Algorithms for

calculating number of ions and photons

MU Wei

Jan. 2021

Existing Algorithms in LArSoft (larsim)

Two LArG4
- Refactored
- Legacy

Three algorithms
- Separate
- NEST
- Correlated

Refactored

Legacy

File Name (directory order)	Directory
ISCalc.cxx	larsim\larsim\IonizationScintillation
ISCalc.h	larsim\larsim\IonizationScintillation
ISCalcCorrelated.cxx	larsim\larsim\IonizationScintillation
ISCalcCorrelated.h	larsim\larsim\IonizationScintillation
ISCalcNESTLAr.cxx	larsim\larsim\IonizationScintillation
ISCalcNESTLAr.h	larsim\larsim\IonizationScintillation
ISCalcSeparate.cxx	larsim\larsim\IonizationScintillation
ISCalcSeparate.h	larsim\larsim\IonizationScintillation
ISCalculation.cxx	larsim\larsim\LegacyLArG4
ISCalculation.h	larsim\larsim\LegacyLArG4
ISCalculationCorrelated.cxx	larsim\larsim\LegacyLArG4
ISCalculationCorrelated.h	larsim\larsim\LegacyLArG4
ISCalculationNEST.cxx	larsim\larsim\LegacyLArG4
ISCalculationNEST.h	larsim\larsim\LegacyLArG4
ISCalculationSeparate.cc	larsim\larsim\ElectronDrift
ISCalculationSeparate.cxx	larsim\larsim\LegacyLArG4
ISCalculationSeparate.h	larsim\larsim\ElectronDrift
ISCalculationSeparate.h	larsim\larsim\LegacyLArG4

Code for the three algorithms

The Algorithms in Photon Fast Simulation

Legacy largeant

- **LArG4** module (photon fast simulation)
  - Calculate number of photons by the **IonizationAndScintillation** instance
    - Choose algorithm via **ISCalculation + LArG4Parameters**
  - Photon fast simulation by the **OpFastScintillation** instance
    - Generate PD response according to PVS: Library/Parameterization

Refactored largeant

- **larg4Main** module
  - Simulate propagation of the primary particle
- **IonAndScint** module
  - Calculate number of ions and photons at each energy deposit
  - Choose algorithm via **ISCalc + InputTag**
- **PDFastSim** modules
  - Generate PD response according to Library/Parameterization/GAN

Separate Algorithm

- Calculate number of ions based on:
  - Recombination models, and
  - Ionization yield from **LArG4Parameters**
- Calculate number of photons based on:
  - Particle type
  - Scint yield from **LArProperties**

```
fGeVToElectrons    = LArG4PropHandle->GeVToElectrons();
if(fUseModBoxRecomb)
{
    if(ds>0)
    {
        double Xi = fModBoxB * dEdx / EFieldStep;
        recomb    = log(fModBoxA + Xi) / Xi;
    }
    else
    {
        recomb = 0;
    }
}
else
{
    recomb = fRecombA / (1. + dEdx * fRecombk / EFieldStep);
}

// 1.e-3 converts fEnergyDeposit to GeV
fNumIonElectrons = fGeVToElectrons * 1.e-3 * e * recomb;
```

```
if(fLArProp->ScintByParticleType())
{
    switch(pdg)
    {
        case 2212:
            scintYield = fLArProp->ProtonScintYield(true);
            break;
        case  13:
        case -13:
            scintYield = fLArProp->MuonScintYield(true);
            break;
        case  211:
        case -211:
            scintYield = fLArProp->PionScintYield(true);
            break;
        case  321:
        case -321:
            scintYield = fLArProp->KaonScintYield(true);
            break;
        case 1000020040:
            scintYield = fLArProp->AlphaScintYield(true);
            break;
        case  11:
        case -11:
        case  22:
            scintYield = fLArProp->ElectronScintYield(true);
            break;
        default:
            scintYield = fLArProp->ElectronScintYield(true);
    } « end switch pdg »

    fNumScintPhotons = scintYield * e;
} « end if fLArProp->ScintByPart... »
```

Correlated Algorithm

- Newly developed – after the refactored largeant
- Algorithm is identical in legacy or refactored version
- Consider the anti-correlation between ion and photons due to the ion-photon recombination effect:
  - Box recombination model, or
  - A “Saturation” model

```
fGeVToElectrons    = LArG4PropHandle->GeVToElectrons();

// ionization work function
fWion              = 1./fGeVToElectrons * 1e3; // MeV

// ion+excitation work function (\todo: get from LArG4
fWph               = 19.5 * 1e-6; // MeV

// calculate total quanta (ions + excitons)
double Nq = fEnergyDeposit / fWph;
```

