Integration of Opticks[1] and Geant4 (an advanced example: CaTS)

Hans Wenzel
Krzysztof Genser
Soon Yung Jun
Alexei Strelchenko

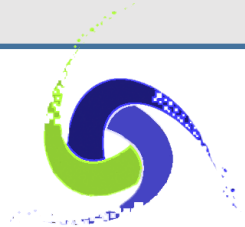
Fermilab-SLIDES-21-106-SCD

[1]developed by Simon Blyth
(Institute of High Energy Physics, Chinese Academy of Sciences)

Outline

- Motivation:
  - Liquid Argon TPC's,
  - Scintillation/Ionization properties in liquid Argon,
  - The computational challenge.
- Opticks/G4Opticks.
- CaTS.
- Timing and Validation.
- Summary and Outlook.
- Next Steps.

CaTS: Calorimetry **a**nd **T**racking **S**imulation

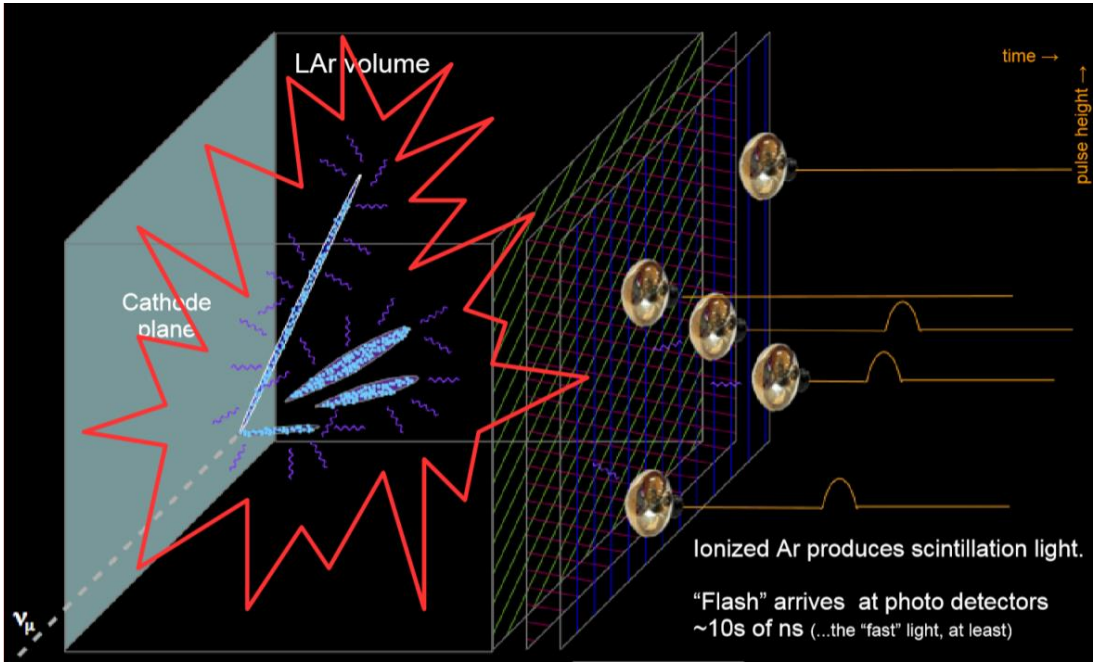


GEANT4
A SIMULATION TOOLKIT

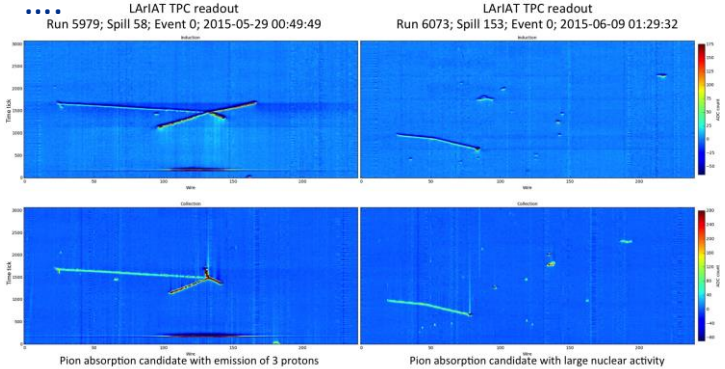
Fermilab

Motivation

Liquid Argon TPC's are the technology of choice for many neutrino experiments: DUNE, protoDUNE, μBoone, LArIAT, ICARUS, SBND… as well as dark matter searches: DarkSide, ARGO, …



The image below shows a display of the ionization signal: time + stereo wires or pixels allows for 3D reconstruction
dEdx for PID
track length for energy/momentum



