EPJ Web of Conferences **251**, 03009 (2021) https://doi.org/10.1051/epjconf/202125103009
CHEP 2021

Integration of JUNO simulation framework with Opticks: GPU accelerated optical propagation via NVIDIA® OptiX™

Simon Blyth[1,*]

[1]Institute of High Energy Physics, CAS, Beijing, China.

Abstract. Opticks is an open source project that accelerates optical photon simulation by integrating NVIDIA GPU ray tracing, accessed via NVIDIA OptiX, with Geant4 toolkit based simulations. A single NVIDIA Turing architecture GPU has been measured to provide optical photon simulation speedup factors exceeding 1500 times single threaded Geant4 with a full JUNO analytic GPU geometry automatically translated from the Geant4 geometry. Optical physics processes of scattering, absorption, scintillator reemission and boundary processes are implemented within CUDA OptiX programs based on the Geant4 implementations. Wavelength-dependent material and surface properties as well as inverse cumulative distribution functions for reemission are interleaved into GPU textures providing fast interpolated property lookup or wavelength generation. In this work we describe major recent developments to facilitate integration of Opticks with the JUNO simulation framework including on GPU collection efficiency hit culling which substantially reduces both the CPU memory needed for photon hits and copying overheads. Also progress with the migration of Opticks to the all new NVIDIA OptiX 7 API is described.

1 Introduction

Opticks[1-6] enables Geant4[7-9] based optical photon simulations to benefit from high performance GPU ray tracing made accessible by NVIDIA® OptiX™[10-12]. The Jiangmen Underground Neutrino Observatory (JUNO)[15] under construction in southern China will be the world's largest liquid scintillator detector, with a 20 kton spherical volume of 35 m diameter. The large size and high photon yield, illustrated in Figure 1, makes optical photon simulation extremely computationally challenging for both processing time and memory resources. Opticks eliminates these bottlenecks by offloading optical simulation to the GPU. Monte Carlo simulation is the primary technique used to design, optimize and analyse diverse detection systems. However, sequential simulation of large numbers of optical photons has extreme computational and memory costs. Opticks enables drastically improved optical photon simulation performance that can be transformative to the design, operation and understanding of diverse optical systems. Although Opticks was developed for the simulation of the JUNO detector, it is structured to enable use with other detector geometries. Any detector simulation limited by optical photons can benefit from Opticks.

Opticks was presented to the three prior CHEP conferences with proceedings focussing on various aspects of its development. The 2019 plenary presentation and proceedings[4]

*Corresponding author and speaker on behalf of the JUNO collaboration. e-mail: simon.c.blyth@gmail.com.

