Update on the integration of the SBND light simulation into LArSoft

Diego Garcia-Gamez
The University of Manchester

Introduction

- The presence of TPB-coated reflector foils in the SBND geometry introduces an extra light component (reflected/visible)
- For the (fast) simulation of this "new" component we need to save its visibility at each position in the detector, as we do for the direct/VUV light
- To account for the propagation time of the scintillation light, neglected so far, we have modeled their arrival time distributions and parametrized the model parameters for both components:

  - Direct/VUV component is universal for all experiments (at first order)
  - Reflected/visible component depends on the particular detector geometry and LDS configuration

- **We have modified LArSoft to get the previous points (details on how in next slides)**

Modifications to LArSoft out of sbndcode

larsim:
PhotonPropagation/
PhotonLibrary.{h,cxx}, PhotonVisibilityService.h, PhotonVisibilityService_service.cc, photpropservices.fcl
LArG4/
MaterialPropertyLoader.{h,cxx}, OpFastScintillation.{hh,cxx}

lardataobj:
Simulation/
SimPhotons.h

larana:
OpticalDetector/
SimPhotonCounter_module.cc

lardata:
DetectorInfo/Utilities/
LArProperties.h, LArPropertiesStandard.{h,cxx}, larproperties.fcl

Configuration of time parametrizations

- I need the parametrizations in OpFastScintillation.cxx
- We add all the configurable parameters as a PhotonVisibilityService (in photpropservices.fcl)
- For the case of SBND (i.e. in sbndcode) photpropservices_sbnd.fcl → sbnd_photonvisibilityservice: @local::sbnd_timeparametrization_photonvisibilityservice instead of @local::standard_photonvisibilityservice

sbnd_timeparametrization_photonvisibilityservice:

{ (Standard definitions plus all below)

---------- Direct/VUV component modeled with a Landau + Expontial function --------
The 5 parameters are parametrized as a function of the distance

Direct_landauNormpars: [7.85903, -0.108075, 0.00110999, -6 ...
6.20863e-14, -2.97559e-17]
Direct_landauMPVpars: [1.20259, 0.0582674, 0.000308053, - ...
Direct_landauWidthpars: [0.346667, -0.00768231, 0.0002118 ...
Direct_expoCtepars: [13.6592, -0.188798, 0.00192431, -1.106 ...
3.17657e-14]
Direct_expoSlopepars: [-0.57011, 0.0156393, -0.000197461, 1.34491e-06, -5.24544e-09, 1.1703e-11,
-1.38811e-14, 6.78368e-18]

Parameters of the direct light model: landau + exponential function

At long distances we extrapolate the behaviour of the parameters

Direct_landauNormpars_far: [2.23151, -0.00627503]
Direct_landauMPVpars_far: [-3.04952, 0.128638]
Direct_expoCtepars_far: [3.69578, -0.00989582]

Extrapolation used at long distances

Direct_functions: ["pol7", "pol4", "pol3", "pol6", "pol7", "expo", "pol1", "expo"]

Functions used for the different parameters

range of distances where the parametrization is valid [~10 - 500cm], then:

Continue next slide ...

```
D_break: 500.
# farther are extrapolations
D_max: 750.
# increase by this factor the number of points used to sample the function
# improve the accuracy when the scintillation happens very close to the PMT
# where the signal shape (function) is VERY sharp. BUT SLOW DOWN THE SIMULATION!

  TF1_sampling_factor: 1

# ----------  Direct/VUV component modeled with a Landau + Expontial function -------- #
# The 5 parameters are parametrized as a function of the distance

  Reflected_landauNormpars: [7.54642, -0.441946, 0.0107579, -9.53399e-05]
  Reflected_landauMPVpars: [-1.61482, 1.18624, 0.00105223, -9.52016e-05]
  Reflected_landauWidthpars: [0.440124, -0.0557912, 0.00544957, -9.39128e-05]
  Reflected_expoCtepars: [14.6874, -0.896761, 0.0214977, -0.000185728]
  Reflected_expoSlopepars: [-0.650584, 0.0800897, -0.00379933, 7.91909e-05, -6.1

# range of t0s where the parametrization is valid [~8 - 55ns], then:
  T0_max: 55.

  Reflected_functions: ["pol3", "pol3", "pol3", "pol3", "pol4"]

# ns after the parametrization must be corrected (lack of statistics!)
  T0_break_point: 42.
}
```

We have points up to 500 cm in our simulations. We extrapolate up to 750cm

Sampling factor. Only relevant very close to the photon detectors: O(10's of cm)

Similar for the reflected light

Test branch with LArSoft v06_30_00

- All these modifications can be found and tested in a feature branch called **dgg_lightprop** *(you need to check out larana, lardata, lardataobj, larsim and sbndcode)*

- First tests: for the case of SBND, in sbndcode/JobConfigurations you can find two new fcl files **prodsingle_opfast_proptime.fcl** and **sbnd_buildopticallibrary_withrefl.fcl**.

sbnd_buildopticallibrary_dgg.fcl, prodsingle_opfast_dgg.fcl:

```
# enable extra materialproperties (including tpb WLS)
services.LArPropertiesService.LoadExtraMatProperties: true
services.LArPropertiesService.SimpleBoundaryProcess: false
services.LArPropertiesService.SimpleScintillation: false
```

