Opticks : GPU Optical Photon Simulation for Particle Physics using NVIDIA® OptiX™

Simon C Blyth

HEP Lab, Department of Physics, National Taiwan University, Taipei, Taiwan.

E-mail: simon.c.blyth@gmail.com

Abstract. Opticks is an open source project that integrates the NVIDIA OptiX GPU ray tracing engine with Geant4 toolkit based simulations. Massive parallelism brings drastic performance improvements with optical photon simulation speedup expected to exceed 1000 times Geant4 when using workstation GPUs. Optical photon simulation time becomes effectively zero compared to the rest of the simulation. Optical photons from scintillation and Cherenkov processes are allocated, generated and propagated entirely on the GPU, minimizing transfer overheads and allowing CPU memory usage to be restricted to optical photons that hit photomultiplier tubes or other photon detectors. Collecting hits into standard Geant4 hit collections then allows the rest of the simulation chain to proceed unmodified. Optical physics processes of scattering, absorption, scintillator reemission and boundary processes are implemented in 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. Geometry is provided to OptiX in the form of CUDA programs that return bounding boxes for each primitive and ray geometry intersection positions. Some critical parts of the geometry such as photomultiplier tubes have been implemented analytically with the remainder being tessellated. OptiX handles the creation and application of a choice of acceleration structures such as boundary volume hierarchies and the transparent use of multiple GPUs. OptiX supports interoperation with OpenGL and CUDA Thrust that has enabled unprecedented visualisations of photon propagations to be developed using OpenGL geometry shaders to provide interactive time scrubbing and CUDA Thrust photon indexing to enable interactive history selection.

1. Introduction

Newton's Opticks[1] first elucidated the optical physics that is central to the operation of a wide gamut of machines from medical imaging scanners to neutrino detectors. Drastically improved optical photon simulation performance can be transformative to the design, understanding and operation of such machines. Opticks[2] enables Geant4[3][4][5] based simulations to benefit from the high performance GPU ray tracing made accessible by NVIDIA® OptiX™ [6][7].

Cosmic muon induced processes are crucial backgrounds for neutrino detectors such as Daya Bay [8] and JUNO[9], necessitating underground sites, water shields and muon veto systems[10]. Minimizing the dead time and dead volume that results from applying a veto requires an understanding of the detector response to the muon. Large simulated samples of muon events are crucial in order to develop such an understanding and also to provide efficiency estimates that are critical inputs to physics analyses.

The number of optical photons estimated to be produced by a muon of typical energy 100 GeV crossing the Daya Bay detector is at the level of several millions, for JUNO the estimate is an order of magnitude larger. Profiling the Geant4 toolkit based Daya Bay simulation shows the propagation of optical photons to consume more than 95% of CPU time. In addition to the CPU time the memory requirements for handling many millions of photons pose severe constraints.

As optical photons in neutrino detectors can be considered to be produced by only the scintillation and Cerenkov 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. The uncoupled, highly parallel and rather simple nature of the optical physics that is sufficient to describe neutrino detectors makes optical photon propagation well suited to general purpose GPU computing techniques where high performance requires massive parallelism with minimal communication between threads and low register usage[11].

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. This is understandable as the large data structures that model the geometry contrast markedly with the simple structures used to model the optical physics.

Image synthesis algorithms in computer graphics can be divided into the two approaches of rasterization and ray tracing. Ray tracing casts rays from a viewpoint through image plane pixels out into the scene and recursively reflects and refracts rays with the geometry intersected to compute pixel values. Rasterization projects 3D object vertices onto the image plane. The closeness of the ray tracing algorithm to the optical physics of image formation makes it the preferred approach for the creation of realistic images. Ray traced image synthesis and optical photon simulation have much in common, they are both limited by the calculation of intersections between rays and geometry. Due to the many applications of realistic image synthesis in the advertising, design, film and games industries the calculation of ray geometry intersections continues to be extensively optimized by the computer graphics community with performance claims reaching several hundreds of millions of intersections per second[12][13].