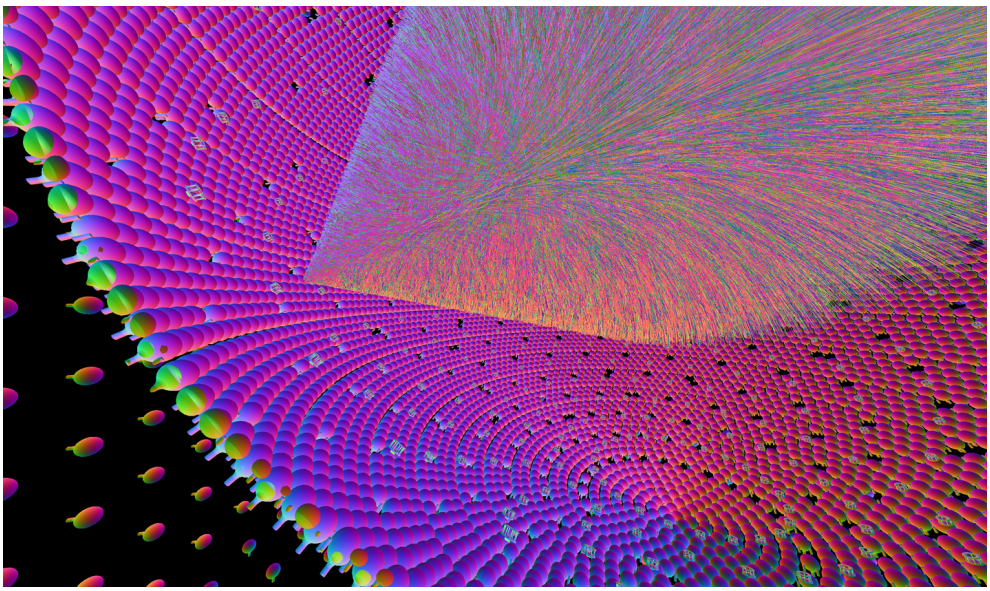

Figure 1. Cutaway OpenGL rendering of millions of simulated optical photons from a 200 GeV muon crossing the JUNO liquid scintillator. Each line corresponds to a single photon with line colors representing the polarization direction. Primary particles are simulated by Geant4, scintillation and Cherenkov "gensteps" are uploaded to the GPU and photons are generated, propagated and visualized all on the GPU. Representations of some of the many thousands of photomultiplier tubes that instrument the liquid scintillator are visible. The acrylic vessel that contains the liquid scintillator is not shown.

focused on performance measurements with ray trace dedicated RT cores via the RTX platform. The 2018 proceedings[5] cover the implementation of automated translation of Geant4 geometry into a GPU geometry. The 2016 proceedings[6] detail the CUDA port of Geant4 photon generation and optical physics as well as the use of GPU textures.

These proceedings focus on recent developments that facilitate the integration of Opticks with detector frameworks such as the JUNO simulation framework[14] using a minimal `G4Opticks` interface class that aims to simplify the integration by minimizing detector specific code. Also the implementation of angle dependent collection efficiency culling on GPU is described. This enhancement substantially reduces the CPU memory required for sensor hits. Relocating the culling decision to the GPU means that only photon hits needed by the next stage electronics simulation are copied from GPU to CPU.

1.1 Importance of optical photon simulation to JUNO

Cosmic muon induced processes are crucial backgrounds for neutrino detectors such as JUNO[15], necessitating underground sites, water shields and muon veto systems. Minimizing the dead time and dead volume, that result from applying a veto, requires an understanding of the detector response to a muon. Large simulated samples of muon events are crucial in order to develop such an understanding. The number of optical photons estimated to be produced by a muon of typical energy 200 GeV crossing the JUNO scintillator is at the level of tens of millions. Profiling the Geant4-toolkit-based simulation shows that the optical photon propagation consumes more than 99% of CPU time, and imposes severe memory constraints that have forced the use of event splitting. As optical photons in neutrino detectors

can be considered to be produced by only the scintillation and Cherenkov processes and yield only hits on photomultiplier tubes, it is straightforward to integrate an external optical photon simulation with a Geant4 simulation of all other particles.

1.2 GPU ray tracing

Graphics Processing Units (GPUs) evolved to perform rasterized rendering, optimizing throughput[17] rather than minimizing latency like CPUs. Greater GPU area dedicated to parallel compute increases throughput across all threads at the expense of slower single-thread execution. Threads blocked while waiting to access memory are tolerated by using hardware multithreading to resume other unblocked threads, which hides latencies when there are sufficient parallel threads in flight. GPUs are suited to problems with many millions of independent low resource parallel tasks allowing thousands of threads to be in flight simultaneously. Optical simulation is well matched to these requirements with abundant parallelism from huge numbers of photons and low register usage from simplicity of the physics.

The most computationally demanding aspect of photon propagation is the calculation of intersection positions of rays representing photons with the detector geometry. This ray tracing limitation of simulation is shared with the synthesis of realistic images in computer graphics. Many recent NVIDIA GPUs feature the RTX platform including hardware "RT Cores" dedicated to the acceleration of ray geometry intersection. NVIDIA claims performance of more than 10 billion ray intersections per second, which is a factor 10 more than possible with earlier GPUs which perform the intersection acceleration in software.

