Optical Photon Simulation on GPUs

Simon C. Blyth

May 1, 2015

Contents

1 The Optical Photon Simulation Problem

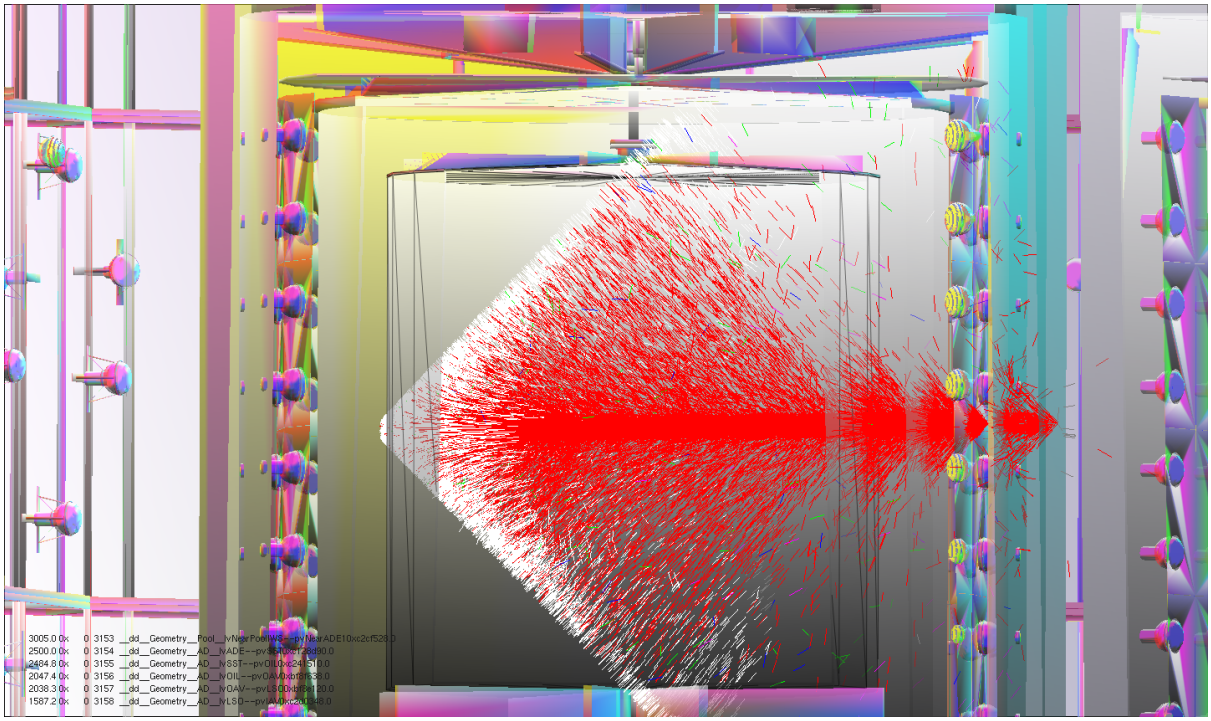

Figure 1: Side view of simulated Cerenkov photons from a 100 GeV muon travelling from right to left across the Daya Bay antineutrino detector. The shock front nature is apparent. Primary particles are simulated by Geant4, Cerenkov "steps" of the primaries are transferred to the GPU and photons are generated, propagated and visualized all on the GPU. Photons that hit PMTs are returned back to Geant4 to be included into standard hit collections. Photon colors indicate reemission (green), absorption(red), specular reflection (magenta), scattering(blue), no history (white).

1.1 Neutrino Detection with Optical Photons

Incoming neutrinos interact weakly with materials by the inverse beta decay process where an electron anti-neutrino is captured on a proton of the target resulting in the production of a positron and a neutron. The positron yields a prompt energy deposit followed by a delayed energy deposit from the neutron. Detector materials and geometry are chosen to enable detection of these energy deposits. The majority of neutrino detectors operate by the detection of visible light produced in transparent target materials such as water, ice or liquid scintillator. The weak nature of the neutrino interactions typically necessitate large volumes of material and large numbers of Photo Multiplier Tubes (PMT) to detect the light.