← Loading TPB properties

```
services.PhotonVisibilityService.StoreReflected: true
services.PhotonVisibilityService.StoreReflT0: true
services.PhotonVisibilityService.IncludePropTime: true
services.PhotonVisibilityService.TF1_sampling_factor: 1.

services.LArPropertiesService.ScintByParticleType:  true
```

Save and use (in case saved) the visibilities for the reflected component (StoreReflected, StoreReflT0)

← Account for propagation time (IncludePropTime)

```
physics.analyzers.pmtresponse.MakeAllPhotonsTree: false
physics.analyzers.pmtresponse.MakeDetectedPhotonsTree: false
physics.analyzers.pmtresponse.MakeOpDetsTree: false
physics.analyzers.pmtresponse.MakeOpDetEventsTree: false
physics.analyzers.pmtresponse.MakeLightAnalysisTree: true
```

← Save a new analysis tree with the information about the light

sbnd_buildopticallibrary_dgg.fcl, prodsingle_opfast_dgg.fcl:

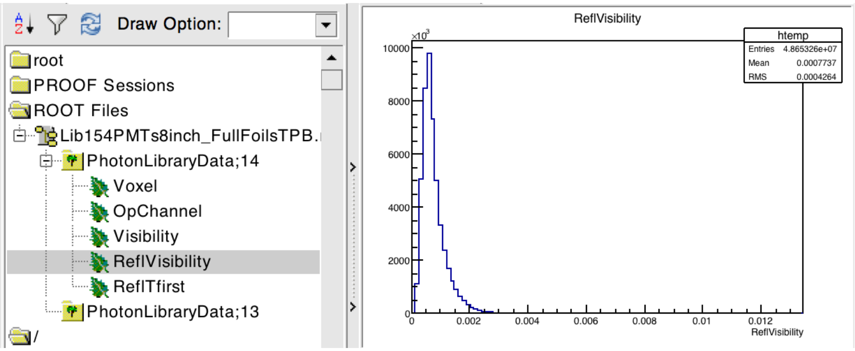
```
//generating the tree for the light analysis:
if(fMakeLightAnalysisTree)
  {
    fLightAnalysisTree = tfs->make<TTree>("LightAnalysis","LightAnalysis");
    fLightAnalysisTree->Branch("RunNumber",&fRun);
    fLightAnalysisTree->Branch("EventID",&fEventID);
    fLightAnalysisTree->Branch("TrackID",&fTrackID);
    fLightAnalysisTree->Branch("PdgCode",&fpdg);
    fLightAnalysisTree->Branch("MotherTrackID",&fmotherTrackID);
    fLightAnalysisTree->Branch("Energy",&fEnergy);
    fLightAnalysisTree->Branch("dEdx",&fdEdx);
    fLightAnalysisTree->Branch("StepPrePositions",&fstepPrePositions);
    fLightAnalysisTree->Branch("StepPostPositions",&fstepPostPositions);
    fLightAnalysisTree->Branch("StepPreTimes",&fstepPreTimes);
    fLightAnalysisTree->Branch("StepPostTimes",&fstepPostTimes);
    fLightAnalysisTree->Branch("SignalsVUV",&fSignalsvuv);
    fLightAnalysisTree->Branch("SignalsVisible",&fSignalsvis);
    fLightAnalysisTree->Branch("Process",&fProcess);
  }
```

```
# enable ex
services.LA
services.LA
services.LA

services.Ph
services.Ph
services.Ph
services.Ph

services.LA

physics.ana
physics.ana
physics.analyzers.pmtresponse.MakeOpDetsTree: false
physics.analyzers.pmtresponse.MakeOpDetEventsTree: false
physics.analyzers.pmtresponse.MakeLightAnalysisTree: true
```

Save a new analysis tree with the information about the light

First tests
(only with SBND geometry so far)

1. Optical library generated **with and without** the new branches for the reflected light
   → **It seems to work**



2. Fast optical simulation **with and without** accounting for the propagation time and **with and without** the reflected light
   → **It seems to work,** I mean, it runs and produces the output analysis tree, **BUT** I found a bug: *ScintByPartycleType* option does not work properly, I get always (for any particle) a constant scintillation yield of 600 (surprisingly small!)

Not sure if bug related with my changes. I don't see why, but still debugging (using NEST?)

MaterialPropertyLoader

```
//Loop through geometry elements and apply relevant material table where materials match
for ( G4LogicalVolumeStore::iterator i = lvs->begin(); i != lvs->end(); ++i ){
  G4LogicalVolume* volume = (*i);
  G4Material* TheMaterial = volume->GetMaterial();
  std::string Material = TheMaterial->GetName();

  G4MaterialPropertyVector* PropertyPointer = 0;
  if(MaterialTables[Material])
    PropertyPointer = MaterialTables[Material]->GetProperty("REFLECTIVITY");

  if(Material=="Copper"){
    std::cout<< "copper foil surface set "<<volume->GetName()<<std::endl;
    if(PropertyPointer) {
      std::cout<< "defining Copper optical boundary "<<std::endl;
      G4OpticalSurface* refl_opsurfc = new G4OpticalSurface("Surface copper",glisur,ground,dielectric_metal);
      refl_opsurfc->SetMaterialPropertiesTable(MaterialTables[Material]);
      refl_opsurfc->SetPolish(0.2);
      new G4LogicalSkinSurface("refl_surfacec",volume, refl_opsurfc);
    }
    else
      std::cout<< "Warning: Copper surface in the geometry without REFLECTIVITY assigned"<<std::endl;
  }
```

