

# Progress Report on Geant4 Optical Physics

Daren Sawkey

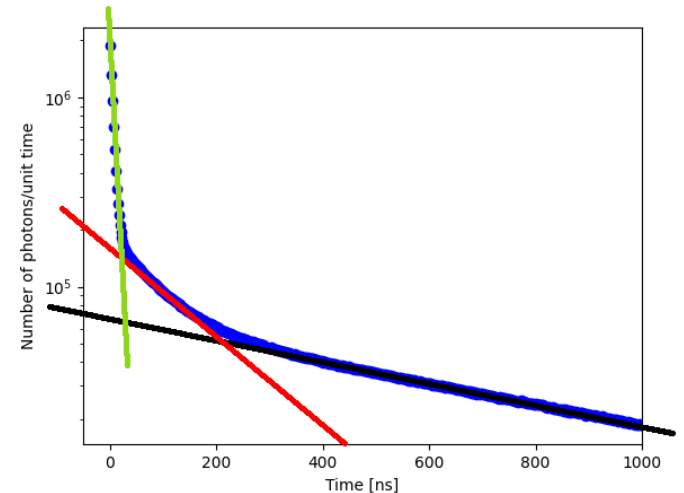
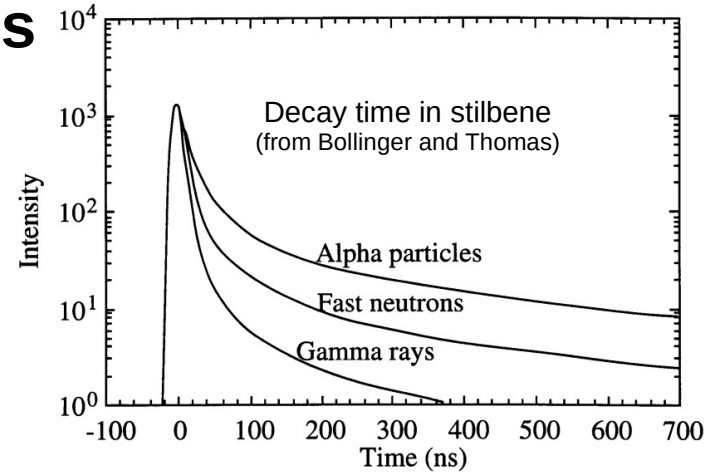
Sept 23, 2020  
2020 Geant4 Collaboration Meeting

# Outline

- New physics
  - Scintillation time constant
  - Additional wave length shifting
- Infrastructure
  - Optical parameters class and messenger
  - User defined material properties with `std::vector`
- Material data
- Examples
  - New macros `OpNovice2/fresnel.mac` and `wls2.mac`
  - All extended example macros in `Ctest`
- Validation
- Speed increase
  - Code improvements
  - GPU (Opticks)
  - Biasing

# New physics: Scintillation time constants

- In  $\leq 10.7$ , have the choice of fast and slow time constants, with the same yield for all particles; OR particle specific yields, with one time constant
  - Could also write your own physics list!
- In  $\geq 10.7$ , 3 time constants and particle-specific yields at the same time
  - by users requests
- **Both ways work now, but the old way to be deprecated in the next major release**
- In 10.6: material properties SCINTILLATIONYIELD and YIELDRATIO. New method uses SCINTILLATIONYIELD and SCINTILLATIONYIELD[1/2/3].
  - In both methods SCINTILLATIONYIELD gives number of photons (per unit energy).
- Fraction of photons in channel 1 is  $\text{SCINTILLATIONYIELD1}/(\text{SCINTILLATIONYIELD1} + \text{SCINTILLATIONYIELD2} + \text{SCINTILLATIONYIELD3})$  etc.
- Change material property names from [FAST/SLOW]TIMECONSTANT etc. to SCINTILLATIONTIMECONSTANT[1/2/3] etc.
- Analogous names for particles: PROTONSCINTILLATIONYIELD1 etc.