2. Related Work

2.1. Chroma

Chroma[14][15] implements both GPU ray tracing and optical photon simulation using CUDA kernels that are launched from python scripts. Ray tracing performance comparisons between OptiX and a fork of Chroma[16] halted my work with Chroma. In addition to performance other limitations of Chroma include: no support for multiple GPUs, memory inefficient photon handling requiring the copying of photons from CPU to GPU and memory inefficient geometry handling due to lack of geometry instancing. Also the use of python makes integration with Geant4 and other C++ code inconvenient. The development of Opticks was made possible by the experience gained from working with Chroma.

2.2. VecGeom and GeantV

The GeantV[17] (Geant-Vector) project aims to develop an all-particle transport simulation package that is several times faster than Geant4. The principal approach investigated is the application of SIMD operations across vectors of multiple tracks collected from multiple events according to locality criteria, requiring development of vectorised geometry algorithms. The VecGeom[18] project aims to provide these algorithms and also to optimize scalar performance; CUDA implementations are also provided.

3. NVIDIA® OptiX™

OptiX[6] [7] is a general-purpose ray tracing engine designed for NVIDIA GPUs that exposes an accessible single ray programming model while still providing performance competitive with the very best manually optimized approaches[12][13]. OptiX ray tracing pipelines are constructed from a small set of user provided CUDA programs analogously to how OpenGL rasterization pipelines are constructed from GLSL shaders. The programs handle different aspects of the ray tracing workflow including: ray generation, object intersection and closest hit. The programs communicate via a ray payload data structure as well as GPU buffers and textures which are typically uploaded to the GPU at initialization. The programs are combined together with the geometry traversal functionality into a ray tracing kernel by a domain optimized Just-In-Time compiler.

The central data structure enabling fast ray tracing is a geometry index known as an acceleration structure which sorts primitives into spatial or object groups. This structure is used by the traversal algorithm to efficiently search for primitives that potentially intersect a given ray. Construction of GPU acceleration structures optimized for traversal performance and also build speed remain active areas of research in computer graphics. OptiX provides a simple interface to associate acceleration structures with different parts of a scene geometry but all the details of creation, updating and traversal are handled by OptiX internally using the geometrical information provided by the bounding box programs. Opticks currently uses the SBVH (Split boundary volume hierarchy) structure[19]. The OptiX acceleration structure system also supports instancing, allowing the sharing of both geometry information and acceleration structures among the instances.

The design of OptiX was informed by studies[12][13] comparing GPU simulator performance against measurements for various ray tracing algorithms. The GPU simulators provide hard upper bounds based on the native assembly operations for each algorithm, the SIMD width and work scheduling approach. OptiX uses a dynamically load balanced GPU execution model that enables performance to scale linearly with CUDA cores across multiple GPUs. This scaling linearity has been verified by ray trace measurements on a borrowed 4 GPU workstation (Tesla K40M) by progressively masking GPUs.

OptiX has been regularly updated, with algorithms added and improved in order to make the best use of the characteristics of each generation of NVIDIA GPUs. The Opticks use of OptiX ray tracing is expected to lead to improved performance with future GPU generations simply by updating to the lastest OptiX version.

4. Geometry Translation

Implementing an equivalent external OptiX based simulation demands that the Geant4 context of geometry, material and surface properties is translated into an appropriate form and uploaded to the GPU. This is achieved by first writing the tessellated Geant4 geometry into a COLLADA or Digital Asset Exchange (DAE) file using the G4DAE[20] exporter, which was implemented based upon the standard GDML exporter. COLLADA is a widely supported 3D file format allowing the geometry to be imported into many commercial and open source tools, some of which provide high performance visualisation. The extensibility of the XML based COLLADA file format was used to incorporate all material and surface properties as a function of wavelength. Opticks reads the G4DAE geometry file using a forked version[21] of the Assimp[22] asset importer that adds handling of the extra optical material and surface properties. The reconstructed volume tree is analysed to locate repeated geometry using digests of progeny transforms and mesh identities. The repeated subtrees and their transforms allow the use of OptiX and OpenGL geometry instancing which results in huge memory savings. Instanced sharing of OptiX acceleration structures was also configured. Geometry instancing was found to be essential with the JUNO geometry and improved performance with the Daya Bay geometry.