1.3 NVIDIA® OptiX™ ray tracing engine

OptiX is a general-purpose ray tracing engine designed for NVIDIA GPUs that exposes an accessible single ray programming model. The core of OptiX is a domain-specific just-in-time compiler that constructs ray tracing pipelines, combining code for acceleration structure creation and traversal, together with user provided CUDA code for ray generation, object intersection and closest hit handling. Spatial index data structures, such as the boundary volume hierarchy (BVH), are the principal technique for accelerating ray geometry intersection. OptiX provides only the acceleration of ray geometry intersection, not the intersection itself, thus affording full flexibility to implement intersections with any form of geometry. OptiX acceleration structures supports instancing, allowing them to be shared between multiple placements of the same geometry such as the photomultiplier tubes in the JUNO geometry.

In August 2019 NVIDIA introduced OptiX 7, an entirely new much lower level CUDA-centric API[13]. The level of abstraction of the new API is similar to that of two other ray tracing extension APIs for Vulkan: VKRay and Microsoft's DirectX: DXR. All three ray tracing APIs provide access via the NVIDIA driver to the same underlying NVIDIA ray tracing technology, including construction of acceleration structures and access to ray trace dedicated RT Core hardware. The three APIs for geometry and instance acceleration structure creation and preparation of shader context have many similarities. The low level convergence of these three APIs is suggestive that further major OptiX API transitions are unlikely.

The OptiX 7 API has minimal host state, all host functions are thread safe and many operations are asynchronous using CUDA streams. The API gives explicit control over memory management and shader compilation to the application. The new API gives more control and flexibility but shifts the implementation burden for important features such as multi-GPU scaling to the application.

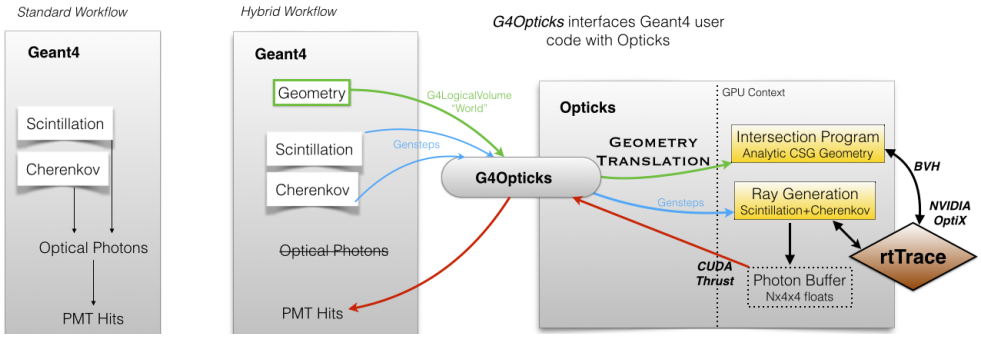

Figure 2. Comparison of the standard workflow of Geant4 optical photon simulation (left) with the hybrid Geant4 + Opticks workflow (right). A single Opticks class `G4Opticks` acts to interface Geant4 user code with the Opticks GPU propagation. Hybrid simulation requires modification of the classes representing scintillation and Cherenkov processes to collect "genstep" data structures.

2 Hybrid simulation workflow

Implementing an efficient GPU optical photon simulation equivalent to the Geant4 simulation requires that all aspects of the Geant4 context relevant to optical photon generation and propagation are translated into an appropriate form and uploaded to the GPU. The primary aspects are the detector geometry including material/surface properties, optical physics and optical photons; the translations of these are described in the below sections.

Figure 2 provides an overview of the hybrid simulation workflow. A single class, `G4Opticks`, is used to provide a minimal interface between Geant4 user code and the Opticks package. At initialization the Geant4 top volume pointer is passed to Opticks which translates the geometry and constructs the OptiX GPU context. The hybrid workflow replaces the generation of photon secondary tracks in a loop with the collection of "genstep" parameters including the number of photons to generate and the line segment along which to generate them and all other parameters needed for the generation. These gensteps together with CUDA ports of the Cherenkov and scintillation generation allow the photons to be generated directly on the GPU within the ray generation program provided to OptiX. This avoids allocation of CPU memory for the photons, only photon hits necessary for the next stage electronics simulation require CPU memory allocation. The genstep arrays are typically several orders of magnitude smaller than the photon arrays that are generated from them. Gensteps are valid only for specific versions of the implementation of the processes as they must be used only with matched CUDA ports of the generation.