1.2 Importance of Optical Photons

Many detectors, including Daya Bay and JUNO, utilize liquid targets with materials chosen for their scintillation response to energy deposits resulting in re-emission in the form of visible light. In addition the liquids are required to have sufficient transparency within the appropriate wavelength range to allow adequate visible light to reach the surrounding PMTs, which in turn are chosen to have appropriate quantum efficiency within the wavelength range to convert the incident light into measurable electrical pulses. In addition to the dominant source of visible light from scintillation there is also a significant contribution from Cerenkov radiation. Cerenkov radiation is emitted when a charged particle travels through a medium at a speed greater than the speed of light in that medium.

1.3 Importance of Monte Carlo Simulation and Geant4

High energy physics experiments utilize Monte Carlo simulations to develop a detailed model of detector operation that provides essential insights throughout the lifetime of the experiment from detector design, development of reconstruction algorithms and eventually during data analysis in comparison with actual data. A detailed quantitative understanding of the entire chain from particle generation through to optical photon generation, propagation and detection is necessary for detectors to achieve their design goals and for data analysis to make best use of the data collected, allowing measurement uncertainties to be estimated.

The Geant4 simulation toolkit is used as the basis for the simulation of most High Energy Physics experiments, it has been developed over the past 20 years principally by Physicists working in Collider experiments, but has been adopted for the simulation of neutrino detectors including Daya Bay and JUNO and is used by many Scientists working in other fields including Medical Physics and Space Science.

1.4 Geant4 Optical Photon Simulation

Despite neutrino detectors typically being located underground to reduce the Cosmic muon background, this usually remains the principal background. Such cosmic muon background events in the Daya Bay simulation may yield up to several millions of optical photons which are individually propagated through the geometry of the detector by Geant4. Only a small fraction of the optical photons register as hits on the PMTs.

Unfortunately the serial nature of this propagation and the large numbers of optical photons make this the slowest single element of the Daya Bay simulation. Profiling indicates that more than 95% of CPU time is expended on propagating optical photons. The performance of the simulation of other PMT based detectors is expected to be similarly dominated by the bottleneck of optical photon simulation.

1.5 Geant4 Geometry Model Inhibits Performant Ray Tracing

The most computationally intensive aspect of optical photon simulation is the determination, at each step of the propagation, of the intersection point of rays representing photon directions with the geometry of the detector, which is represented in Geant4 by large trees of C++ objects corresponding to the volumes of the detector. Other than finding intersections, all that is computationally required at each step are a handful of random number draws and lookups of material and surface properties against simple data structures.

The problem with Geant4 optical photon simulation can be summarised as poor ray tracing performance caused by a geometry model which inhibits the use of modern acceleration techniques.

2 Solving the Optical Photon Simulation Problem

2.1 Bridging between Geant4 and external simulation

The Geant4 geometry model is central to the structure of the simulation toolkit and has adequately served the needs of Collider experiments for more than 20 years. Major changes to Geant4 that would only be of value to the minority who critically depend on optical photon simulation are unlikely. A hybrid solution combining external optical photon simulation with standard Geant4 simulation of all other particles is called for.

Fortunately optical photons constitute the final stage of the physical simulation making it straightforward to integrate the results of external optical photon generation and propagation back into Geant4 in the form of collections of PMT hits. Implementing external optical photon simulation necessitates bridging geometry, material and surface properties and PMT identity information from Geant4 to the external simulation. Also simulated event data needs to be communicated and PMT hits returned and integrated into Geant4 hit collections. I developed the G4DAE and G4DAEOpticks software packages to provide this bridging.

2.2 G4DAE : Bridging Geometry Information

My development of the G4DAE geometry exporter began in October 2013, the exporter writes COLLADA/DAE files (a 3D industry standard format) containing the Geant4 triangulated vertices of the geometry together with material and surface properties. The bulk of development of the G4DAE exporter was completed by early 2014.

The G4DAE exporter has wide applicability within the Geant4 user community as it allows detector geometries to be visualized with many commercial and open source packages. Many of these packages use modern OpenGL techniques providing drastically faster visualization than that provided by Geant4 itself.