In the generation of the optical libraries we don't use the "Simple Boundary Model" in the tracking of the photons: f% diffusion + (1-f)% specular reflection. Instead we use more advanced models available in Geant4

Continue in next slide …

MaterialPropertyLoader

```
if(Material=="G10"){
  std::cout<< "G10 surface set "<<volume->GetName()<<std::endl;
  if(PropertyPointer) {
    std::cout<< "defining G10 optical boundary "<<std::endl;
    G4OpticalSurface* refl_opsurfg = new G4OpticalSurface("g10 Surface",glisur,ground,dielectric_metal);
    refl_opsurfg->SetMaterialPropertiesTable(MaterialTables[Material]);
    refl_opsurfg->SetPolish(0.1);
    new G4LogicalSkinSurface("refl_surfaceg",volume, refl_opsurfg);
  }
  else
    std::cout<< "Warning: G10 surface in the geometry without REFLECTIVITY assigned"<<std::endl;
}

if(Material=="vm2000"){
  std::cout<< "vm2000 surface set "<<volume->GetName()<<std::endl;
  if(PropertyPointer) {
    std::cout<< "defining vm2000 optical boundary "<<std::endl;
    G4OpticalSurface* refl_opsurf = new G4OpticalSurface("Reflector Surface",unified,groundfrontpainted,dielectric_dielectric);
    refl_opsurf->SetMaterialPropertiesTable(MaterialTables[Material]);
    G4double sigma_alpha = 0.8;
    refl_opsurf->SetSigmaAlpha(sigma_alpha);
    new G4LogicalSkinSurface("refl_surface",volume, refl_opsurf);
  }
  else
    std::cout<< "Warning: vm2000 surface in the geometry without REFLECTIVITY assigned"<<std::endl;
}
if(Material=="STEEL_STAINLESS_Fe7Cr2Ni"){
  std::cout<< "STEEL_STAINLESS_Fe7Cr2Ni surface set "<<volume->GetName()<<std::endl;
  if(PropertyPointer) {
    std::cout<< "defining STEEL_STAINLESS_Fe7Cr2Ni optical boundary "<<std::endl;
    G4OpticalSurface* refl_opsurfs = new G4OpticalSurface("Surface Steel",glisur,ground,dielectric_metal);
    refl_opsurfs->SetMaterialPropertiesTable(MaterialTables[Material]);
    refl_opsurfs->SetPolish(0.5);
    new G4LogicalSkinSurface("refl_surfaces",volume, refl_opsurfs);
  }
  else
    std::cout<< "Warning: STEEL_STAINLESS_Fe7Cr2Ni surface in the geometry without REFLECTIVITY assigned"<<std::endl;
}
```

Summary

- We are introducing:

i. more advanced reflective properties
ii. a second component of light to the optical library
iii. the ability to parametrize arrival times due to direct transport and Rayleigh scattering

- All above options are introduced as optional → can be turned off via .fcl parameter

- Code checked out into an available branch **`feature/dgg_lightprop`**

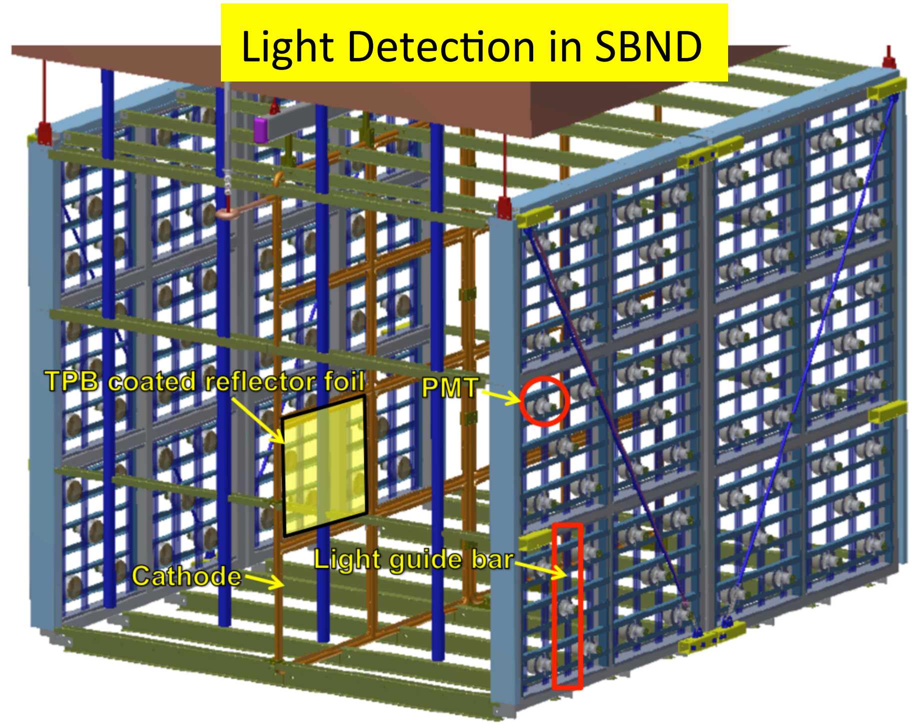
Back-Up

Motivation

- SBND is implementing a high LY Light Detection System scheme
- PMTs + Bars as detectors
- Possibility of adding WLS covered reflector foils