See e.g.,: Dorota Stefan:
https://indico.cern.ch/event/575069/contributions/2326563/attachments/1363382/2064171/LArPrinicipals.pdf

Scintillation/Ionization properties in liquid Argon[1]

- Liquid Argon is a very efficient scintillator with a very high light yield that depends on E-field and purity (Scintillation and e^--I^+ pair production are competing effects.). For the simulation here we assume no E-field (Scintillation only) and a yield of 50,000 photons/MeV. A minimum ionization particle deposits 2.105 MeV/cm.
- Liquid argon scintillation photons are emitted in a narrow band of 10 nm centered around 128 nm.
- To match the photon wavelength to the quantum efficiency of the photodetector wavelength shifters (WLS) are used. A typical material is TPB with an emission spectrum peaked at 430 nm. WLS is "not" yet included in the simulation presented here.
- The time profile consist of two components (slow 1500 ns and fast 6 ns) originated by the decay of the lowest lying singlet, Σ1u+, and triplet states, Σ3u+, of the excimer Ar2* to the dissociative ground state.
- Liquid Argon is highly transparent to its own scintillation light (absorption length in the 10s of meters).
- Rayleigh scattering length ~ 50-60cm.

[1]) Input to the simulation

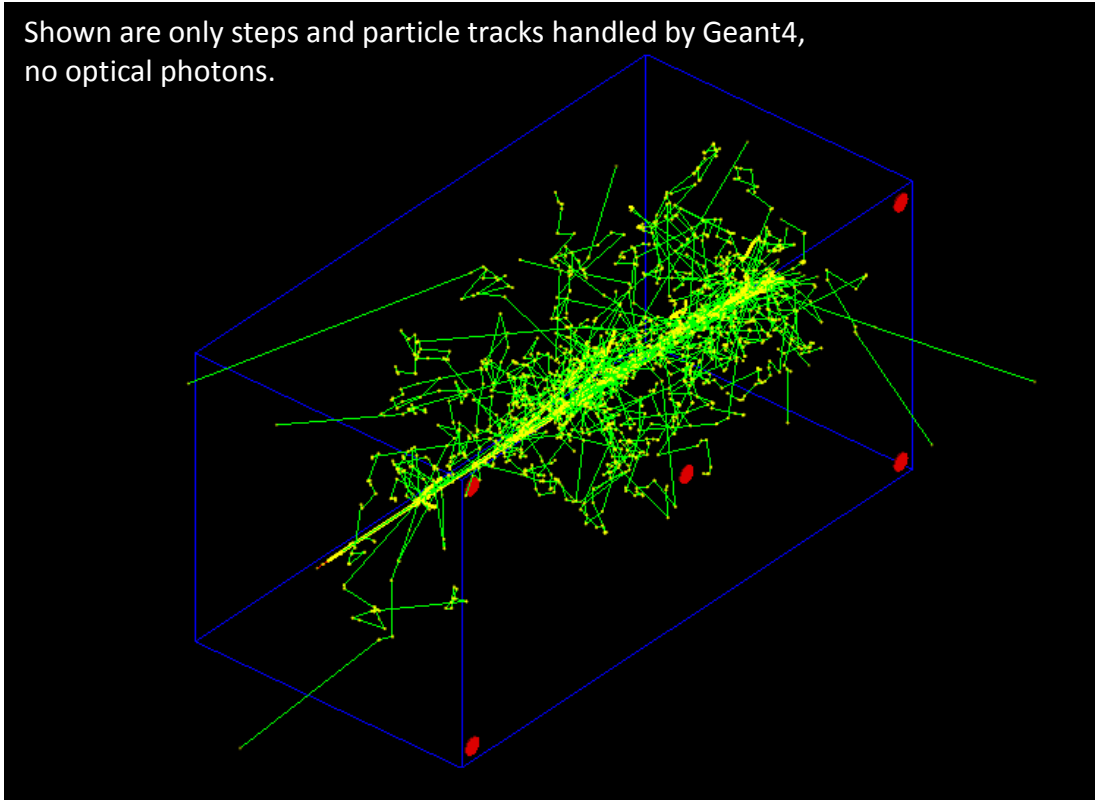
The computational challenge: Simulating a single 2GeV electron shower results in about 70 million VUV photons

Simple Geometry:
Liquid Argon: x y z: 1 x 1 x 2 m (blue)
5 photo detectors (red)
photon yield (no E-field): 50000 γ/MeV
single 2GeV electron (shower not fully contained)
(low Z=18, low $\rho = 1.78$ g/cm^3).