I was invited to present the G4DAE package at the 19th Geant4 Collaboration Meeting held in Okinawa in September 2014. The Geant4 Collaboration accepted my proposal to contribute the G4DAE exporter to Geant4 and we plan to include it with the 2015 Geant4 release. This will entail work to perform the inclusion into the latest Geant4 version and test the exporter with a large variety of detector geometries.

2.3 G4DAEOpticks : Bridging Event Data

G4DAEOpticks provides transport of photons, hits and Cerenkov and Scintillation generation steps and allows externally formed hits to be integrated within standard Geant4 hit collections. I integrated earlier developments to form the G4DAEOpticks runtime bridge starting from September 2014, the intended features were completed by January 2015.

- request/response network communication of Geant4 data, implemented using NumPy serialization and ZeroMQ queues, with optional JSON based metadata
- integration of returned PMT hit data into standard Geant4 hit collections allowing subsequent processing such as electronics simulation to proceed unchanged
- data definitions for photons, Cerenkov photons, scintillation photons, scintillation steps, Cerenkov steps and PMT hits
- Geant4 level implementation to facilitate integration with simulation packages of any experiment via minor changes to the Scintillation and Cerenkov processes.

G4DAEOpticks was originally intended to provide a bridge to external photon propagation with the Chroma optical photon simulation, however the bridge is general in nature and can be used unchanged with other external optical photon simulation packages. To reflect this generality the former name of G4DAEChroma has been changed to G4DAEOpticks.

2.4 Chroma : GPU based Optical Photon Simulation

Chroma is an open source project providing ultra-fast photon simulation using GPU processing with CUDA. It was originally developed by Stanley Seibert, University of Pennsylvania, within the context of LBNE detector design studies. Chroma claims optical photon simulation speeds 200 times faster than Geant4.

The Chroma project provides:

- optical photon simulation CUDA C kernels
- geometry intersection CUDA C kernels using a boundary volume heirarchy BVH
- infrastructure for compiling and launching kernels using Python, PyCUDA and NumPy

Following initial investigations in early 2014, I forked Chroma in April 2014, my modifications from April 2014 to January 2015 can be summarized:

- integrated G4DAE exported geometry loading allowing pre-existing detector geometries to be uploaded to the GPU and used with Chroma
- enabled usage on non-workstation GPUs which demand short kernel launch times to avoid system freezes
- adopted CUDA/OpenGL interoperation techniques to allow efficient visualization of Chroma generated data

- adopted NumPy serialization format for efficient transport of Geant4 event data, eliminating multiple levels of data marshalling
- added generation step transport and optical photon generation via Cerenkov and Scintillation processes, reducing data transport overheads by a factor of 100 compared to the transport of photons

2.5 Adopting NumPy serialization format

In order to transport photon or other data over the network or between processes it is necessary to first convert objects into a serialized stream of bytes, transport the bytes and then deserialize the bytes back into objects at the other end. The technique initially adopted for communication between Geant4 and external simulation used ROOT TObject serialization. This approach was found to be inconvenient due to the large number of data transformations required for the communication from Geant4 to external simulation and onwards to the GPU and back again.

Instead the NumPy serialization format was adopted, the extreme simplicity of the format allows G4DAEOpticks to effectively fill NumPy arrays directly from Geant4 C++, which after deserialization can be copied to the GPU without any transformation. NumPy is the most popular package for scientific computing with Python.

2.6 GPU Generation of Scintillation and Cerenkov Photons

Runtime integration by collecting and transporting photons was found to cause large overheads due to the up to 200MB of photon data per event. As the primary particles of the simulation step through the materials of the detector optical photons are regarded to be generated by only two physical processes:

- Scintillation light from the scintillating liquids of the detector
- Cerenkov light from particles travelling at speeds greater than the speed of light in the liquids

These processes are modelled by G4Cerenkov and G4Scintillation subclasses in which the number of optical photon secondaries to generate are calculated followed by a loop over the photons to perform the generation. Typically each step results in up to 300 optical photons being generated.