- We have developed a detailed MC simulation to determine the capabilities of the system and the effect of adding foils using the LArSoft framework

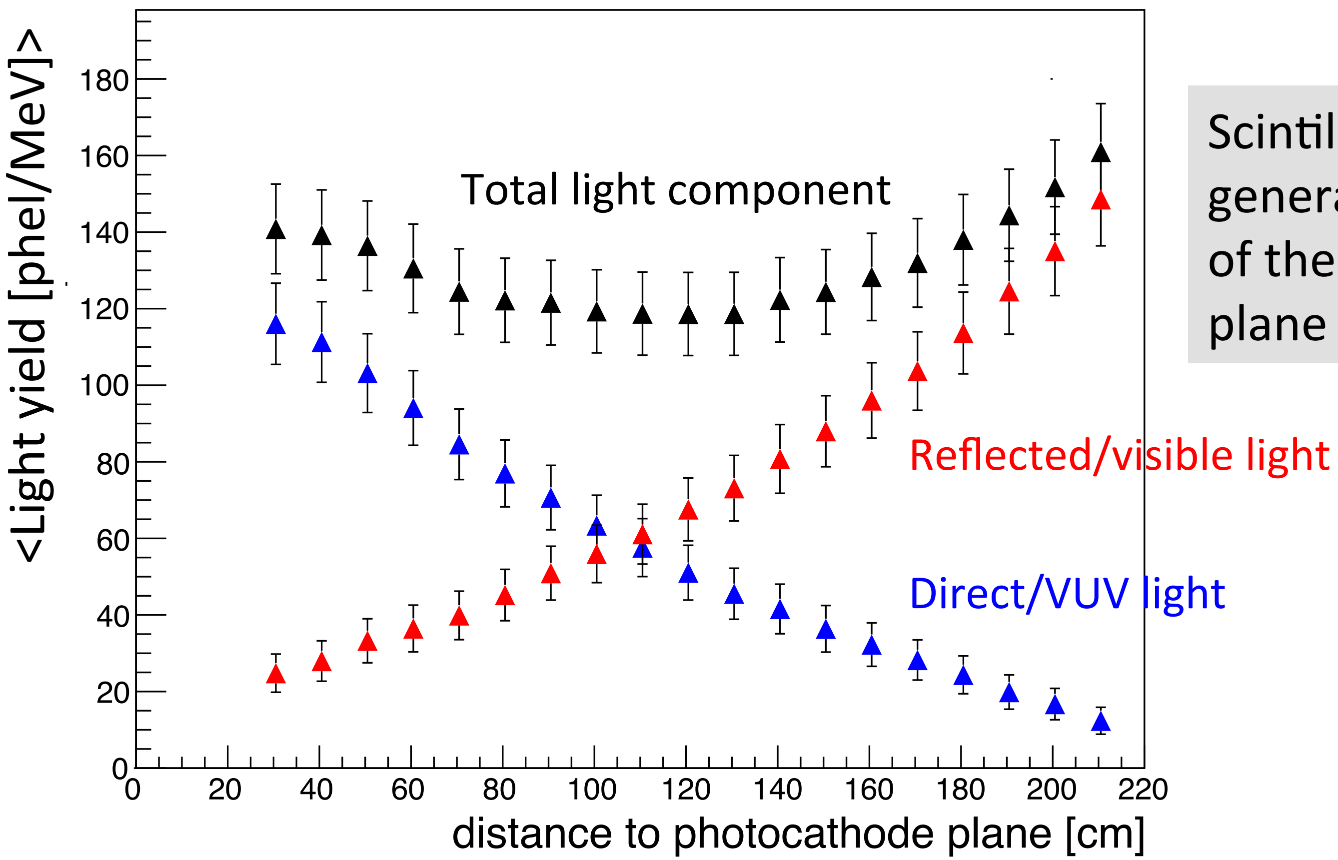
- For those studies some modifications/additions were needed in LArSoft not only in sbndcode (next slides)