Steering of the simulation is implemented in the ray generation program, which performs parallel photon generation and propagation up to a configurable maximum number of steps. For each step of the propagation, rays representing photons are intersected with the geometry. The intersected boundary provides a boundary index which allows material properties such as absorption and scattering lengths to be looked up. Converting these lengths to distances using pseudorandom numbers and the known exponential distributions allows a comparison of absorption and scattering distances with geometrical distance to boundary to assign photon histories. Details on efficient use of pseudorandom number generation with cuRAND[18] are in the earlier proceedings[6].

3 Detector geometry

Detector geometry is modelled on the GPU with OptiX intersection, bounding box and closest hit programs and buffers that these programs access. Opticks provides automated translation of Geant4 geometries first into the Opticks `GGeo` geometry and then into OptiX buffers. The translation starts by traversing the Geant4 volume tree converting materials, surfaces, solids, volumes and sensors into the Opticks geometry model. The Opticks geometry model is complete in the sense that it does not depend on Geant4 and it also provides a serialization in the form of NumPy[19] binary files. For large detector geometries with many thousands of volumes, such as the JUNO geometry, this translation can take several minutes. To avoid repeating this processing the serialization is used to persist the geometry into a "geocache" directory structure. On subsequent runs, the NumPy binary files are loaded and uploaded to the GPU, allowing simulation and visualization to initialize full geometries in seconds rather than minutes.

3.1 Material and surface properties

Material and surface properties as a function of wavelength are interpolated onto a common wavelength domain. The properties include refractive indices, absorption lengths, scattering lengths, reemission probabilities, as well as surface reflectivities, detection efficiencies and absorption fractions. Each volume of the geometry is assigned a boundary index uniquely identifying the combination of four indices representing outer and inner materials and outer and inner surfaces. Outer/inner surfaces handle inwards/outwards going photons, which allows the Geant4 border and skin surface functionality to be translated. Surfaces with a non-zero efficiency property are used to identify sensor volumes.

GPUs contain hardware dedicated to fast texture lookup and interpolation. This is exploited by using a single 2D `float4` texture, named the boundary texture, that contains interleaved material and surface properties as a function of wavelength for all unique boundaries. The boundary index returned from a ray traced primitive intersection enables four wavelength interpolated material or surface properties to be obtained from a single hardware optimized texture lookup.

3.2 Solid shapes

Opticks provides CUDA functions that return ray intersections for ten primitive shapes including sphere, hyperboloid and torus. These functions use implicit equations for the primitives together with the parametric ray equation, to yield a polynomial in t, the distance along the ray from its origin position. Roots and derivatives yield intersections and surface normals. Arbitrarily complex solids are described using constructive solid geometry (CSG) modelling, which builds shapes from the combination of primitive constituents by boolean set operations and is represented with a binary tree data structure. Each primitive or operator node is serialized into an array of up to 16 elements. These elements include float parameters of the primitives and integer index references into a separate transform array. A complete binary tree serialization with array indices matching level order tree indices and zeros at missing nodes is used for the serialization of the CSG trees. This simple serialization allows tree navigation directly from bitwise manipulations of the serialized array index. Complete binary tree serialization is simple and effective for small trees but very inefficient for large and unbalanced trees, necessitating tree balancing for shapes with many constituent primitives to reduce the tree height. The prior proceedings[5] provide further details of the constructive solid geometry modelling, tree balancing and translation between Geant4 and Opticks solids.