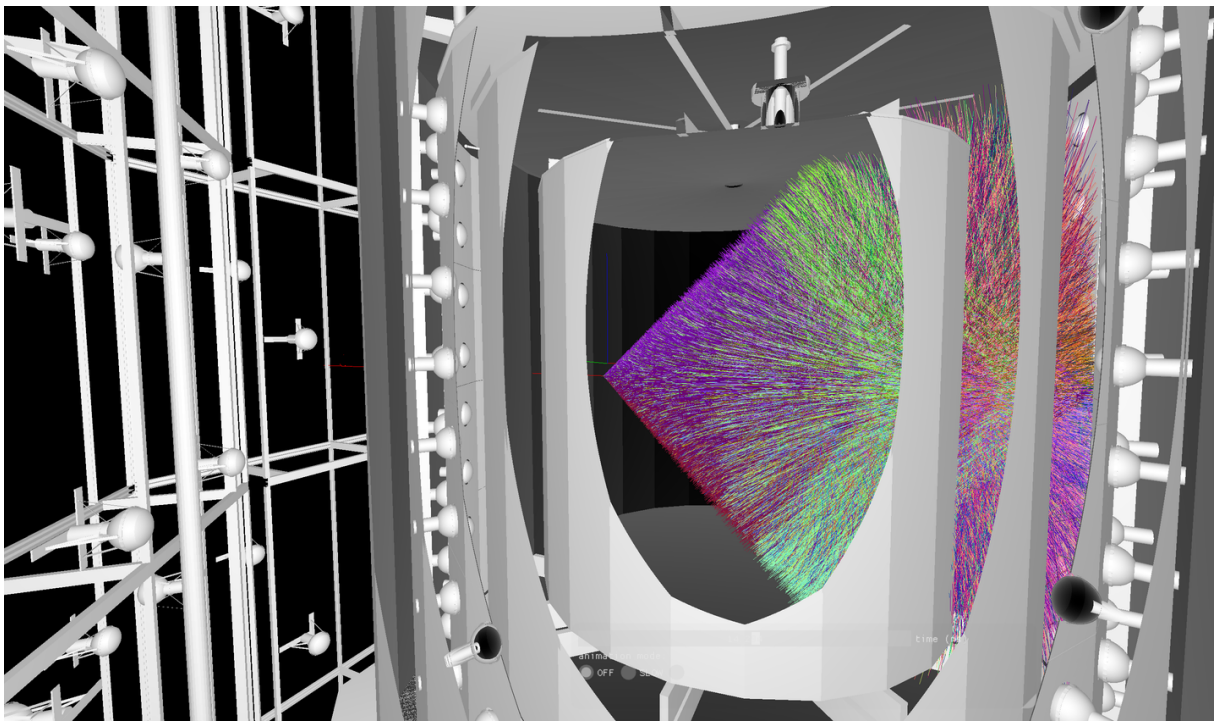

Figure 1. Simulated Cerenkov and scintillation photons from a 100 GeV muon travelling across the Daya Bay antineutrino detector. The screenshot image is composited from a ray traced rendering of the detector geometry with analytically defined photomultiplier tubes and an OpenGL rasterized representation of an optical photon propagation directly from OptiX written GPU buffers. The line colors represent the photon polarization direction. Opticks visualization provides interactive navigation and time scrubbing of the propagation on a Macbook Pro (2013) laptop with NVIDIA GeForce GT 750M Kepler generation GPU.

To avoid repeated parsing of XML at every initialization a geometry cache was implemented using the NPY[23] serialization format, reducing initialization times for the JUNO geometry from several minutes to several seconds. Multi dimensional arrays backed by an NPY format serialization, requiring only a header describing array type and dimension, are used throughout Opticks for the ease of uploading into GPU buffers, saving to file and loading as NumPy[25] arrays into ipython[24] for analysis.

Optical physics and ray tracing favors the use of a boundary based geometry model rather than the tree of volumes model used by Geant4. At initialization the volume tree is translated into a boundary based model with each primitive labelled with a boundary index which uniquely identifies a combination of four indices representing outer and inner materials and outer and inner surfaces. Outer/inner surfaces handle inwards/outwards going photons allowing the Geant4 border and skin surface functionality to be translated. 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 together with an orientation offset identified from the angle between the geometric normal and ray direction enables four wavelength interpolated material or surface properties to be obtained from a single hardware optimized texture lookup.