- Parts of the simulation were started by Pawel Kryczynski for LArIAT

Light Yield

LDS = array of 60 8” PMTs + TPB-coated reflector foils covering the cathode

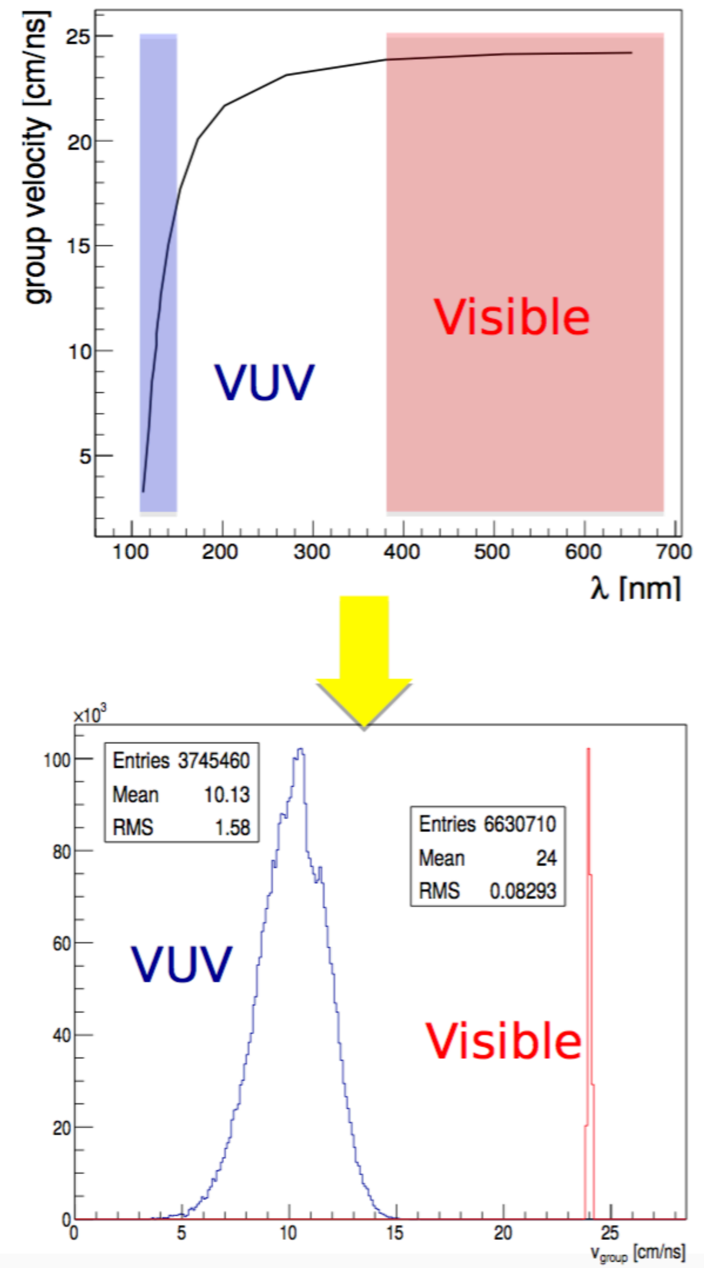


Scintillation points generated in the center of the photocathode plane

Average number of photoelectrons/event/MeV (adding the signal in all the PMTs) vs X position (drift distance to the photocathode plane)

Time

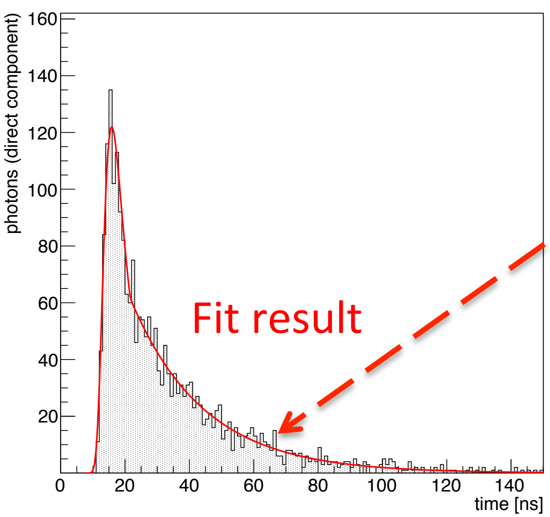
- To see if ~ns resolutions are possible needed to account for second order effects, like Rayleigh scattering ~55cm f(λ)
- Note high refractive index ~1.5 and gradient for VUV → relatively slow light
- Impossible to reproduce using a lookup library (memory) -> parametrization of arrival times



Arrival time distributions

- In SBND we have included the propagation time in the fast optical mode.
- And we have validated it (the direct component) also using the MicroBooNE geometry (next two slides)

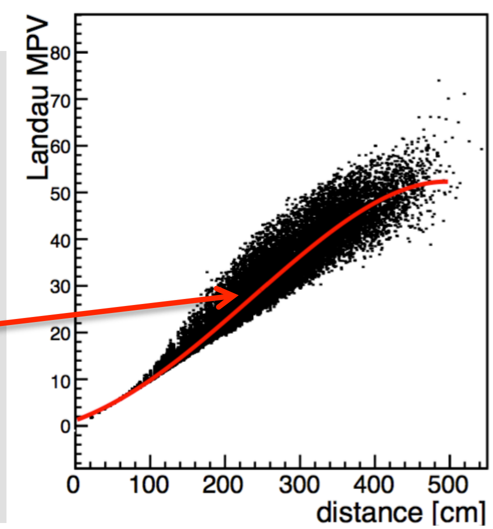
We have parameterized the time distributions → resulted only from direct transport + Rayleigh scattering



