EPJ Web of Conferences **295**, 11014 (2024) https://doi.org/10.1051/epjconf/202429511014
CHEP 2023

Opticks : GPU Optical Photon Simulation via NVIDIA OptiX

Simon C. 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 GPU ray tracing, accessed via the NVIDIA OptiX 7+ API, with Geant4 based simulations. A single NVIDIA Turing GPU from the first RTX generation 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 in CUDA programs based on Geant4. 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 the near complete re-implementation required to adopt the entirely new NVIDIA OptiX 7+ API, with the implementation now directly CUDA based with OptiX usage restricted to providing intersects. The re-implementation features a modular many small header design that enables fine grained testing both on GPU and CPU together with large code reductions from CPU/GPU sharing. Enhanced modularity has enabled CSG tree generalization to support "list-nodes", similar to G4MultiUnion, that improve performance for complex CSG solids. In addition support for interference effects on boundaries with multiple thin layers, such as anti-reflection coatings and photocathodes, using CUDA compatible transfer matrix method (TMM) calculations of reflectance, transmittance and absorptance are reported.

1 Introduction

Opticks[1-7] enables Geant4[8-10] based simulations to benefit from high performance GPU ray tracing made accessible by NVIDIA® OptiX™[11-15]. The Jiangmen Underground Neutrino Observatory (JUNO)[16] 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. Sequential simulation of large numbers of optical photons has extreme computational and memory costs. Opticks enables drastically improved optical photon simulation performance. Although developed for simulation of the JUNO detector, Opticks supports use with other detector geometries. Any optical photon limited simulation can benefit from Opticks.

*Corresponding author e-mail: simon.c.blyth@gmail.com.

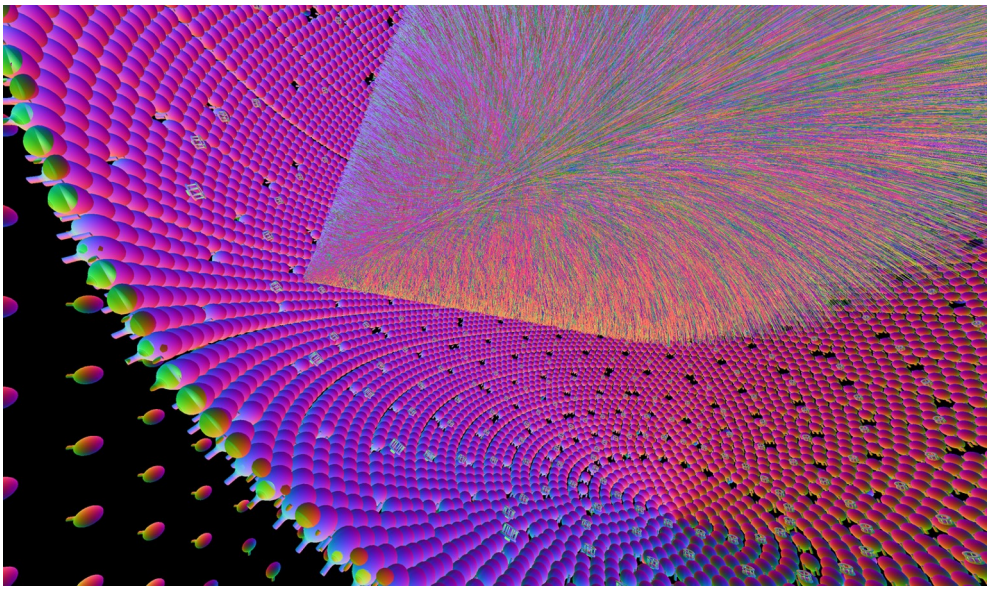

Figure 1. Cutaway OpenGL rendering of millions of simulated optical photons from a 200 GeV muon crossing the JUNO liquid scintillator. Each line corresponds to a single photon with line colors representing the polarization direction. Primary particles are simulated by Geant4, "gensteps" are uploaded to the GPU and photons are generated, propagated and visualized all on the GPU.

Opticks was presented to the four prior CHEP conferences, with each contribution covering different aspects of its development. The 2021 contribution[4] covered initial stages of the transition to the NVIDIA OptiX 7+ API and the integration of Opticks with detector simulation frameworks. The 2019 plenary presentation and proceedings[5] focused on RTX[15] performance measurements. The earlier contributions[6,7] covered the first implementations of geometry translation and the CUDA port of photon generation and optical physics.