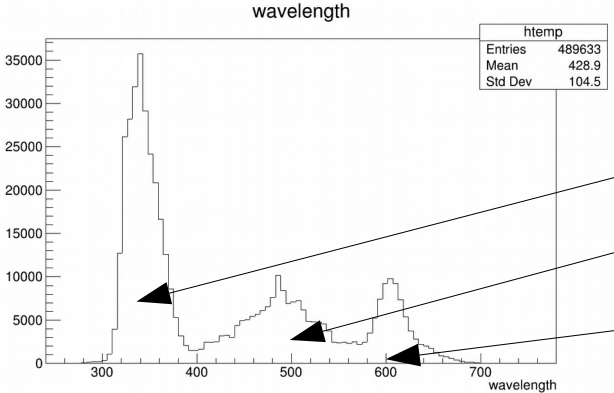


# New physics: Two WaveLengthShifting Process

- Process (and slide) by Alex Howard, Imperial College
- Since beta release a simple “clone” of the existing WLS has been included in G4OpticalPhysics constructor
- Cannot convolve the response function as it’s a discrete mechanism
  - either WLS-1 or WLS-2 and some transfer from WLS-1 to WLS-2
- Try it with OpNovice2/wls.mac

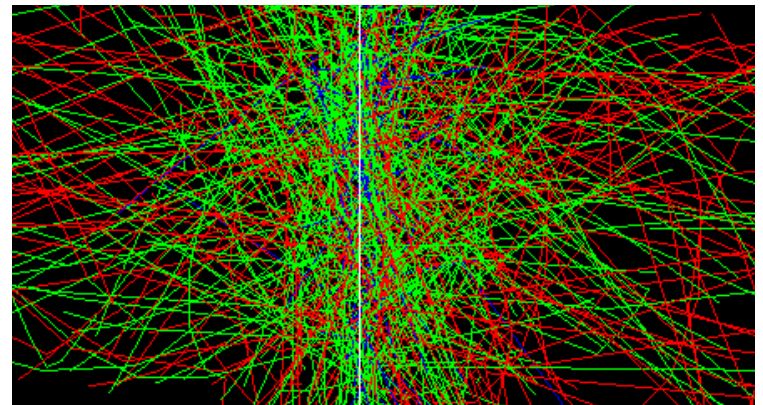
- Identical interface – append “2”:

```
scintCoreMaterialProperties->AddProperty("WLSCOMPONENT",wls1SpecVector);
scintCoreMaterialProperties->AddProperty("WLSCOMPONENT2",wls2SpecVector);
scintCoreMaterialProperties->AddProperty("WLSABSLENGTH",Wls1AbsEnergy,Wls1AbsLength,WLS1_ABS_ENTRIES);
scintCoreMaterialProperties->AddProperty("WLSABSLENGTH2",Wls2AbsEnergy,Wls2AbsLength,WLS2_ABS_ENTRIES);
scintCoreMaterialProperties->AddConstProperty("WLSTIMECONSTANT",Parameters::GetInstance()->WlsDecayTime()*ns);
scintCoreMaterialProperties->AddConstProperty("WLSTIMECONSTANT2",Parameters::GetInstance()->WlsDecayTime()*ns);
```



Generated Photons

**Blue: Primary Scintillation**  
**Green: WLS1**  
**Red: WLS2**



# Infrastructure: new G4OpticalParameters class

- Decouple the physics constructor and the parameters
- Based on G4EmParameters
- Use with G4OpticalPhysics constructor, or attach it to your own physics constructor
  - Improve thread safety
- User interaction via G4OpticalParametersMessenger
  - G4OpticalPhysicsMessenger deprecated, to be removed next major release
  - Macro commands unchanged by this
  - C++ method names the same as before, but different class
- UI commands now available PreInit and Idle states
- Redundant commands deprecated, to be removed next major release
  - Check command guidance
- Trying to give sensible names to commands. e.g `Configure(processName, true)` -> `SetProcessActivation(processName, true)`

# Infrastructure: Allow std::vector for material properties

- New method  
G4MaterialPropertiesTable::AddProperty(G4String, std::vector<G4double>, std::vector<G4double>)
- User code no longer requires C arrays
- Check that two vectors are same length now done in AddProperty()

## Old (still allowed)

```
G4double photonEnergy [] =  
    { 2.034*eV, 2.068*eV, 2.103*eV, 2.139*eV,  
      // ...  
      3.760*eV, 3.877*eV, 4.002*eV, 4.136*eV };
```

```
const G4int nEntries =  
    sizeof(photonEnergy)/sizeof(G4double);
```

```
G4double refractiveIndex1 [] =  
    { 1.3435, 1.344, 1.3445, 1.345,  
      // ...  
      1.359, 1.3595, 1.36, 1.3608};
```

```
assert(sizeof(refractiveIndex1) ==  
       sizeof(photonEnergy));
```

```
myMPT1->AddProperty("RINDEX",  
                    photonEnergy,  
                    refractiveIndex1,  
                    nEntries)
```

