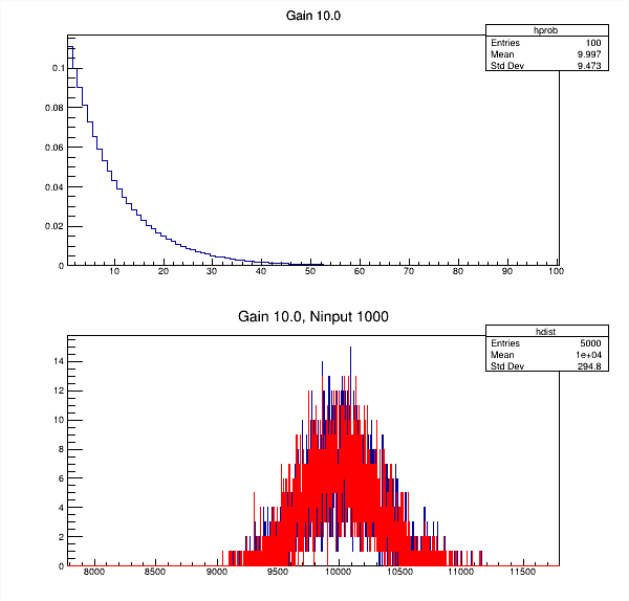




if

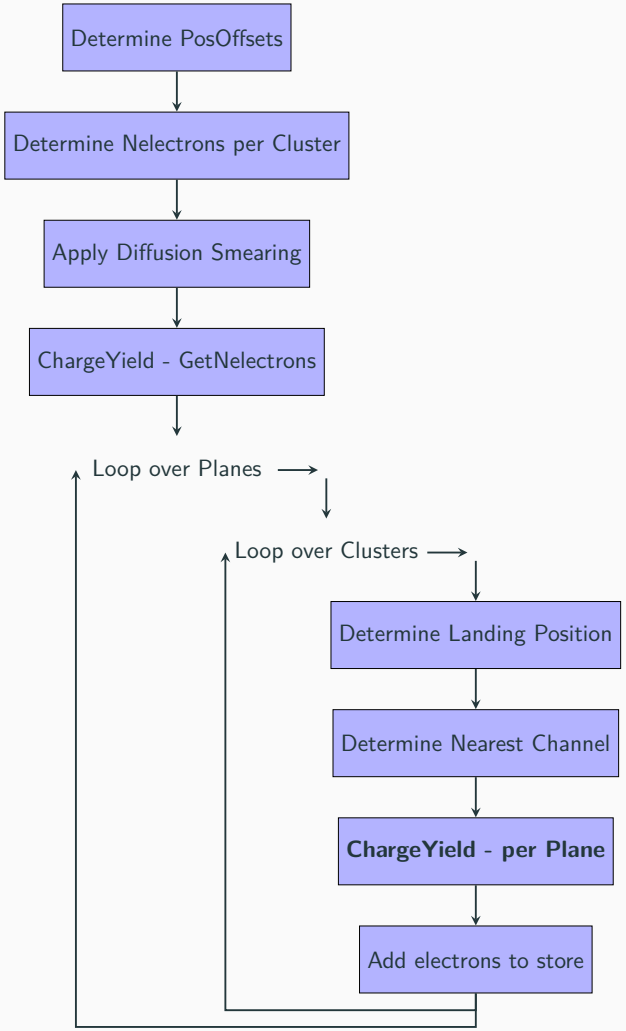
```
EnableRandom: true
```

then sample LEM gain fom Furry distribution.
If the number of electrons is low, distributions are non-Gaussian.
Example for a gain of 10 and 1000 input electrons.



Otherwise just use mean gain.

and charge sharing before electrons are stored in SimChannels.




```
..
// Drift nClus electron clusters to the
     induction plane
   for(int k = 0; k < nClus; ++k){
..

     //if using ChargeYield module --> calculate
          electron signals per plane
     double nel=nElDiff[k];
     if (CYE->EnableChargeYield() == true)
         {//calculate how many electrons on each
              plane
           nel= CYE->GetNElectronsPlane(cryostat
                ,tpc, p, nElDiff[k]);
           //planes can be collection or
                induction
           nElDiff[k] = CYE->
                GetRemainingElectrons(nElDiff[
                k], nel);
           if(nel<=0) continue;
         }
       // Add electrons produced by each cluster
            to the map
       DepositsToStore[channel][tdc].add(nEnDiff
            [k], nel);
   }
   }
```

```
..
// Drift nClus electron clusters to the
      induction plane
   for(int k = 0; k < nClus; ++k){
..

     //if using ChargeYield module --> calculate
           electron signals per plane
     double nel=nElDiff[k];
     if (CYE->EnableChargeYield() == true)
         {//calculate how many electrons on each
               plane
           nel= CYE->GetNElectronsPlane(cryostat
                 ,tpc, p, nElDiff[k]);
           //planes can be collection or
                 induction
           nElDiff[k] =  CYE->
                 GetRemainingElectrons(nElDiff[
                 k], nel);
           if(nel<=0) continue;
         }
       // Add electrons produced by each cluster
             to the map
       DepositsToStore[channel][tdc].add(nEnDiff
             [k], nel);
   }
   }
```

double GetNElectronsPlane(unsigned short int cryostat, unsigned short int tpc, int p, double Nelec$_i n$)*const*
and
double GetRemainingElectrons(double NeleCluster, double NelePlane) const
These two functions deal with the fact that our 2 anodes are collection planes, without it being assumed in LArVoxelReadOut.

Refactored for Dual Phase

- Very recently..In use for *Light* simulation (with alternative geometry - ydrift)
- Not yet in use for *Charge* simulation
- We will move towards refactored so OK to implement there
- What do I need?