3.3 Structural volumes

The Geant4 structural geometry model comprises a containment tree hierarchy of volumes with associated transforms. The Opticks geometry model is based upon the observation that many of these volumes are repeated in groups, corresponding for example to the small number of volumes that represent each type of PMT, and thus an efficient representation must make full use of geometry instancing. Geometry instancing is a technique used in graphics libraries including OpenGL and NVIDIA OptiX that avoids duplication of information on the GPU by storing repeated elements only once together with 4x4 transform matrices that specify the locations and orientations of each instance.

The translation between the Geant4 and Opticks geometry models starts by creating a parallel tree of Opticks `GVolume` nodes. Each `GVolume` holds references to both analytic `GParts` and triangulated `GMesh` objects. The JUNO geometry tree contains more than 300,000 volumes. This geometry information is effectively factorized into about 10 `GMergedMesh` objects by the `GInstancer` class by first assigning a geometry digest string to every `GVolume` node based on the transforms and shape indices of the progeny nodes of the full subtree descended from it. Subsequently repeated groups of volumes and their placement transforms are identified using the progeny digests, after disqualifying repeats that are contained within other repeats or which are insufficiently repeated. With the digests of the repeats identified it is then straightforward to label all nodes with a repeat index, leaving remainder insufficiently repeated nodes with a repeat index of zero.

The labelled tree then allows `GMergedMesh` objects for each repeat index, including the remainder volumes, to be created, collecting the many thousands of transforms for all placements of for example PMTs and combining together the analytic `GParts` and triangulated `GMesh` into composite objects for each repeated group. Prior to combination both analytic and triangulated data has a placement transform applied. For the repeated instance volumes the placement transform is relative to the instance base whereas for the non-repeated remainder volumes global transforms are used for the placement.

3.4 Translation of Opticks geometry to OptiX and OpenGL models

The small number of `GMergedMesh` objects correspond to repeated assemblies of groups of volumes which bring together the structural information of the instance transforms and the solid shape information in the composite analytic `GParts` and triangulated `GMesh` objects.

Each composite `GParts` object combines multiple constructive solid geometry (CSG) node trees containing concatenated arrays of constituent CSG nodes, transforms, planes and identity information that are inputs to the analytic CSG intersection and bounding box programs. These programs and instance transforms are linked together into the OptiX geometry node graph which is also used to configure the resulting acceleration structures. Figure 3 presents a ray traced rendering of the analytic representation of the JUNO detector geometry.

Analogously each composite `GMesh` object combines arrays of triangle vertices and indices from the Geant4 polygonization of the solids of each of the volumes which are used as inputs together with the arrays of instance transforms to the OpenGL rasterized visualization using a single draw call for each `GMergedMesh`.

3.5 Sensor efficiencies

Photon detection on sensors such as PMTs is typically modelled by stochastic culling a fraction of candidate hits based on overall and angle dependent sensor collection efficiencies. With the hybrid CPU-GPU workflow used by Opticks it is highly beneficial to perform this