Rather than collect photons the approach was modified to collect the parameters of the Cerenkov and Scintillation steps including the number of photons to generate. Using these step parameters as inputs I reimplemented the optical photon generation in CUDA C as an extension to Chroma, allowing GPU generation of the optical photons. This modification allowed the amount of transported data per simulated event to be reduced by a factor of 100. Also the requirement to copy the photon data to the GPU was avoided. Near perfect agreement between distributions of the parameters of the GPU generated photons and Geant4 counterparts has been achieved.

3 Background on OptiX and Migration Progress

3.1 Finding NVIDIA OptiX

Chroma's python based infrastructure provides an excellent fast development and learning environment, however for production usage a C++ infrastructure is preferable as this facilitates library integration with other projects and makes multi-threading easier. Ease of multi-threading is particularly important as it is required to make efficient use of multiple GPUs.

Towards the end of 2014, I became increasingly concerned regarding the difficult development work that would be necessary to allow Chroma to make efficient use of multiple GPUs. My searches for projects making efficient use of multiple GPUs led me to NVIDIA OptiX. OptiX is a C++/CUDA based framework providing accelerated ray tracing. The OptiX project makes compelling claims:

- state-of-the-art GPU accelerated ray tracing performance, improved with each release of OptiX and tuned to fully exploit new GPU architectures
- performance scaling across multiple GPUs, or even across multiple networked machines, with little development effort

- support for analytic geometry definition avoiding the need to tesselate and instanced geometry avoiding repetition of data; potentially drastic memory and performance savings may be achievable by exploiting these features to describe the repeated PMT geometry

Clearly shifting the burden of handling acceleration techniques and efficient GPU/multi-GPU utilization to OptiX, a project actively maintained by NVIDIA engineers, is hugely advantageous compared to using the ray tracing acceleration provided by the Chroma project.

The OptiX framework is freely distributed by NVIDIA for non-commercial usage. It is used as the basis for NVIDIA's commercial renderers: Iray and mental ray, suggesting that OptiX will remain actively maintained and improved over the coming years. These commercial renderers use OptiX to provide photorealistic image rendering using physically based material definitions to many companies in the Design and Film industries, including Adobe, Canon, Sony, Honda, Lockheed, Pixar and Disney.