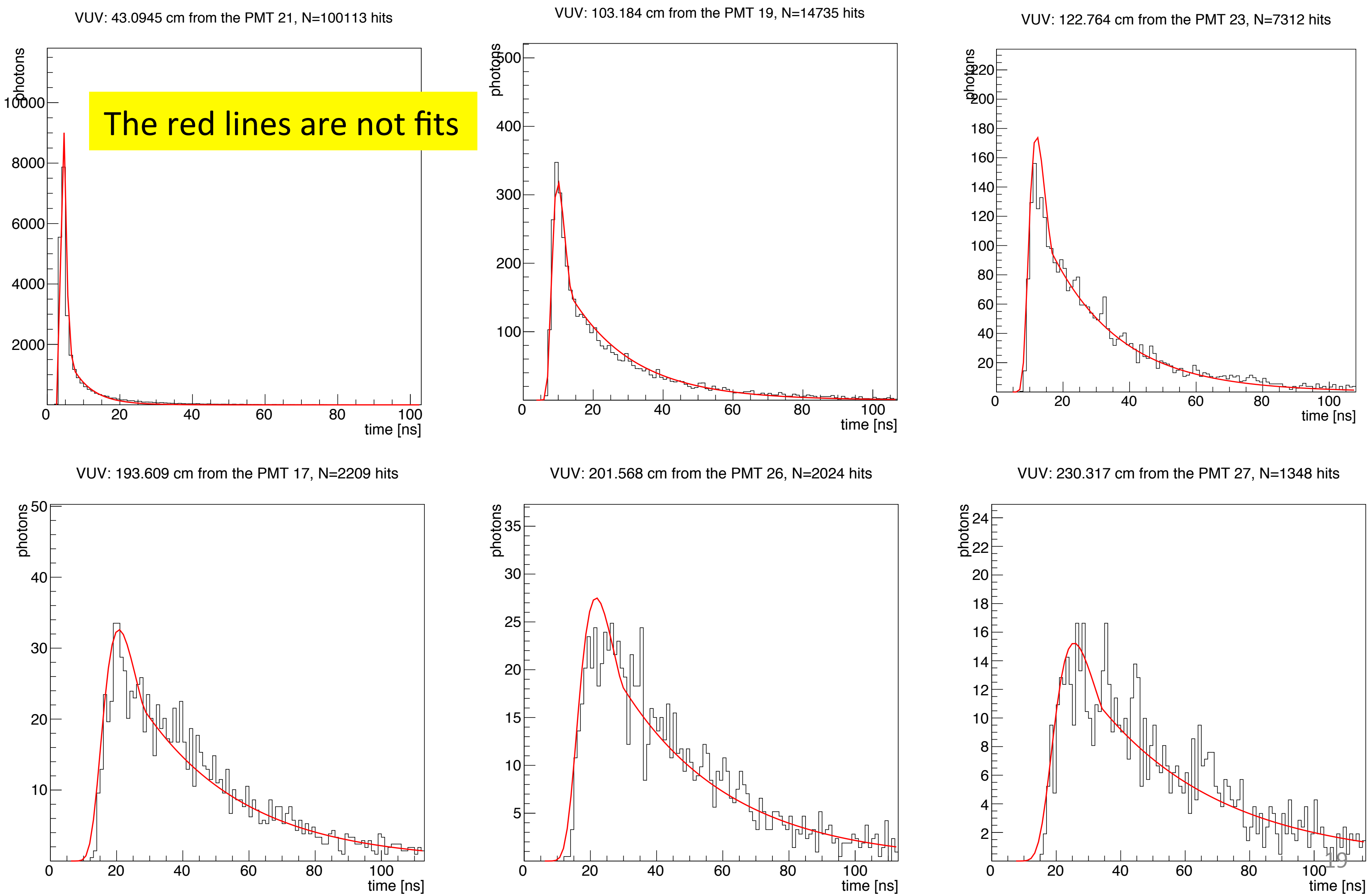
A **Landau + Exponential** function describes well the arrival time distributions of the direct/VUV light at any distance from the photocathode

Parameterization ready in LArSoft (next weeks):