- 70 000 000 scintillation photons are produced.
- Using Geant4 to simulate photon generation and propagation on the CPU takes:

~41 minutes/event

(Compared to 0.056 sec/event when no optical photon simulation)



Opticks[1]

[1]developed by Simon Blyth

Opticks is an open source project. It translates Geant4 optical physics to NVIDIA® CUDA, OptiX™.
Generation and tracing of optical photons is the ideal application to be ported to GPU's:

- Only one particle involved (optical photon), but many of them (10s of millions)→ allows for massive parallelism (low latency, no big fluctuations in computing time.).
- Only a few simple physics processes need to be implemented on the GPU. The processes are :
  - G4Cerenkov (generate),
  - G4Scintillation (generate),
  - G4OpAbsorption,
  - G4OpRayleigh,
  - G4OpBoundaryProcess (only a few surface types),
  - G4OpWLS (not yet implemented),
  - G4OpMieHG, Reemission (not needed).
- These processes don't need a lot of input data (the so called Gensteps for the Cerenkov (C) and Scintillation (S) processes).
- Only few photons reach the Photodetectors and produce a Hit. So very little data to transfer between host and device. (Gensteps→ Device, PhotonHits → Host).
- Optical ray tracing is a well established field → benefit from available efficient algorithms (e.g., OptiX™).
- use NVIDIA® hardware (some with RTX: raytracing hardware acceleration) and software (CUDA, OptiX).

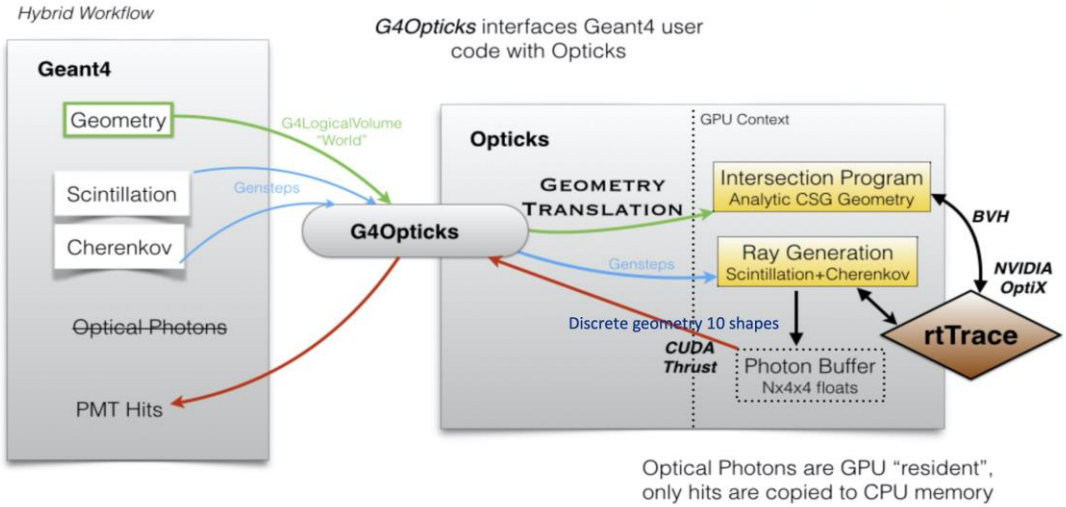
G4Opticks

G4Opticks (part of Opticks): interfaces Geant4 user code with Opticks. It defines a hybrid workflow where generation and tracing of optical photons is offloaded to Opticks (GPU/device) at stepping level when a certain amount photons is reached. Geant4 (CPU/host) handles all other particles.

The Geant4 Cerenkov and Scintillation (C/S) processes are only used to calculate the number of optical photons to be generated at a given step and to provide all necessary quantities to generate the photons on the GPU. The information collected is the so called GenStep which is different for Cerenkov and Scintillation (C/S).

Photon Hits are collected at the end of the G4Opticks call and added to the event hits collection.

Figure from Simon's presentation



Opticks Resources:

https://simoncblyth.bitbucket.io/env/presentation/opticks_may2020_hsf.html

EPJ Web of Conferences **214**, 02027 (2019), https://doi.org/10.1051/epjconf/20192140202
Simon Blyth:
"**Opticks : GPU Optical Photon Simulation for Particle Physics using NVIDIA® OptiX™ "**

Detector geometry in Opticks: https://indico.cern.ch/event/975008/
Documentation: https://simoncblyth.bitbucket.io/opticks/index.html

Code repositories:

https://bitbucket.org/simoncblyth/opticks/ : main development repository

https://github.com/simoncblyth/opticks : used for snapshots and tagged releases.
The most recent tag is https://github.com/simoncblyth/opticks/releases/tag/v0.1.2

CaTS: Calorimeter and Tracker Simulation

Geant4 Application that allows to simulate Detector setups ranging from a single Detector (e.g. single crystal) to complex setups (e.g. test beam experiment consisting of many detector elements). It is used to demonstrate the G4Opticks hybrid workflow, works with Geant4 > 10.7.01. Main Code repository: https://github.com/hanswenzel/CaTS