These proceedings describe the almost completely re-implemented Opticks, as required to adopt the entirely new NVIDIA OptiX 7+ API. Ongoing developments to improve performance with complex solid shapes with "list-nodes" are described and modelling of multiple thin layer interference effects are reported.

1.1 Importance of optical photon simulation

Suppression of cosmic muon induced backgrounds with veto selections are crucial for neutrino detectors such as JUNO[16], necessitating production of large simulated samples of muon events. However, a muon of typical energy 200 GeV crossing the JUNO scintillator can yield tens of millions of optical photons, which are found with Geant4 simulations to consume more than 99% of CPU time and impose severe memory constraints. As optical photons in neutrino detectors can be considered to be produced only by scintillation and Cherenkov processes and yield only hits on photomultiplier tubes, it is straightforward to combine an external optical photon simulation with a Geant4 simulation of all other particles.

1.2 GPU ray tracing

GPUs evolved to perform rasterized rendering, optimizing throughput[19] rather than minimizing latency. GPUs are suited to problems with millions of independent low resource

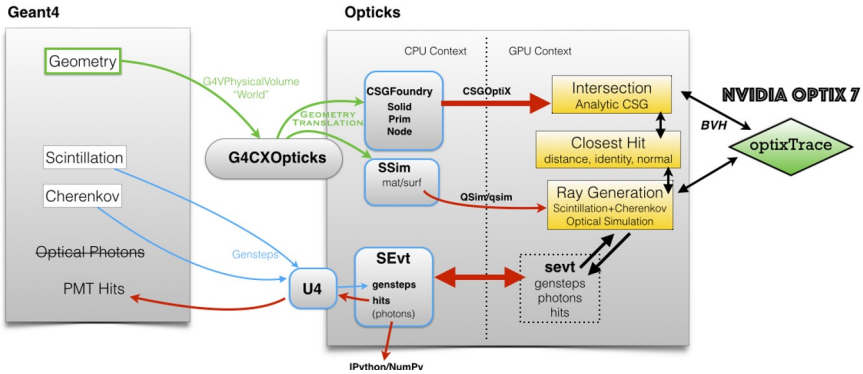

Figure 2. Hybrid Geant4 + Opticks workflow : `G4CXOpticks` translates Geant4 geometry to GPU appropriate form. `U4` collects "gensteps" enabling GPU generation of scintillation and Cerenkov photons.

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 include 3rd generation RTX[15] hardware dedicated to ray geometry intersection. NVIDIA GPU ray tracing performance continues to improve rapidly, with performance doubling with each RTX generation.

1.3 NVIDIA® OptiX™ ray tracing engine

OptiX makes GPU ray tracing accessible with a single ray programming model. Ray tracing pipelines are constructed combining code for acceleration structure traversal, with user code for ray generation, intersection and closest hit handling. Spatial index acceleration structures (AS) provide accelerated ray geometry intersection. OptiX provides only acceleration, not the intersection itself, thus allowing any form of geometry to be implemented. In August 2019 NVIDIA introduced the OptiX 7 API[14], that together with the Vulkan and DirectX ray tracing extensions provides access to the same NVIDIA ray tracing technology including AS construction and RTX hardware access.

2 Hybrid simulation workflow

Figure 2 summarizes the hybrid workflow. At initialization the Geant4 top volume is passed to Opticks which translates the geometry and uploads it to the GPU as described in section 3. Geant4 models scintillation and Cerenkov processes with the classes `G4Scintillation` and `G4Cerenkov`. At each simulated step of applicable particles the classes calculate a number of optical photons to generate depending on particle and material properties, followed by a loop that generates the optical photons. With the hybrid workflow these classes are modified, replacing the generation loop with the collection of generation parameters termed "gensteps" that include the number of photons and the line segment along which to generate them and all other parameters needed to reproduce photon generation on the GPU. Relocating photon generation to the GPU avoids CPU memory allocation. Only non-culled photon hits needed for the next stage electronics simulation require CPU memory allocation.