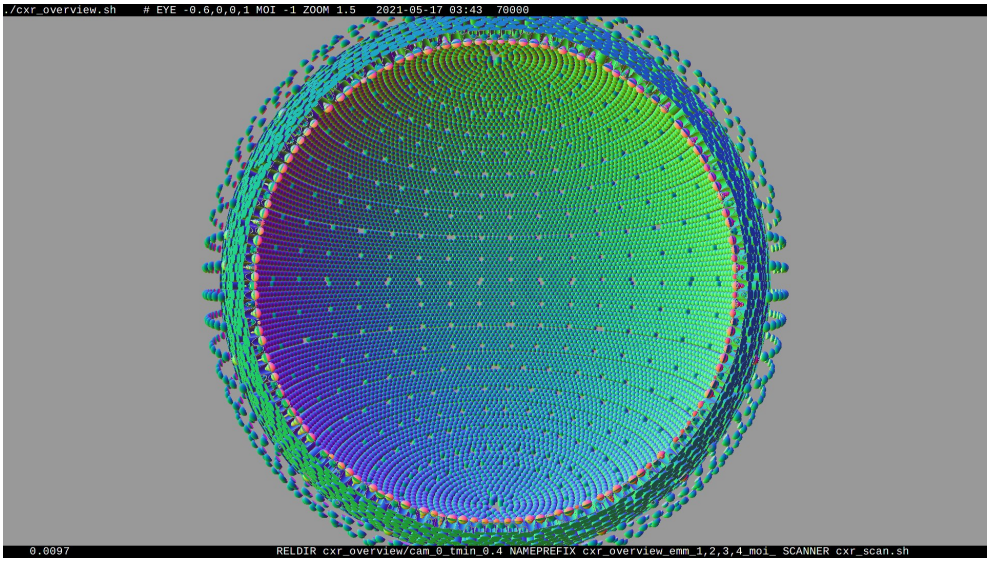

Figure 3. Render of the PMTs of the JUNO detector comprising 1920x1080 ray traced pixels created with a CUDA launch of under a hundredth of a second using a single NVIDIA TITAN RTX GPU and NVIDIA OptiX 7. The geometry is modelled using a shared CPU/GPU geometry model[21] designed to work with the NVIDIA OptiX 7 API, converted with CSG_GGeo[22] and rendered using CSGOptiX[23].

culling on the GPU as then the CPU memory necessary for hits can be reduced by the culling fraction and also the sizes of GPU to CPU copies are correspondingly reduced. In order to implement this it is necessary for overall and angle dependent collection efficiences for all sensors to be collected and uploaded to the GPU at initialization. The `G4Opticks` interface provides methods to collect efficiencies and sensor category indices for all sensors as well as efficiencies as a function of the sensor local frame spherical coordinate angles for each category of sensor.

During the recursive traversal of the Geant4 geometry tree, that creates the parallel tree of Opticks `GVolume`, sensor volumes are identified and sensor indices assigned by the presence of surfaces with non-zero efficiencies. These automatically assigned sensor indices and vectors of the corresponding Geant4 `G4PVPlacement` volumes allow sensor efficiency data to be handled in a detector independent manner. In addition to sensor efficiency values each sensor can be associated with sensor categories and detector specific sensor identifiers. The sensor categories correspond to different types of sensors with different angular efficiencies. Separate GPU $\theta - \phi$ textures for each sensor category are used to provide interpolated angular efficiency lookups from the sensor local frame coordinates of hits. In this way every point on the surfaces of all sensor volumes is assigned an efficiency. Comparison of the efficiency for each sensor hit with a pseudorandom number is used to assign *collect* or *cull* photon flags. These flags are then used to control the selection of photons that are copied back to the CPU using the stream compaction implementation provided by CUDA Thrust[20] methods `thrust::count_if` and `thrust::copy_if` used by Opticks `TBuf`.

4 Optical physics

Optical physics processes of scattering, absorption, scintillator reemission and boundary processes are implemented in CUDA functions based on the Geant4 implementations. The single ray programming model of NVIDIA OptiX enables direct ports of the corresponding Geant4 implementations adapted to use GPU textures for property access. On the CPU, it is convenient to implement scintillator reemission using Geant4 secondary tracks. A different approach is adopted on the GPU where a fraction of absorbed photons are reborn with modified direction and wavelength within the same CUDA thread. A reemission texture that encapsulates an inverse cumulative distribution function is used to generate wavelengths that follow the desired distribution on lookup of pseudorandom numbers. Further details on the CUDA ports of Geant4 optical physics and use of GPU textures are given in the 2016 proceedings[6].

5 Random number aligned comparison of Opticks and Geant4

Validation comparisons use a single executable that performs both the Geant4 and hybrid Opticks simulations, starting from common CPU generated input photons. Copying cuRAND random sequences from the GPU to the CPU and configuring the Geant4 random engine to use them makes it possible to align the consumption of random numbers between the two simulations, resulting in nearly perfectly matched results with every scatter, absorption and reflection happening with the same positions, times, wavelengths and polarizations. Direct comparison of the aligned simulation results allows any discrepancies to be identified without being clouded by statistics. The executable writes two events in a format which includes highly compressed positions, times, wavelengths and polarizations at up to 16 steps of the optical photon propagations.