## New

```
std::vector<G4double> photonEnergy =  
    { 2.034*eV, 2.068*eV, 2.103*eV, 2.139*eV,  
      // ...  
      3.760*eV, 3.877*eV, 4.002*eV, 4.136*eV };
```

```
std::vector<G4double> refractiveIndex1 =  
    { 1.3435, 1.344, 1.3445, 1.345,  
      // ...  
      1.359, 1.3595, 1.36, 1.3608};
```

```
myMPT1->AddProperty("RINDEX",  
                    photonEnergy,  
                    refractiveIndex1)
```



# Data need to be specified by the user

- Why?
- Not aware of any other case in Geant4 this needs to be done
- **Proposal:** include optical data with distribution!
- Large benefit for new users ( == young users ? ) to have data already part of Geant4
- The data is out there
- Any users of optical physics willing to contribute data?
  - Especially large experiments might have this data
- Still need to design data storage etc.

Listing 64 Specification of scintillation properties in [DetectorConstruction](#).¶

```
const G4int NUMENTRIES = 9;
G4double Scnt_PP[NUMENTRIES] = { 6.6*eV, 6.7*eV, 6.8*eV, 6.9*eV,
                                  7.0*eV, 7.1*eV, 7.2*eV, 7.3*eV, 7.4*eV };

G4double Scnt_FAST[NUMENTRIES] = { 0.000134, 0.004432, 0.053991, 0.241971,
                                     0.398942, 0.000134, 0.004432, 0.053991,
                                     0.241971 };
G4double Scnt_SLOW[NUMENTRIES] = { 0.000010, 0.000020, 0.000030, 0.004000,
                                     0.008000, 0.005000, 0.020000, 0.001000,
                                     0.000010 };

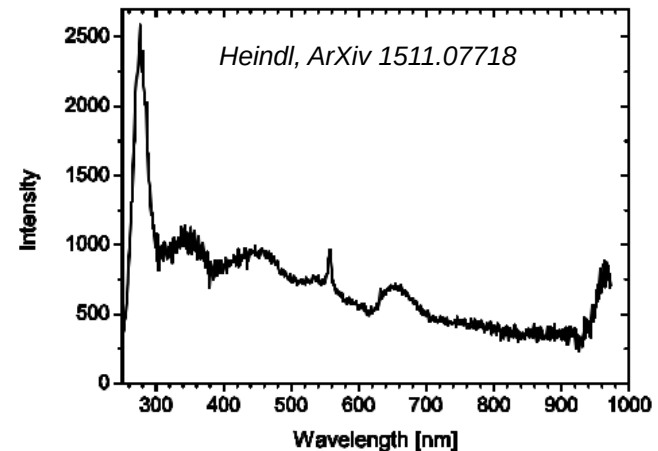
G4Material* Scnt;
G4MaterialPropertiesTable* Scnt_MPT = new G4MaterialPropertiesTable();

Scnt_MPT->AddProperty("FASTCOMPONENT", Scnt_PP, Scnt_FAST, NUMENTRIES);
Scnt_MPT->AddProperty("SLOWCOMPONENT", Scnt_PP, Scnt_SLOW, NUMENTRIES);

Scnt_MPT->AddConstProperty("SCINTILLATIONYIELD", 5000./MeV);
Scnt_MPT->AddConstProperty("RESOLUTIONSCALE", 2.0);
Scnt_MPT->AddConstProperty("FASTTIMECONSTANT", 1.*ns);
Scnt_MPT->AddConstProperty("SLOWTIMECONSTANT", 10.*ns);
Scnt_MPT->AddConstProperty("YIELDRATIO", 0.8);

Scnt->SetMaterialPropertiesTable(Scnt_MPT);
```