Its features are:

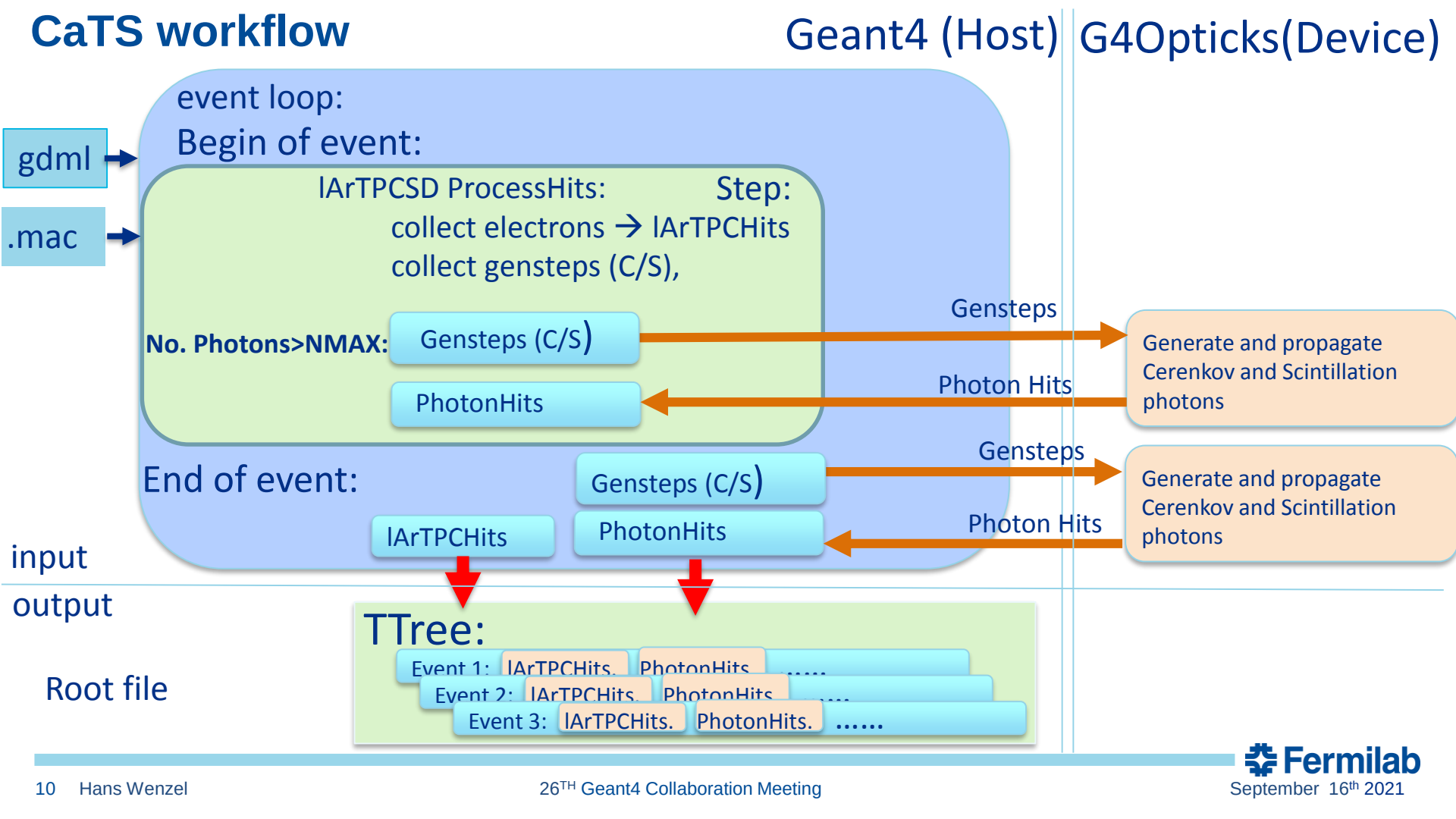
- No changes to Geant4! Only make use of provided interfaces: UserActions, Sensitive Detectors...
- G4Opticks is a runtime/build time option.
- Uses gdml with extensions for flexible Detector construction and to provide optical properties at runtime. The gdml extensions include:
  - Assigning sensitive detectors to logical Volumes. Available:
    - RadiatorSD, lArTPCSD, TrackerSD, PhotondetectorSD, MscSD, CalorimeterSD, DRCalorimeterSD,….
  - Assigning step-limits to logical Volume.
  - Assigning visualization attributes.
- Uses G4PhysListFactoryAlt to define and configure physics at runtime via command line option:

```
time ./CaTS -g simpleLArTPC.gdml -pl 'FTFP_BERT+OPTICAL+STEPLIMIT' -m time.mac

time ./CaTS -g simpleLArTPC.gdml -pl 'FTFP_BERT+OPTICAL+STEPLIMIT' -m time_G4.mac
```

- Uses Root IO to provide persistency for Hits.
- G4opticks:
  - Uses Geant4 to collect Scintillation and Cerenkov Gensteps. The harvesting is done in sensitive Detectors(SD) (RadiatorSD/lArTPCSD).
  - PhotonHits are collected in PhotonSD.