Checks of all JUNO solids with millions of photons revealed some spurious intersects arising from fragile CSG modelling with constituent solids that had coincident faces. These were fixed by straightforward modelling changes to avoid coincidences. Shapes including the torus as a constituent were found to be prone to poor precision intersects. As the use of torus was cosmetic, the modelling was simplified to avoid its use. After fixing these geometry issues, remaining discrepancies in mis-aligned photon histories were <0.25% and deviant photons within matched histories were <0.05%. Primary sources of discrepancies are photons with grazing incidence or photons incident at constituent solid boundaries. Results in these cases are expected to depend on the arithmetic precision: Opticks uses double precision only where unavoidable, whereas Geant4 uses this everywhere.

6 Performance comparisons

Optical photon simulation performance with the full analytic JUNO geometry is measured using calibration source gensteps, positioned at the center of the scintillator volume, that uniformly emit a range of photon counts. The maximum number of optical photons that can be simulated in a single GPU launch is limited by the available VRAM. Each photon requires 64 bytes for parameters and 48 bytes for the cuRAND random number generator state, corresponding to 45G for 400M photons. The measurements use a production mode with only photomultiplier hits being stored. All non-essential processing such as photon step recording are skipped. Test hardware was a single NVIDIA Quadro RTX 8000 GPU with 48G of VRAM hosted in a DELL Precision 7920T workstation with Intel Xeon Gold 5118, 2.3GHz, 62G CPU. Figure 4 shows results from a scan from 1M to 400M optical photons, where the measured Opticks times are compared with Geant4 times linearly extrapolated from a measurement at 1M photons. Comparing times with the RTX mode enabled and

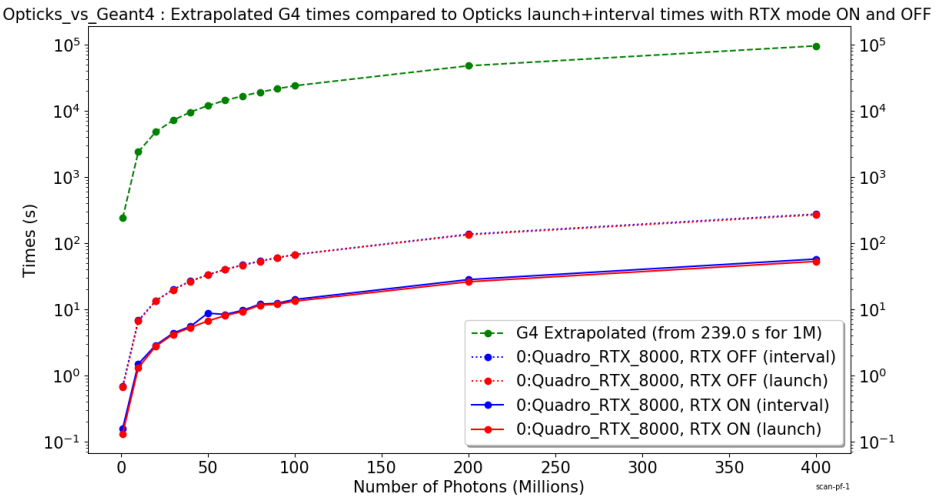

Figure 4. Full analytic JUNO geometry Opticks simulation times in seconds for 1M-400M optical photons using a single NVIDIA Quadro RTX 8000 GPU compared to single threaded Geant4 10.4.2 simulation times extrapolated from a measurement for 1M optical photons. The solid(dotted) blue and red curves show times with RTX enabled(disabled). Differences between interval times which include per event upload and download overheads and launch times are not readily apparent with the logarithmic scale. The linearly extrapolated Geant4 time for 400M photons is 95,600 s (26 hours) contrasts with the Opticks time of 58s, corresponding to a speedup factor of 1,660 times with a single GPU.

disabled indicates a speedup of approximately 5 times from the use of the ray trace dedicated RT cores. The single GPU speedup factor between Opticks with RTX enabled and single threaded Geant4 is measured to be 1,660.