Scintillation emission spectrum of liquid Ar



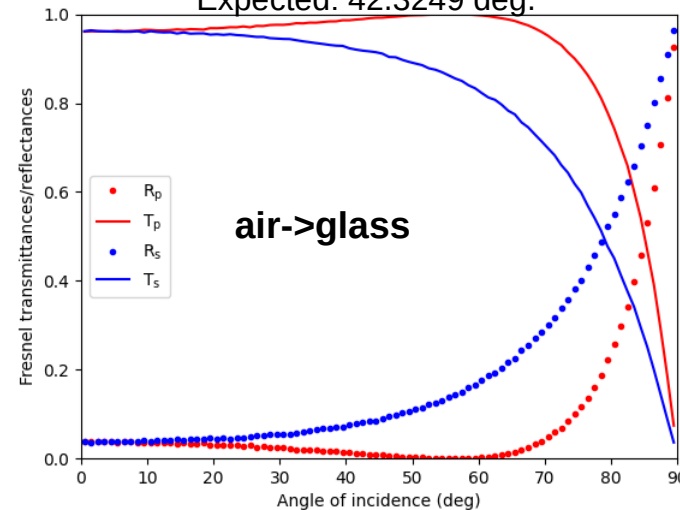
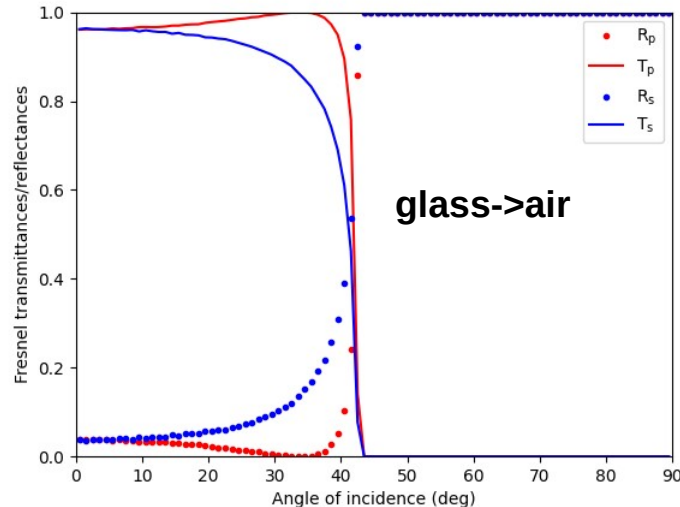
# Examples: new macro fresnel.mac -> validation

- OpNovice2 extended example
- Calculate reflectance, transmittance vs. incident angle and polarization
- Not earth-shattering but:
  - Exercise polarisation and boundary process
  - Text results to cdash
  - Need more of these!

Example output:

Reflectance shows a minimum at: 33.5 +/- 0.5 deg.  
Expected Brewster angle: 33.9537 deg.

Transmission goes to 0 at: 42.5 +/- 0.5 deg.  
Expected: 42.3249 deg.



# Validation

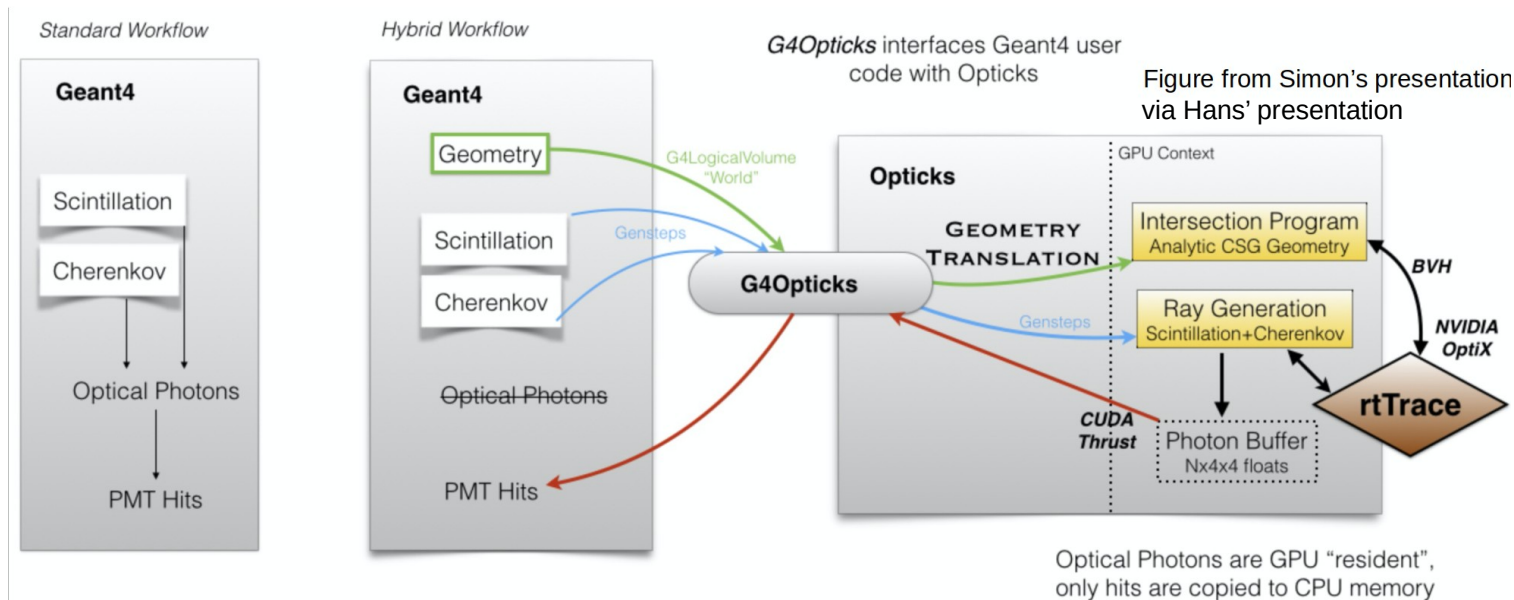
- No systematic optics validations yet
- Somewhat artificial if data need to be entered by user
  - Include data first?
- Include tests in Geant-val etc
  - Open to discussion: what are the priorities?