CaTS workflow



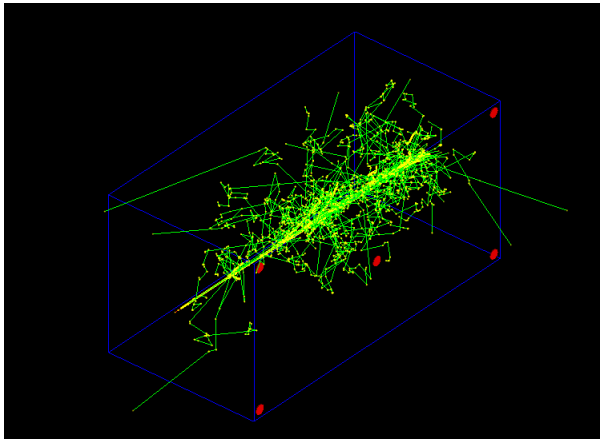
CaTS resources

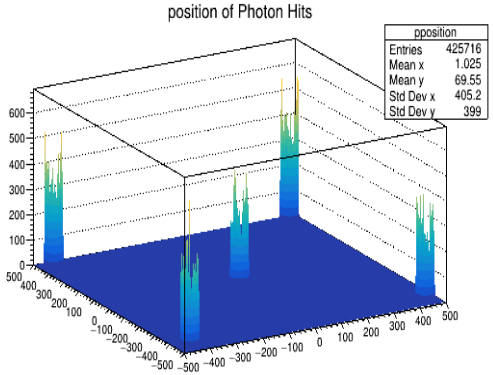
- Main git repository: https://github.com/hanswenzel/CaTS/: used for development.
- Instructions how to build and run CaTS: https://github.com/hanswenzel/CaTS/blob/master/README.md
- Instructions how to build Opticks and how to install all necessary software: https://github.com/hanswenzel/CaTS/blob/master/Instructions.md
- CaTS examples (examples consist of: gdml file, Geant4 macro and an application that makes histograms from the hits collections): https://github.com/hanswenzel/CaTS/blob/master/Examples.md
- CaTS is also available in fork of the Geant4 GitLab repository (as advanced example), used for snapshots and tagged releases.

https://gitlab.cern.ch/wenzel/geant4-dev/-/tree/CaTSv1_0/examples/advanced/CaTS

Performance:

Hardware:	
CPU	Intel(R) Core i7-9700K 3.6GHz 32 GB memory.
GPU	GeForce RTX 2070 CUDA Driver Version /11.3 CUDA Capability: 7.5 VRAM: 7981 Mbytes Cores: 2304





Timing results (Geant4 10.7.p01):	
Geant4 optical physics	2438 sec/event
G4Opticks, RNGmax[1] 10	6.45 sec/event
G4Opticks RTX enabled, RNGmax[1] 10	2.72 sec/event
G4Opticks, RNGmax[1] 100	6.86 sec/event
G4Opticks RTX enabled, RNGmax[1] 100	2.87 sec/event

1) Memory pre allocated for pre-initialized (at installation) curandState files to load.

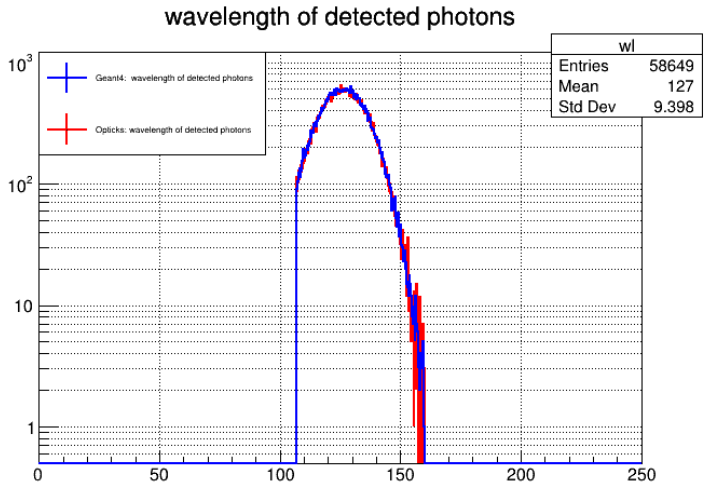
Geant4/(Geant4 + Opticks) comparison:
2438/6.45 = 378 (x 2.4 ~ 900 with RTX) x speed up
RTX Ray tracing hardware acceleration is usually not available on HPC platforms