Performance measurements with very simple analytic geometries are found to reach Giga Rays/s, more than a factor of 10 faster than performance with the full JUNO analytic geometry. This great performance sensitivity to the geometry suggests there is potential for substantial improvement by optimization of geometry modelling.

7 Summary

Opticks enables Geant4-based optical photon simulations to benefit from state-of-the-art NVIDIA GPU ray tracing, made accessible via NVIDIA OptiX, that allows memory and time processing bottlenecks to be eliminated. Recent developments enable Opticks to greatly reduce the CPU memory for hits by moving collection efficiency hit culling to the GPU. Opticks meets the challenge of optical photon simulation in JUNO, the world's largest scintillator detector, and can benefit any simulation limited by optical photons.

Several groups from various experiments and the Geant4 Collaboration are evaluating Opticks. Physicists from the LZ dark matter experiment and LBNL suggested and organized a series of meetings with NVIDIA engineers that have assisted with the migration of Opticks to the all new NVIDIA OptiX 7 API. Figure.3 is one of the first renders of the full JUNO geometry with OptiX 7.

Acknowledgements

The JUNO collaboration is acknowledged for the use of detector geometries and simulation software. Dr. Tao Lin is acknowledged for his assistance with the JUNO offline software. This work is funded by Chinese Academy of Sciences President's International Fellowship Initiative, Grant No. 2018VMB0002.

References

[1] Opticks Repository, https://bitbucket.org/simoncblyth/opticks/

[2] Opticks References, https://simoncblyth.bitbucket.io

[3] Opticks Group, https://groups.io/g/opticks

[4] S. Blyth, EPJ Web Conf. **245**, 11003 (2020)
https://doi.org/10.1051/epjconf/202024511003

[5] S. Blyth, EPJ Web Conf. **214**, 02027 (2019)
https://doi.org/10.1051/epjconf/201921402027

[6] Blyth Simon C 2017 J. Phys.: Conf. Ser. **898** 042001
https://doi.org/10.1088/1742-6596/898/4/042001

[7] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce et al., Nucl. Instrum. Methods. Phys. Res. A **506**, 250 (2003)

[8] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Dubois, M. Asai et al., IEEE Trans Nucl Sci, **53**, 270 (2006)

[9] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso et al., Nucl. Instrum. Methods. Phys. Res. A **835**, 186 (2016)

[10] OptiX: a general purpose ray tracing engine
S. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock et al., ACM Trans. Graph.: Conf. Series **29**, 66 (2010)

[11] OptiX introduction, https://developer.nvidia.com/optix

[12] OptiX API, http://raytracing-docs.nvidia.com/optix/index.html

[13] OptiX 7 https://developer.nvidia.com/blog/how-to-get-started-with-optix-7/

[14] The Application of SNiPER to the JUNO Simulation,
T. Lin et al., J.Phys.Conf.Ser. **898** 042029 (2017)
https://doi.org/10.1088/1742-6596/898/4/042029

[15] Neutrino physics with JUNO
F. An et al., J. Phys. G. **43**, 030401 (2016)

[16] NVIDIA RTX, https://developer.nvidia.com/rtx

[17] Understanding Throughput Oriented Architectures
M. Garland, D.B. Kirk, Commun. ACM **53**(11), 58 (2010)

[18] cuRAND, http://docs.nvidia.com/cuda/curand/index.html

[19] The NumPy array: a structure for efficient numerical computation
S. Van der Walt, S. Colbert, G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011)

[20] Chapter 26 - Thrust: A Productivity-Oriented Library for CUDA
N. Bell, J. Hoberock, GPU Computing Gems Jade Edition, (2012), pp 359-371

[21] Shared GPU/CPU "CSGFoundry" geometry model,
https://github.com/simoncblyth/CSG

[22] Converter of Opticks/GGeo geometry to "CSGFoundry" model,
https://github.com/simoncblyth/CSG_GGeo

[23] NVIDIA OptiX 7 and pre-7 renderer of "CSGFoundry" geometry,
https://github.com/simoncblyth/CSGOptiX