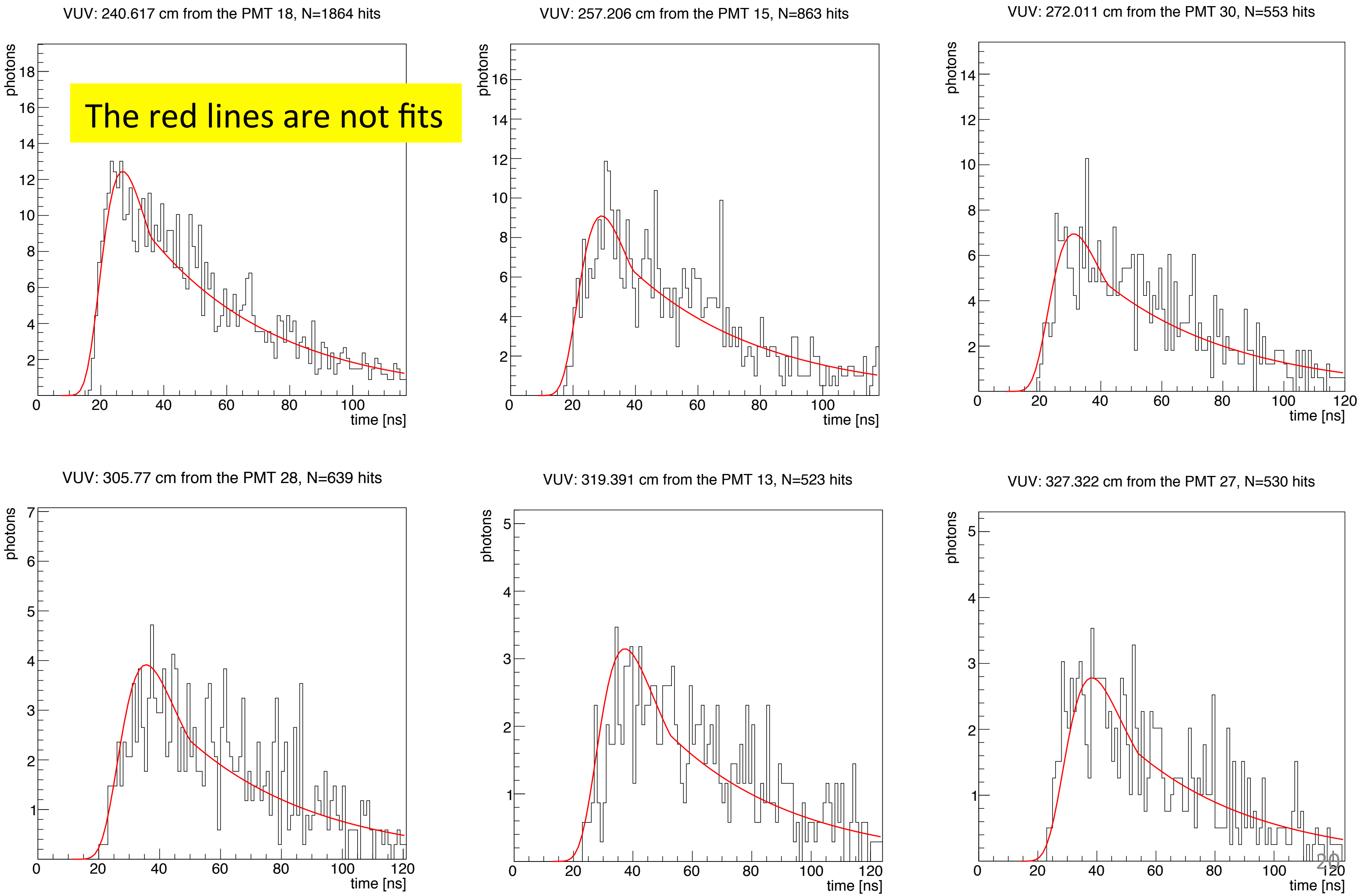
```
par0 = Landau normalization
par1 = Landau MPV
par2 = Landau width
par3 = Expo cte
par4 = Expo tau
```



Validating direct light time parameterization (with uBooNE geometry)



Validating direct light time parameterization (with uBooNE geometry)



Modifications to LArSoft out of sbndcode

lardataobj: Simulation/
SimPhotons.h

```
class OnePhoton
{
public:
  OnePhoton();

  bool          SetInSD;
  TVector3      InitialPosition;
  TVector3      FinalLocalPosition; // in cm
  float         Time;
  float         Energy;
  int           MotherTrackID;
};
```

larana: OpticalDetector/
SimPhotonCounter_module.cc

```
//generating the tree for the light analysis:
if(fMakeLightAnalysisTree)
  {
    fLightAnalysisTree = tfs->make<TTree>("LightAnalysis","LightAnalysis");
    fLightAnalysisTree->Branch("RunNumber",&fRun);
    fLightAnalysisTree->Branch("EventID",&fEventID);
    fLightAnalysisTree->Branch("TrackID",&fTrackID);
    fLightAnalysisTree->Branch("PdgCode",&fpdg);
    fLightAnalysisTree->Branch("MotherTrackID",&fmotherTrackID);
    fLightAnalysisTree->Branch("Energy",&fEnergy);
    fLightAnalysisTree->Branch("dEdx",&fdEdx);
    fLightAnalysisTree->Branch("StepPrePositions",&fstepPrePositions);
    fLightAnalysisTree->Branch("StepPostPositions",&fstepPostPositions);
    fLightAnalysisTree->Branch("StepPreTimes",&fstepPreTimes);
    fLightAnalysisTree->Branch("StepPostTimes",&fstepPostTimes);
    fLightAnalysisTree->Branch("SignalsVUV",&fSignalsvuv);
    fLightAnalysisTree->Branch("SignalsVisible",&fSignalsvis);
    fLightAnalysisTree->Branch("Process",&fProcess);
  }
```

Track ID necessary to merge the light with the TPC information

Modifications to LArSoft out of sbndcode

lardata:
DetectorInfo/
LArProperties.h, LArPropertiesStandard.{h,cxx},
larproperties.fcl

```
TpbEmmisionEnergies: [0.05,1.0,1.5, 2.25, 2.481, 2.819, 2.952,2.988,3.024, 3.1, 3.14,3.1807, 3.54, 5.5, 50.39]
TpbEmmisionSpectrum: [0.0, 0.0, 0.0, 0.0588,0.235, 0.853, 1.0,1.0,0.9259,0.704,0.0296,0.011, 0.0,0.0, 0.]
TpbAbsorptionEnergies: [0.05,1.77,2.0675, 7.42, 7.75, 8.16, 8.73, 9.78,10.69, 50.39]
TpbAbsorptionSpectrum: [100000.0,100000.0, 100000.0,0.001,0.00000000001,0.00000000001, 0.00000000001, 0.00000000001,
 0.00000000001, 0.00000000001]
```