OptiX provides only the acceleration of geometrical intersection not the intersection itself. Opticks implements code to return bounding box and ray primitive intersections for a small

number of shapes such as triangles, spheres, cubes and cylinders. Use of tessellated geometry is the simplest approach as all triangle primitives can share the same intersection code, however some shapes are poorly modelled with tessellation. To improve realism for the optically critical photomultiplier tubes an analytic description was developed using the original constructive solid geometry (CSG) which defines solids using boolean set operations applied to basis shapes. Implementation of general CSG boolean handling was avoided by partitioning the geometry into primitives at the intersection planes of the basis shapes, forming sliced single basis shape primitives. This partitioning procedure is however not easily generalizable. The five solids representing the Daya Bay photomultiplier tube are partitioned into a total of twelve single primitive parts which are copied to the GPU instead of the close to 3000 triangles of the tessellated geometry.

5. Optical Photon Simulation

Opticks is integrated with Geant4 by modifications to the scintillation and Cerenkov processes, instead of generating photons in a loop the "genstep" parameters are collected, including the number of photons to generate and a line segment along which to generate them and any other parameters needed for the photon generation. Collecting and copying gensteps rather than photons greatly reduces transfer overheads and avoids allocation of photon memory twice. In order to associate each photon with its genstep a seeding procedure is implemented using CUDA Thrust[26] that distributes genstep indices to photons entirely on the GPU.

Opticks uses cuRAND[27], from 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 thread which maintain their position within the sub-sequence. Initialization of cuRAND within the OptiX ray generation program was found to require increasing the stack size by a factor of 10 which led to poor ray tracing performance. To avoid this, cuRAND initialization was moved to a separate CUDA launch, allowing OptiX programs to use cuRAND without having to initialize it.

OptiX ray generation programs are the entry point of the ray tracing pipeline. A single OptiX launch conceptually creates an instanciation of the program for every photon. The generation of scintillation, Cerenkov or artificial torch photons are implemented in the ray generation program immediately prior to the propagation loop. Within the loop a ray trace is performed that provides the distance to the closest boundary and its boundary index. The boundary index together with the photon wavelength yields via texture lookups material properties such as absorption length, reemission probability and scattering length. These quantities together with uniform random number throws determine whether the photon is absorbed, scattered, undergoes reemission or survives to the boundary. Scintillator reemission changes the direction, polarization and wavelength of the photon. An inverted cumulative distribution function saved into a GPU texture allows reemission wavelengths to be generated from uniform random number texture lookups.

Photons reaching a boundary which has no associated optical surface are reflected or transmitted using a simplified translation of the Geant4 boundary processing with optimization techniques from the ray tracing literature that avoid the use of transcendental functions. Photons reaching a surface with an associated optical surface are detected, absorbed, diffusely reflected or specularly reflected following the Geant4 implementations.

Performing all photon operations on the GPU allows millions of optical photons to be generated and propagated with only the small fraction that are detected requiring memory allocation on the CPU to allow them to be copied back, using Thrust stream compaction, and added to Geant4 hit collections.