3.2 OptiX Raycasting

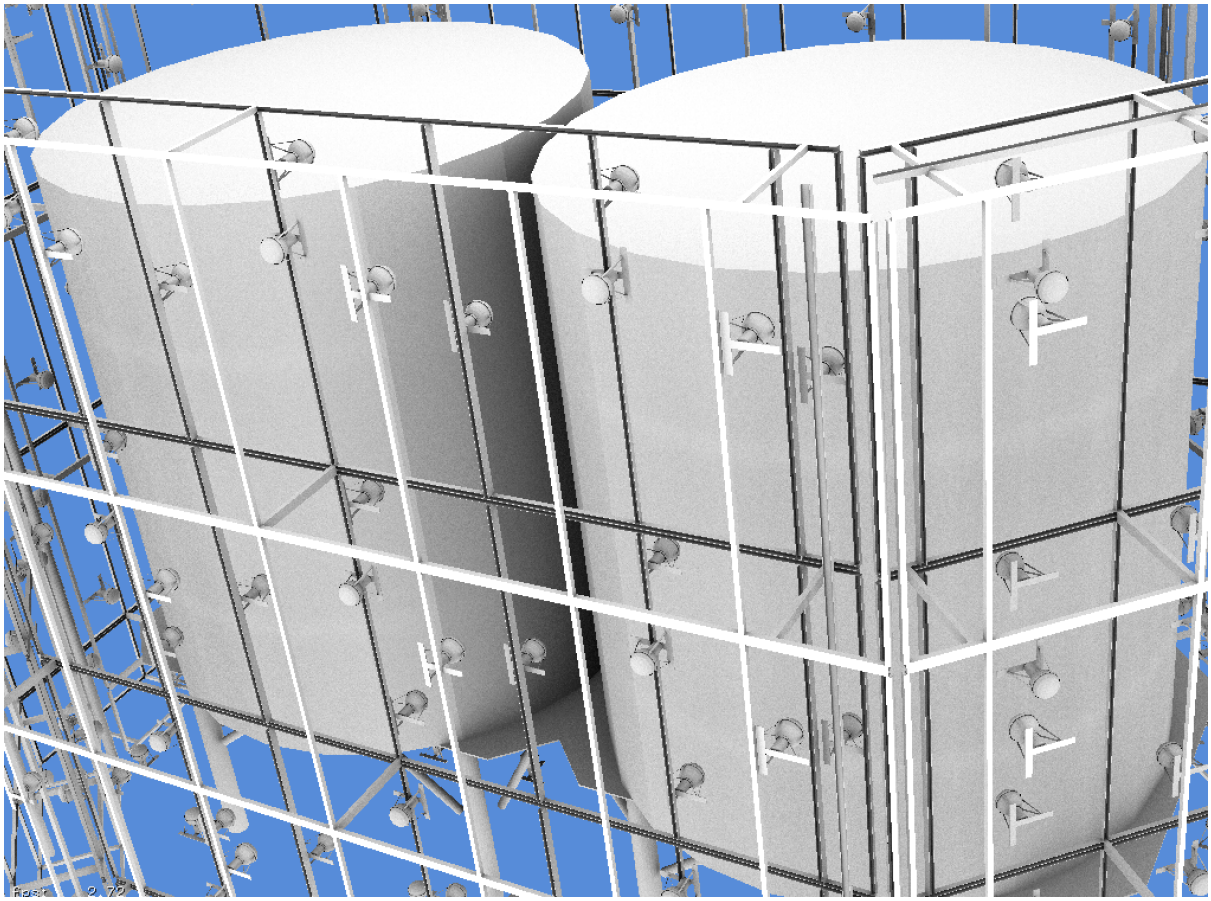

Figure 2: NVIDIA OptiX raycast of Daya Bay near site geometry at interactive speeds of 30 frames per second for 1024x768 pixels, corresponding to 23M intersection tests per second on a MacBook Pro (2013) with NVIDIA GeForce GT 750M 2048 MB.

With OptiX raycast renders can be generated at interactive speeds approaching 30 frames per second. This contrasts markedly with Chroma where comparable renders take ~1.8s per frame, corresponding to a 50x speedup. This speedup factor is sufficiently large to make development of carefully fair comparisons unnecessary.

3.3 OptiX Multiple GPU Scaling

In order to assess OptiX scaling across GPU cores the performance of a borrowed workstation at IHEP with two Tesla K20m GPUs was compared with my laptop via rendering benchmarks with the Daya Bay geometry. The

same code was run on my laptop and on the workstation in two different configurations.

- 2 Tesla K20m (4992 cores) 28.0 ms/f
- 1 Tesla K20m (2496 cores) 49.1 ms/f
- 1 GeForce GT 750m (382 cores) 345.1 ms/f

Raycast rendering performance was found to be approximately linear with the number of CUDA cores, with the 2 GPU workstation which has 13x the number of cores being 12x faster than my laptop. This indicates that OptiX is succeeding to schedule and balance work to the GPUs in a near optimal manner.

3.4 Migration of Infrastructure to C++

The OptiX framework provides host and device side APIs in C++ and CUDA C. Although complicated approaches to retain some of the existing python infrastructure could have been attempted, my experience directed me to the simplest possible approach of reimplementing the infrastructure using equivalent C++ libraries to the python dependencies where available as being the best approach to minimise code maintenance over the longterm.

External optical photon simulation depends on the infrastructure both in operation and for its development and debugging. Aspects covered include the below, some of these are detailed in subsequent sections.

- conversion of geometry and material/surface properties into the OptiX model
- OpenGL rasterized visualization of geometry and event data
- OptiX raytraced visualization of geometry data
- cuRAND random number generation operational within OptiX program
- wavelength interpolated material/surface property lookups

Application steering, asynchronous IO and event infrastructure have been reimplemented using the Asio-ZMQ package and several Boost C++ libraries including Boost.Asio, Boost.Program_options and Boost.Filesystem replacing python standard library packages and glumpy.

This migration work was done from Feb-April 2015 and is ongoing.

3.5 OptiX Geometry Handling using Assimp

Geometry and material and surface properties parsing of G4DAE exported geometry files have been reimplemented using my fork of the Assimp project replacing PyCOLLADA.