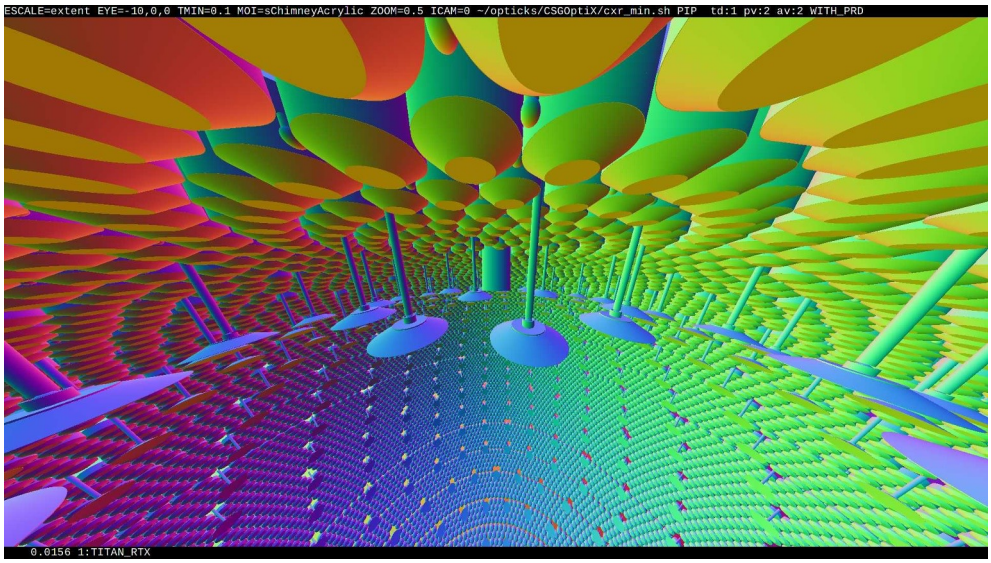

Figure 3. Render of the PMTs of the JUNO detector comprising 1920x1080 ray traced pixels created with a CUDA launch under 16 ms using a single NVIDIA TITAN RTX GPU and NVIDIA OptiX 7.

GPU optical photon generation and propagation are implemented in simple headers that are included into the OptiX ray tracing pipeline that runs in parallel. For each step of the propagation, rays representing photons are intersected with the geometry using simple header intersect functions that are also included into the ray tracing pipeline. The intersected boundary together with the photon wavelength are used to do interpolated texture lookups of material properties such as absorption and scattering lengths. Converting these lengths to distances using pseudorandom numbers and the known exponential distributions allows a comparison of absorption and scattering distances with geometrical boundary distance to assign photon histories. Earlier proceedings[7] detail efficient use of the cuRAND[20] pseudorandom generator.

3 Detector geometry

At initialization Opticks translates between geometry models in the below sequence:

1. Geant4 : deep hierarchy of structural volumes and trees of `G4VSolid` CSG nodes
2. Opticks `stree` : intermediate n-ary trees of volumes and CSG nodes
3. Opticks `CSGFoundry` : GPU model with `CSGSolid`, `CSGPrim` and `CSGNode`
4. NVIDIA OptiX 7+ : Instance and Geometry Acceleration Structures (IAS, GAS)

The `stree` intermediate model is created from the Geant4 model by `U4Tree.h`, which traverses the Geant4 volume tree converting materials, surfaces, solids, volumes and sensors. Subsequently the `CSGFoundry` model is created from the intermediate model and uploaded to GPU. The intermediary provides a complete representation of the needed geometry information implemented in a minimal way using only a handful of structs, very different from the heavyweight `GGeo` model that it replaces. The geometry information formerly managed with a large number of classes becomes in the `stree` model separate directories managed by

a single header only struct, `NPFold.h`, which provides an in memory directory tree of arrays using a recursive folders of folders of arrays data structure. The complete structural n-ary tree of volumes and n-ary CSG trees for each `G4VSolid` are serialized into arrays using first child and sibling references.

Both the `stree` and `CSGFoundry` geometry models are independent of Geant4 and can be persisted into directories of NumPy[21] binary files. Fast binary file loading and uploading to GPU allows optical simulation and visualization executables to initialize full detector geometries in less than a second.

3.1 Structural volumes and geometry factorization

The Geant4 model of the JUNO geometry contains almost 400,000 structural volumes organized in a deep containment tree heirarchy of volumes with associated transforms. The Opticks approach to modelling these volumes is based upon the observation that many of the volumes are repeated in groups, corresponding for example to the small number of volumes that represent each type of PMT. Hence an efficient representation must make full use of geometry instancing to avoid duplication of information on the GPU by storing repeated elements only once together with 4x4 transforms that specify the locations and orientations of each instance.