# Speed-up

- Optical physics is slow
  - Krzysztof's talk on Monday
- Two problems
  - Code not very efficient
    - Improvements measured in percent
    - Need to make sure physics unchanged when making any changes
    - Some improvements are apparent:
      - Caching of material properties, sin/cos calculations
    - Could use profiling tools described by Guilherme Tuesday
      - Any interest? Anyone with a realistic application to share?
  - Many optical photons produced
    - FNAL and others need orders of magnitude improvement

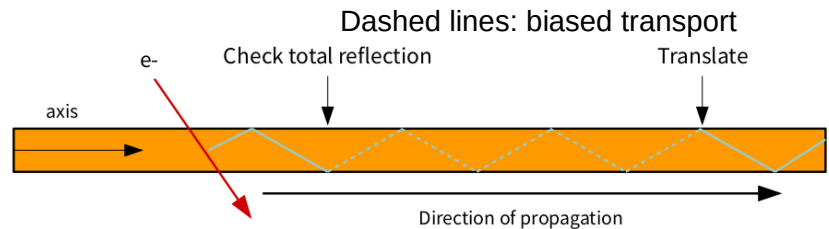
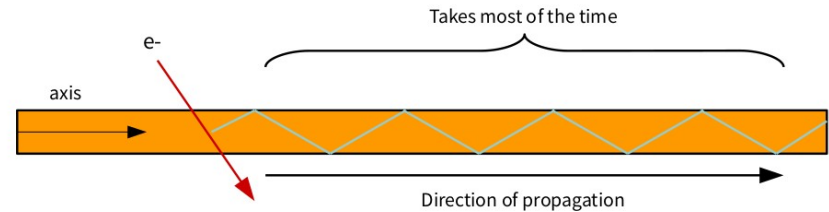
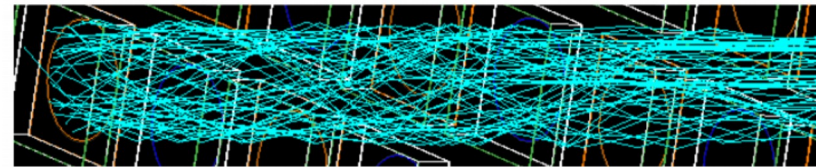
# Speed-up: Offload to GPU

- See talk by Hans Wenzel at R&D Task Force parallel session
- Integration of Opticks with Geant4: FNAL
- Transport the huge number of photons on a GPU
- Opticks: <https://simoncblyth.bitbucket.io/>



# Speed-up: Generic biasing

- Sanghyun Ko, Seoul National University
- R&D presentation: <https://indico.cern.ch/event/915715/#2-fast-optical-photon-transport>
- Total internal reflection in optical fiber not efficient
- With generic biasing, 70x speed-up observed
- Maybe extended example



# Summary: discussion topics

- Add material properties to Geant4 distribution?
- Any new physics to add?
- Realistic geometries/physics for efficiency benchmarking and improvements?
- Which physics validations to add to geant4-val (e.g.)?
- Documentation is dense—suggestions to improve?