No Problems

```
// Guard against spurious values of dE/dx. Note: assumes density of LAr
if(dEdx < 1.) dEdx = 1.;

// calculate recombination survival fraction
if(fUseModBoxRecomb)
{
  if(ds>0)
  {
    double Xi = fModBoxB * dEdx / EFieldStep;
    recomb    = log(fModBoxA + Xi) / Xi;
  }
  else
  {
    recomb = 0;
  }
}
else
{
    recomb = fRecombA / (1. + dEdx * fRecombk / EFieldStep);
}
```

```
// using this recombination, calculate number of ionization electrons
fNumIonElectrons = ( fEnergyDeposit / fWion ) * recomb;

// calculate scintillation photons
fNumScintPhotons = Nq - fNumIonElectrons;
```

NEST Algorithm

Legacy largeant - **NestAlg**

- Implementation of NEST model in LArSoft
- Consider the anti-correlation of ions and photons
  - Box model/**Birks law**/…
- Support **neon/argon/krypton/xenon** media
- Support **gaseous/liquid/solid** phases
- Need additional parameters (beyond that in **EnergyDeposit**)

Refactored largeant - **NESTLAr**

- Simplification from NestAlg
- Also consider the anti-correlation of ions and photons
- Only use parameters recorded in **EnergyDeposit**
- Only support **liquid argon**

NEST Algorithm – extracted from NESTLAr

Number of total quanta: e^- and ph

```
fScintYield        = 1.0 / (19.5*CLHEP::eV);

double MeanNumQuanta = fScintYield * fEnergyDeposit;
double sigma         = sqrt(fResolutionScale * MeanNumQuanta); //Fano
int NumQuanta        = int(floor(GaussGen.fire(MeanNumQuanta,sigma)+0.5));
```

Number of initial e^- and ph

```
int NumExcitons = BinomFluct(NumQuanta, fExcitationRatio/(1 + fExcitationRatio));
int NumIons     = NumQuanta - NumExcitons;
```

Birks law for e^--ph recombination

```
DokeBirks[1] = DokeBirks[0]/(1-DokeBirks[2]);                          //B=A/(1-C) (see paper)
recombProb   = (DokeBirks[0]*LET)/(1+DokeBirks[1]*LET)+DokeBirks[2]; //Doke/Birks' Law as spe
```

Anti-correlation between e^- and ph

```
//use binomial distribution to assign photons, electrons, where photons
//are excitons plus recombined ionization electrons, while final
//collected electrons are the "escape" (non-recombined) electrons
int NumPhotons   = NumExcitons + BinomFluct(NumIons,recombProb);
int NumElectrons = NumQuanta - NumPhotons;
```

Working Identically for LAr

Separate Algorithm – issue

When using **Separate** algorithm to calculate the number of **photons** for refactored largeant:

The number of photons in the refactored largeant is **~15% more** than that produced in the legacy largeant

Configurations:

- protodunev7_photonvisibilityservice
- protodune_v7_refactored_nowires.gdml

Separate Algorithm – issue

- Legacy largeant considers EM saturation effect
  - Uses Birks law (G4EmSaturation) to calculate the **visible energy** deposit at a step
  - Calculates the number of photons at a step based on a universal scintYield
  - Visible energy is (~15% to 20%) less than the total energy
- Algorithm in Refactored largeant is ported from larsim/ElectronDrift/ISCalculationSeparate.cc instead of larsim/LegacyLArG4/ISCalculationSeparate.cxx

Shall we port G4EmSaturation to refactored largeant?

```
// Use Birks Correction in the Scintillation process
fEMSaturation = G4LossTableManager::Instance()->EmSaturation();

// if not doing the scintillation by particle type, use the saturation
double scintYield = mpt->GetConstProperty("SCINTILLATIONYIELD");
```

```
} « end if fScintByParticleType »
else if(fEMSaturation){
  // The default linear scintillation process
  fVisibleEnergyDeposition = fEMSaturation->VisibleEnergyDepositionAtAStep(step);
  fNumScintPhotons = fScintYieldFactor * scintYield * fVisibleEnergyDeposition;
}
```