The factorization of the volumes into repeated groups of volumes and a remainder of other insufficiently repeated volumes is done within the intermediate model by the `stree::factorize` method which uses sub-tree digests that represent the geometry and transforms beneath every node of the geometry tree. The factorization implemented in a minimal way reproduces the results of the former heavyweight implementation spread across many classes. The outcome of the factorization is a repeat index on every structural node allowing the instance transforms of each repeat to be collected.

Each of the factors from the intermediate model become compound `CSGSolid` within the `CSGFoundry` model. For the JUNO geometry the factorization yields ten compound solids including four different types of PMT assemblies, some of which are repeated many thousands of times. The instanced `CSGSolid` typically contain a few `CSGPrim`, for example PMT masks and Pyrex and Vacuum volumes of the PMT, and the remainder solid contains several thousand `CSGPrim`. The `CSGPrim` refer to sequences of `CSGNode` which are one-to-one related to the `sn.h` CSG nodes of the intermediate model. These CSG nodes typically correspond to constituent `G4VSolid` such as `G4Ellipsoid` and `G4Tubs`.

3.2 CSGFoundry geometry model

Table 1 summarizes the role and associations of the vector members of the `CSGFoundry` struct which are serialized and uploaded to the GPU forming the inputs to the creation of acceleration structures and also the CSG node parameters used by the CSG intersection functions. The `CSGFoundry` model is designed to facilitate creation of OptiX acceleration structures by the `CSGOptiX` package. Geometry acceleration structures (GAS) are created from each of the compound `CSGSolid` objects and a single instance acceleration structure (IAS) for the entire geometry is created from the instance transform vector.

3.3 Solid shapes

The `stree` intermediate model carries solid shape information within an n-ary tree of `sn.h` CSG nodes. The Opticks CSG package implements ray primitive shape intersection in simple headers that are CUDA compatible but can also be used and debugged on the CPU. These

struct, member	Associations and role	Geant4 Equivalent
`qat4` inst	instance transform, references `CSGSolid`	None
`qat4` tran	CSG transform, referenced from `CSGSolid`	None
`CSGSolid` solid	references sequence of `CSGPrim`	Group of volumes
`CSGPrim` prim	references sequence of `CSGNode`, bounding box	root `G4VSolid`
`CSGNode` node	references CSG transform, node parameters, typecode	constituent `G4VSolid`

Table 1. Principal std::vector members of `CSGFoundry` struct with Geant4 equivalent.

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 with the large trees that result from complex shapes with many constituent primitives. Prior proceedings[6] describe the use of tree balancing to reduce the tree depth and avoid poor performance with complex solids. However studies have revealed that tree balancing is not compatible with the CSG intersection algorithm currently used by Opticks with some complex shapes.

The Opticks CSG intersection implementation has been generalized into three levels : tree, node and leaf. This generalization allows the node level to support compound multi-leaf shapes without resorting to recursion in the intersect function, which is disallowed in OptiX pipelines. As multi-leaf nodes can be used within binary CSG trees it becomes possible for complex solids to be represented by binary trees of greatly reduced depth avoiding deep tree performance issues. The multi-leaf nodes are similar to `G4MultiUnion`, but are restricted to leaf constituents. Various types multi-leaf nodes are under development including dis-contiguous, contiguous and overlap nodes. Communicating the intent of the multi-leaf nodes allows use of better suited intersect algorithms. For example, a common cause of deep CSG trees is the subtraction of a union of many "holes" from a base shape. Typically the holes do not overlap making it possible to use a simple low resource dis-contiguous intersect function providing more efficient intersection thanks to both reduced tree height and the better communication of the dis-contiguous nature of the holes.

3.4 Multi-layer thin film interference effects

The JUNO PMT optical model[17] accounts for optical processes inside PMTs including interference effects from thin layers of anti-reflection coating and photocathode between the PMT glass and interior vacuum. The optical model is implemented using a transfer matrix

Role	CPU	GPU
simulation steering	QSim.hh	qsim.h
curandState setup	QRng.hh	qrng.h
property interpolation	QProp.hh	qprop.h
event handling	QEvent.hh	qevent.h
Cerenkov generation	QCerenkov.hh	qcerenkov.h
scintillation generation	QScint.hh	qscint.h
texture handling	QTex.hh	cudaTextureObject_t

Table 2. Simple host/device counterpart pattern adopted by the Opticks QUDARap package.

method (TMM) calculation of reflectance, transmittance and absorption based on layer thicknesses and complex refractive indices of the layer materials.