Assimp is an open source 3D geometry importer that supports a large number of file formats including DAE/COLLADA. I forked Assimp on github in February 2015 in order to add support for the extra material and surface properties included with G4DAE geometry exports from Geant4.

The complexity and size of the Assimp package made me reluctant for it to become a permanent dependency. Due to this I created the GGeo geometry package to act as an intermediary geometry format. My small AssimpWrap package orchestrates creation of the GGeo model from the imported Assimp model. The OptiX model is then created entirely from the GGeo model. GGeo provides a simple environment, that I fully control, in which to experiment with alternative approaches to creating OptiX geometries and representing surface and material properties within OptiX. Using GGeo I have developed a way of packing tables of material and surface properties as a function of wavelength into GPU textures. Wavelength dependent property lookups can then benefit from GPU hardware interpolation.

- https://github.com/simoncblyth/assimp
- https://bitbucket.org/simoncblyth/env/src/tip/optix/ggeo

3.6 Migration of Visualization

The prior visualizations of geometry and event data were based on PyOpenGL which forced the use of an ancient OpenGL version 2.1. The migration to C++ infrastructure has allowed adoption of modern OpenGL 4.1 which opens new possibilites for efficient visualization of OptiX generated data by using OptiX/OpenGL interoperation features.

OGLRap is a package of classes I developed to simplify the use of modern OpenGL, for example calculating the 4x4 homogenous matrices needed to control 3D view points and projections. Also renderer classes that simplify the use of OpenGL shaders to visualize geometry and event data are included.

- https://bitbucket.org/simoncblyth/env/src/tip/graphics/oglrap

3.7 OptiX Architecture

The OptiX engine provides a programmable ray tracing pipeline analogous to the OpenGL rasterization pipeline where user supplied OptiX programs take the place of OpenGL shaders. OptiX however focuses exclusively on the fundamental ray tracing computations and avoids making rendering specific assumptions. Seven different user supplied program types are compiled into a single optimized CUDA kernel. NVIDIA expertise regarding specific GPU architectures and ray tracing workloads is used in the optimization.

- Ray Generation programs provides the entry point into the ray tracing pipeline, they start the trace and store results into output buffers.
- Intersection programs implement ray-geometry intersection tests which are invoked to perform a geometric queries as acceleration structures are traversed. Simple ray triangle intersection could be provided but also analytic geometry intersection is possible.
- Bounding box programs compute the bounds associated with each primitive to enable acceleration structures over arbitrary geometry
- Closest hit programs are invoked once traversal has found the closest intersection of a ray with the scene geometry. They can cast new rays and store results into the ray payload.
- Any Hit programs are called during traversal for every ray-object intersection, the default of no operation is often appropriate.
- Miss programs are executed when the ray does not intersect any geometry
- Exception programs are called when problems such as stack overflow occur

The bulk of initial development effort for the optical photon simulation is expected to be within the ray generation and closest hit programs. In addition subsequent developments to adopt more memory efficient representations of repeated PMT geometry would require work on the intersection and bounding box programs.

3.8 Random Number Generation in OptiX programs

Chroma uses the cuRAND library, which is part of the CUDA toolkit, for the concurrent generation of millions of reproducible sequences of pseudorandom numbers. Concurrent generation is handled by assigning sub-sequences to each planned CUDA thread, and having the threads maintain state regarding positions within their sub-sequence. The cuRAND library requires initialization of these per-thread states with a CUDA kernel launch.

Initial attempts to initialize cuRAND state within OptiX programs failed, only by increasing the OptiX stack size by a factor of 10 could initialization be made to succeed. However with such large stack size the OptiX kernel was found to run very slowly.

A workaround was found to avoid this impasse by using a separate sequence of CUDA kernel launches to initialize the cuRAND state, persist that state to a cache file, and then load the state from the file into OptiX using OptiX/CUDA interop techniques. This allows OptiX to start from the initialized cuRAND state without needing the large stack size. I packaged this work into CUDAWrap.

- https://bitbucket.org/simoncblyth/env/src/tip/cuda/cudawrap

4 Plans for the year ahead

4.1 Porting Photon Generation and Propagation to OptiX

Cerenkov and Scintillation photon generation based on buffers of generation step parameters clearly belong within the OptiX ray generation program. From experience with Chroma porting the generation to OptiX is expected to be straightforward, however it constitutes the first real test of the random number generation approach adopted and may necessitate further infrastructure development.