Validation: Comparison of Geant4 and Opticks

PhotonHit:

register property of every photon that hits a Photo detector surface:

```
unsigned id{0};                 // id of Photo Detector
unsigned pid{0};                // Process ID
G4double wavelength{0};
G4double time{0};
G4ThreeVector position{0};
G4ThreeVector direction{0};
G4ThreeVector polarization{0};
```



Point like light source,
Photons at sensors-> still need
To compare Geant4/Opticks scintillation implementation in detail

Summary and Outlook:

- The work was performed as part of the Geant4 R&D Activity.
- We integrated Opticks with Geant4 (>10.7.p01). Small changes to Geant4 interfaces were introduced when needed.
- Provided timing/memory usage results. For our test case we observe an overall speedup by a factor of 390/900 (without/with RTX) compared to running Geant4 optical simulation on the CPU.
- The example CaTS:
  - Provides ROOT persistency for Hit collections.
  - Provides Sensitive detector plugins for different detector types.
  - Provides various detector geometries and configurations as gdml.
  - Gensteps chunks are collected in-situ during the stepping loop, once a predetermined chunk size, that can be set via a Geant4 messenger class, is reached the optical photon propagation is offloaded to the GPU (device). The rest of simulation and Gensteps collection is done on the CPU (host). The remaining Photons are processed at the end of Event.
- GPU Hardware is developing fast (accelerating). All results here are based on GeForce RTX 20 Series Turing platforms with first generation RTX.
- Expect performance boost from moving to OptiX™ 7 (Simon Blyth is working on it).