Modifications in lardata basically to include and manage the information of the WLS

larsim:
LArG4/
MaterialPropertyLoader.{h,cxx}

```
void MaterialPropertyLoader::GetPropertiesFromServices()

 if(LarProp->ExtraMatProperties()){
   SetMaterialProperty( "TPB", "RINDEX",                LarProp->RIndexSpectrum(),   1);
   SetMaterialProperty( "TPB", "WLSABSLENGTH",          LarProp->TpbAbs(),           CLHEP::m);
   SetMaterialProperty( "TPB", "WLSCOMPONENT",          LarProp->TpbEm(),            1);
   SetMaterialConstProperty( "TPB", "WLSTIMECONSTANT", LarProp->TpbTimeConstant(), CLHEP::ns);
   SetMaterialProperty( "vm2000", "RINDEX",             LarProp->RIndexSpectrum(),   1);
 }
```

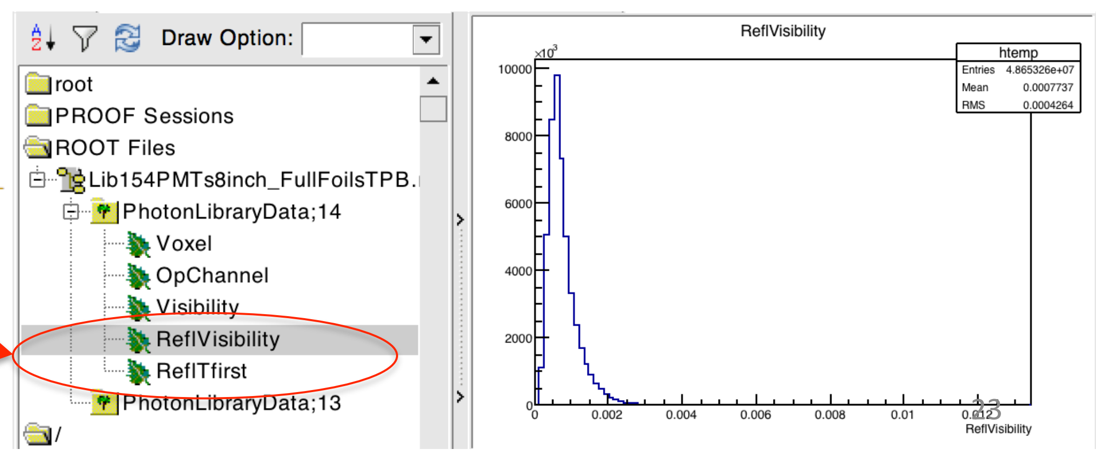
Modifications to LArSoft out of sbndcode

larsim:
PhotonPropagation/
PhotonLibrary.{h,cxx}, PhotonVisibilityService.h, PhotonVisibilityService_service.cc

Modifications in the photon visibility service to include/save and manage the information related with the reflected/visible light component

```
bool                fLibraryBuildJob;
bool                fDoNotLoadLibrary;
bool                fParameterization;
bool                fStoreReflected;
bool                fStoreReflT0;
```

If true, new branches in the optical library



Modifications to LArSoft out of sbndcode

Larsim: LArG4/ OpFastScintillation.{hh,cxx}

All the modifications needed for the "correction" of the arrival time of the photons, both direct and reflected components

```
G4double aSecondaryTime = t0 + deltaTime;
double propagation_time = arrival_time_dist_vuv.at(i)*CLHEP::ns;

// The sim photon in this case stores its production point and time
TVector3 PhotonPosition(x0[0],x0[1],x0[2]);

// We don't know anything about the momentum dir, so set it to be Z
float Energy = 9.7*CLHEP::eV;
float Time = aSecondaryTime + propagation_time;

// Make a photon object for the collection
sim::OnePhoton PhotToAdd;
PhotToAdd.InitialPosition  = PhotonPosition;
PhotToAdd.Energy           = Energy;
PhotToAdd.Time             = Time;
PhotToAdd.SetInSD          = false;
PhotToAdd.MotherTrackID    = tracknumber;

fst->AddPhoton(itdetphot->first, std::move(PhotToAdd));
```

```
// Parametrization of the VUV light timing (result from direct transport + Rayleigh scattering ONLY)
// using a landau + expo function.The function below returns the arrival time distribution given the
// distance IN CENTIMETERS between the scintillation/ionization point and the optical detectotr.
std::vector<double> OpFastScintillation::GetVUVTime(double distance, int number_photons) {
```

```
// Parametrization of the Visible light timing (result from direct transport + Rayleigh scattering ONLY)
// using a landau + exponential function. The function below returns the arrival time distribution given the
// time of the first visible photon in the PMT. The light generated has been reflected by the cathode ONLY.
std::vector<double> OpFastScintillation::GetVisibleTimeOnlyCathode(double t0, int number_photons){
```