Bringing Chroma photon simulation kernels into the OptiX model will entail dividing the Chroma propagation stepping loop between the OptiX ray generation and closest hit programs. As these stages communicate via a ray payload structure there is some flexibility and some experimentation is required to find a workable solution. The last step of forming GPU hits requires further infrastructure work to bring PMT identity information into OptiX.

4.2 Validation of Photon Generation and Propagation within OptiX

The photon generation code has already been validated by comparison against Geant4 within the Chroma context, thus it is expected to be straightforward to verify again within OptiX. Validation of the propagation is much less certain, it is expected that both the Geant4 and OptiX simulation will need to be instrumented to allow detailed step by step comparisons to be performed

4.3 OpenGL Visualization of OptiX Photon Propagation

Development and debugging with Chroma was greatly facilitated by 3D visualizations of geometries and event data. GPU performance and the capabilities of CUDA, OpenGL and OptiX to share access to GPU resident data structures has enabled unprecedented within HEP visualizations. I plan to port features from the former python based visualization into the C++ infrastructure as they are needed for simulation development.

4.4 Integration of G4DAE Geometry Exporter with 2015 release of Geant4

At the 19th Geant4 Collaboration Meeting in September 2014, the Geant4 Collaboration accepted my proposal to contribute the G4DAE exporter. The work of including it ready for release at the end of 2015 remains to be done. It is unclear how much work this will entail as the exporter has so far only been applied to a limited number of detector geometries.

4.5 Adopt more memory efficient Geometry representation

Neutrino detectors, such as the planned JUNO detector, often have very large numbers of PMTs. The geometry representation currently in use simply repeats the tesselated PMT geometry for every PMT. This initial approach has the advantage of simplicity by the disadvantage of large GPU memory requirements.

GPU performance is typically constrained by memory access time thus adopting a geometry representation with much smaller memory demands will not only allow use of GPUs with less available memory but has the potential to yield significant performance improvements. OptiX has two features that may allow a much more efficient geometry representation to be used:

- on GPU instanced geometry, allowing repeated geometry to be represented without duplication by using associated transformation matrices which are applied to photon paths rather than geometry.
- GPU algorithmic geometry definition potentially avoids the need to tesselate altogether, instead a small number of parameters that describe the PMT shape could be used together with an algorithm that computes ray intersections. This is similar to the way Geant4 operates serially on the CPU.

4.6 Moving Into Production

For fast development it is convenient for Geant4 processing and Optical Photon Simulation to run on different machines connected via my G4DAEOpticks message queue based infrastructure. Although such a split architecture remains an option in production it is possible that it will be advantageous to locate these together on the same machine or even within the same process.

The ZeroMQ transport mechanism that G4DAEOpticks is based upon promises to allow simple transitions between such different configurations. Some experimentation is neverthless required to find a workable solution within the memory constraints of the available workstation. The migration to a C++ infrastructure together with use of Boost threading and asynchronous IO libraries makes the likely need for multi-threaded architectures much more straightforward to implement.

The JUNO IHEP group has purchased a 4 GPU workstation that hopefully can be used to perform these configuration experiments and make performance tests. Production runs creating large Monte Carlo samples for validation and use by analysis groups will require working closely with the Monte Carlo Production groups of the Daya Bay and JUNO Collaborations.

4.7 Capitalizing on the Infrastructure developed

A side benefit of this work has been high performance interactive 3D visualizations of detector geometries and events. The machinery developed to liberate the geometry data and allow external simulation can directly be repurposed to implement an event display framework which to first order could be used by any experiment that uses Geant4.

I expect this would be of particular interest to smaller Collaborations lacking the manpower to develop a high performance event display. Creating a shared software repository in which a framework for building event displays can be developed would eliminate duplication of effort. Development of Daya Bay and JUNO event displays on top of this generic framework would be a good way to initiate this shared framework.

Such a repository and the framework would also provide a channel for demonstrating the use of GPU based optical photon simulation. Although a layered approach would need to be implemented as OptiX and CUDA require NVIDIA GPUs whereas OpenGL visualization is usable on almost any machine.