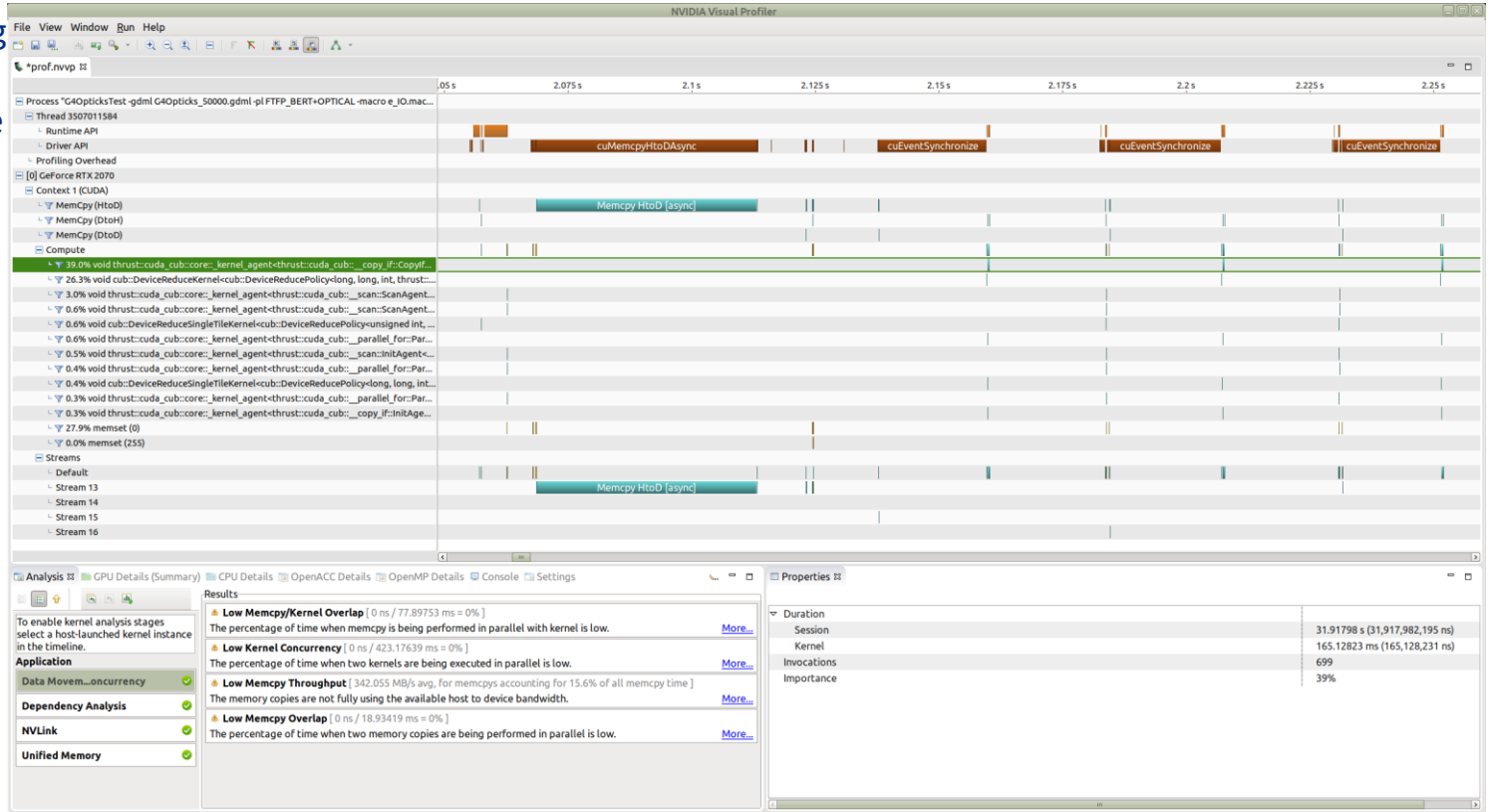
Next Steps

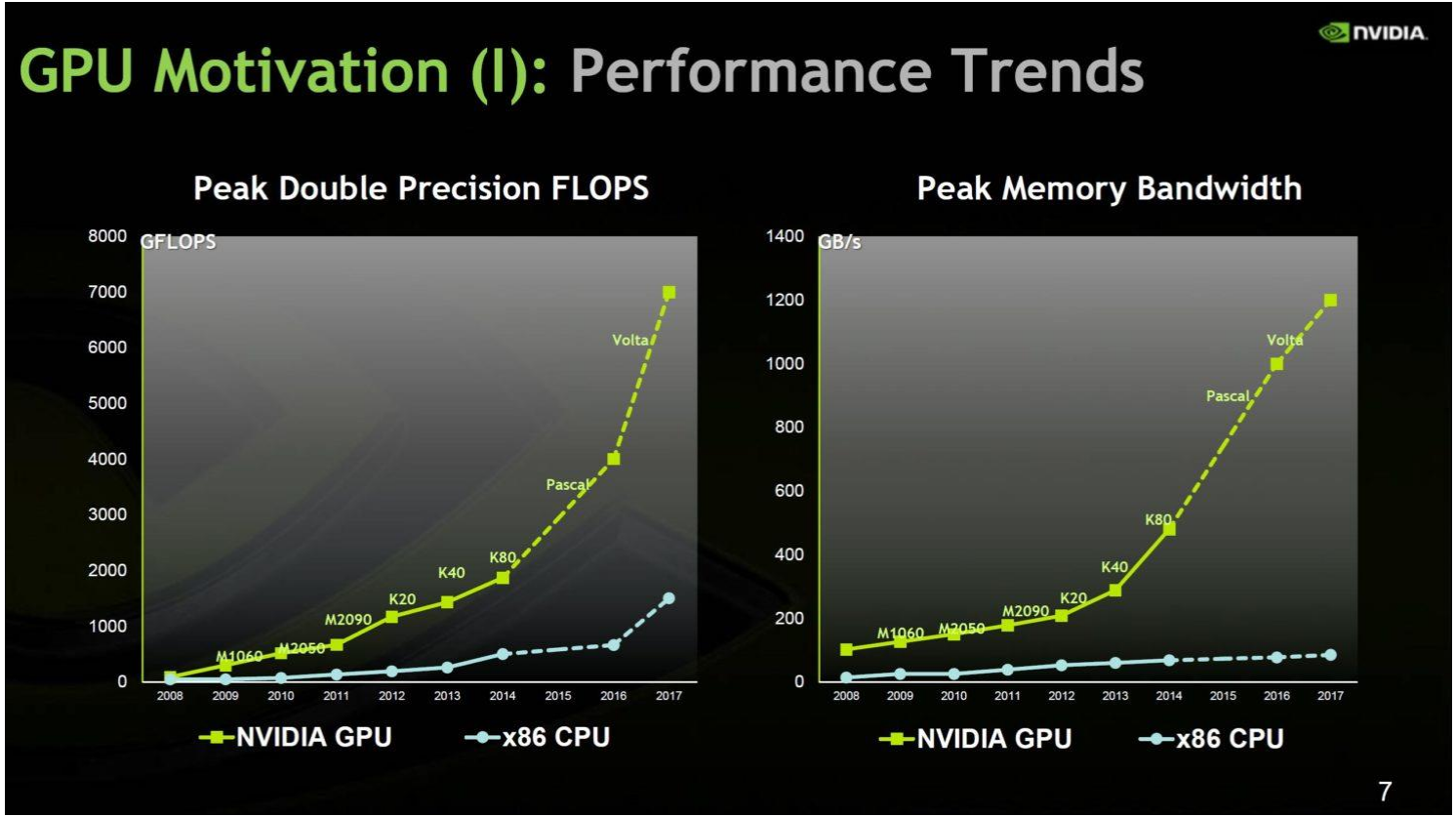
1. Make it an advanced Geant4 example. Available in GitLab for review, create pull request once ready.
2. Make use of modern C++ (11, 17) features.
3. Allocate resources at CERN where Opticks and CaTS can be installed and added to automated testing (Ben Morgan).
4. Create singularity container to run on HPC resources.
5. Identify the best workflow depending on application to best utilize modern hardware (GPU and CPU) making use of event-based multithreading as well as G4Tasking.
6. Implement wavelength shifting (WLS) process on GPU and update Opticks optical processes so that they correspond to the current Geant4 optical processes and uses the same material properties.
7. Evaluate treatment of random numbers on GPU.
8. Keep up with Opticks (e.g., move to Optix 7) development.