In order to optionally perform equivalent calculations within Opticks a CPU/GPU single header `C4MultiLayrStack.h` implementation of TMM has been developed within the Custom4 package[18]. Custom4 is an optional external package that when available is integrated with the Opticks QUDARap package providing special optical surface handling that performs the interference calculations.

4 Optical physics

Opticks optical photon simulation is now implemented in the QUDARap package, that depends only on the SysRap base package and CUDA, with no dependency on OptiX. Table 2 illustrates the simple host/device counterpart pattern adopted where each CPU struct sets up its GPU counterpart first on the host, uploading constituents and setting device pointers within a host instance prior to uploading it to the GPU. This approach simplifies GPU/CPU testing of the simulation.

5 Summary

Opticks enables Geant4-based optical photon simulations to benefit from state-of-the-art NVIDIA GPU ray tracing, made accessible via the NVIDIA OptiX 7+ API, allowing memory and time processing bottlenecks to be eliminated. A full re-implementation of Opticks for the OptiX 7+ API is complete. Opticks now features many shared CPU/GPU headers and a minimal intermediate geometry model that has enabled drastic code reduction and simplification. The many small headers design together with mocking of a CUDA texture and random number functions enables fine grained testing of almost all functionality on both GPU and CPU.

Several groups from various experiments and the Geant4 Collaboration are evaluating Opticks and an example of Opticks usage is now included within the Geant4 distribution.

Acknowledgements

The JUNO collaboration is acknowledged for the use of detector geometries and simulation software. Dr. Tao Lin is acknowledged for presenting this work at conference at short notice and for his assistance with the JUNO offline software. NVIDIA is acknowledged for assistance with the OptiX 7+ API transition provided during a series of meetings throughout 2021 and a UK GPU hackathon in 2022. The LZ and LHCb physicists that suggested and organized these meetings with NVIDIA are also acknowledged.

This work is supported by National Natural Science Foundation of China (NSFC) under grant No. 12275293.

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. **251**, 03009 (2021)
https://doi.org/10.1051/epjconf/202125103009
[5] S. Blyth, EPJ Web Conf. **245**, 11003 (2020)
https://doi.org/10.1051/epjconf/202024511003
[6] S. Blyth, EPJ Web Conf. **214**, 02027 (2019)
https://doi.org/10.1051/epjconf/201921402027
[7] Blyth Simon C 2017 J. Phys.: Conf. Ser. **898** 042001
https://doi.org/10.1088/1742-6596/898/4/042001
[8] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce et al., Nucl. Instrum. Methods. Phys. Res. A **506**, 250 (2003)
[9] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Dubois, M. Asai et al., IEEE Trans Nucl Sci, **53**, 270 (2006)
[10] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso et al., Nucl. Instrum. Methods. Phys. Res. A **835**, 186 (2016)
[11] 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)
[12] OptiX introduction, https://developer.nvidia.com/optix
[13] OptiX API, https://raytracing-docs.nvidia.com/optix8/index.html
[14] OptiX 7 https://developer.nvidia.com/blog/how-to-get-started-with-optix-7/
[15] NVIDIA RTX™ Platform, https://developer.nvidia.com/rtx
[16] Neutrino physics with JUNO
F. An et al., J. Phys. G. **43**, 030401 (2016)
[17] A new optical model for photomultiplier tube
Y. Wang, G. Cao, L. Wen, Y. Wang, Eur. Phys. J. C 82(4), 329 (2022).
https://doi.org/10.1140/epjc/s10052-022-10288-y
[18] Custom4 repository, https://github.com/simoncblyth/customgeant4
[19] Understanding Throughput Oriented Architectures
M. Garland, D.B. Kirk, Commun. ACM **53**(11), 58 (2010)
[20] cuRAND, http://docs.nvidia.com/cuda/curand/index.html
[21] The NumPy array: a structure for efficient numerical computation
S. Van der Walt, S. Colbert, G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011)