4.8 Distribution and Marketing

Once validations are well advanced and impressive performance measurements have been achieved it will be appropriate to market the approach and implementation with the aim of making GPU accelerated optical photon simulation into a widely adopted technique in PMT based experiments.

The machinery needs to be made straightforward to obtain, install, integrate and use. To this end a dedicated software repository website to manage the code and documentation needs to be created to act as focal point for communication.

Publicity channels such as collaboration talks, conference talks, technical papers and video sharing sites need to be used to bring the technique to a wider audience.

5 Status Summary

5.1 Development Overview

The design adopted to transparently replace the Geant4 simulation of optical photons with an external GPU based simulation is comprised of geometry, runtime and propagation categories.

Recreation of Geant4 geometry information on the GPU (G4DAE) was implemented between October 2013 and January 2014, this includes:

- Export Geant4 geometry as COLLADA/DAE files

- Import geometry files into external simulation and upload to GPU

Runtime bridging between Geant4 and external simulation (G4DAEOpticks) was implemented between September 2014 and January 2015, comprising:

- collect photon data or generation step parameters within the Cerenkov and Scintillation implementations and prevent further photon processing by Geant4
- send data to external process and receive reply containing PMT hits
- integrate hits into normal Geant4 hit collections

GPU optical photon simulation is currently being transitioned from Chroma to OptiX. The infrastructure was prepared between January-April 2015, and the porting to OptiX is in progress from April 2015. Broad aspects:

- receive requests over the network
- deserialize bytes received into NumPy arrays and copy these to the GPU
- generate photons and propagate them through the geometry and form hits
- copy photons back from the GPU and reply to Geant4 with hits

5.2 1000x performance, at a price

Large performance factors make good headlines, but the reality is that beyond a certain level of perhaps 100x the optical photon processing time becomes effectively zero compared to other processing. Given the raw intersection speed of OptiX and its scaling performance across multiple GPUs I am confident that performance will reach such a level. Using external GPU optical photon simulation will just become the way things are done.

The latest OptiX version 3.8 supports connection to one or more remote Visual Computing Appliances (VCAs). VCAs are dedicated GPU machines housing 8 NVIDIA Maxwell GPUs. OptiX based applications can with minimal changes use the compute power of clusters of such VCAs. However more modest GPU workstations are expected to provide sufficient performance to make optical photon processing no longer the rate determining step of simulation production.

5.3 Conclusion

I have performed a major re-implementation of simulation codes over the past four months, moving to a C++ infrastructure in order to benefit from NVIDIA OptiX. Over the past year, I completed development of the G4DAEOpticks bridge enabling Geant4 simulations to transparently benefit for external optical photon simulation. The ongoing transition to OptiX has yielded immediate performance benefits, but more importantly the future progression with OptiX releases and GPU architectures looks to be assured. Also the scaling across multiple GPUs and even multiple machines means achieving a performance level that makes the optical photon processing time effectively zero is highly probable.

Migration to OptiX has delayed the validation of external simulation but has also brought this work much closer to being suitable for production usage as delays from the development of efficient multi-GPU operation have been avoided.

Beyond the initial porting and validation the major focus of the year ahead is bringing this work into production usage first within the Daya Bay and JUNO Collaborations and then to the wider community of PMT based experiments.

5.4 Other Work : Daya Bay Infrastructure

Most of the software infrastructure systems I have developed or integrated over the past years remain in continuous operation with multiple processes running around the clock at the main Daya Bay institutions, monitoring every database update and every change to the Daya Bay software, performing software builds and tests and notifying experts of failure conditions.

My responsibilities include:

- Software Infrastructure (Trac and SVN servers)
- Offline Database (Management, Operating procedures, software interfaces)
- Online Slow Control Database (Monitoring and “Scraping” from Online to Offline)
- Data Quality Database backup
- System Administration (backups)

Despite the migration of some aspects of this support work to colleagues I remain the expert when problems or non standard situations arise. Occasional urgent requests for help with development of custom scripts, or with the usage of tools I developed are the most time consuming aspects of these responsibilities.