Backup slides

Profiling using NVIDIA's nvprof

Besides at the beginning there is very little data transfer between device and host.
(Gensteps-> Device, PhotonHits -> Host)

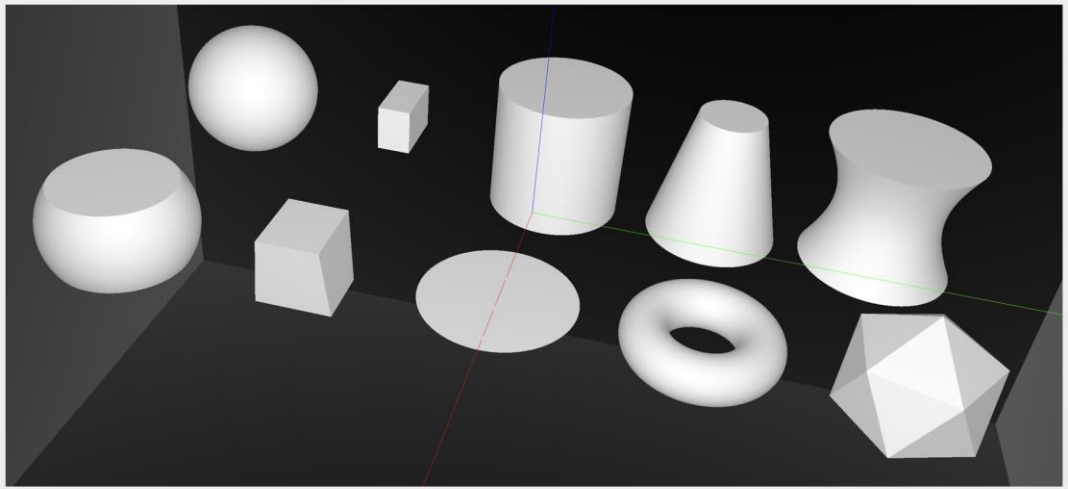




https://wccftech.com/nvidia-pascal-gpu-analysis/

G4Solid -> CUDA Intersect Functions for ~10 Primitives

- 3D parametric ray : **ray(x,y,z;t) = rayOrigin + t * rayDirection**
- implicit equation of primitive : **f(x,y,z) = 0**
- -> polynomial in **t** , roots: **t > t_min** -> intersection positions + surface normals



Sphere, Cylinder, Disc, Cone, Convex Polyhedron, Hyperboloid, Torus, ...

Gensteps

A Genstep collects all information necessary to generate Scintillation and Cerenkov photons on the GPU.

Cerenkov Genstep:

```
OpticksGenstep_G4Cerenkov_1042,          // (int)gentype      (0)
aTrack->GetTrackID(),                    // (int)ParenttId
 aMaterial->GetIndex(),                  // (int)MaterialIndex
 numPhotons,                             // (int)NumPhotons

 x0.x(),                    // x0.x          (1)
 x0.y(),                    // x0.y
 x0.z(),                    // x0.z
 t0,                        // t0

 deltaPosition.x(),                  // DeltaPosition.x   (2)
 deltaPosition.y(),                  // DeltaPosition.y
 deltaPosition.z(),                  // DeltaPosition.z
 aStep->GetStepLength(),             // step_length

 aParticle->GetDefinition()->GetPDGEncoding(),  // (int)code       (3)
 aParticle->GetDefinition()->GetPDGCharge(),    // charge
 aTrack->GetWeight(),                           // weight
 preVelocity,                                   // preVelocity

 betaInverse,                  //            (4)
 wl_minmax ? wmin_nm : pmin,
 wl_minmax ? wmax_nm : pmax,
 maxCos,

 maxSin2,                      //            (5)
 meanNumberOfPhotons1,
 meanNumberOfPhotons2,
 postVelocity
```