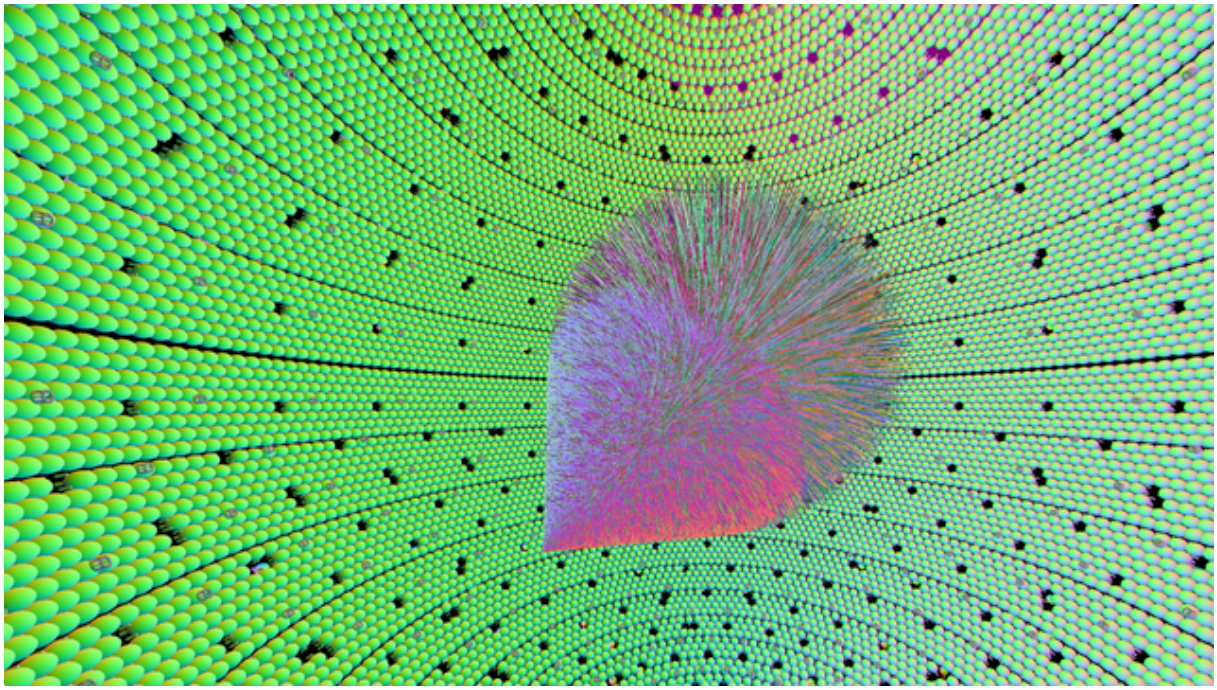

Figure 2. Simulated Cerenkov and scintillation photons from a 100 GeV muon travelling across the JUNO antineutrino detector viewed from inside the spherical scintillator. Primary particles are simulated by Geant4, generation "steps" of the primaries are transferred to the GPU and photons are generated, 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 the polarization direction.

6. Visualisation

Opticks visualisation provides both rasterized and ray traced views of geometry and rasterized representations of photon propagations controlled with an ImGui[28] interface. As modern GLSL shaders are used the visualisation requires at least OpenGL 4.0. The ray traced view uses the same geometry intersection code as that used by the photon simulation. Rasterized and ray traced views can be composited together using fragment depth information that is calculated and included with each ray traced pixel.

Interoperation techniques enable buffer sharing between OpenGL and OptiX that allow buffers written by OptiX programs to be directly rasterized with GLSL shaders invoked by OpenGL draw commands. Such techniques have been used to interactively visualize propagations of up to 16 steps of several millions of photons. Domain compression was used to squeeze the photon step parameters position, time, polarization and wavelength into 128 bits.

OpenGL geometry shaders are used to provide interactive forward/backwards time scrubbing and interactive photon history selection. Scrubbing uses the propagation time as an input allowing interpolation between the recorded photon positions. History selection relies on photon indexing implemented with CUDA Thrust sorting of 64 bit integers representing up to 16 steps of photon propagation history. History selection combined with artificial photon sources has proved very useful for debugging the simulation.

The GPU accelerated visualisation performance has made it possible to create screen capture movies illustrating simulated event propagations within the detector geometry, that have proved useful for outreach.

7. Validation and Outlook

Opticks aims to achieve a match between pure Geant4 simulations and the hybrid Opticks simulation for full geometries. To facilitate validation Opticks supports dynamic test geometries and light sources configured from the command-line that reuse materials and surfaces of a base geometry. Detailed point-by-point validation comparisons use a single executable that performs both the Geant4 and hybrid Opticks simulations and writes two events using an event format that includes highly compressed information describing the optical photon propagations. Propagations of one million photons are divided into the 100 most frequent history categories and χ^2 distances are formed for: photon history counts and point-by-point distributions of position, time, wavelength and polarization. This corresponds to comparisons of many thousands of histograms for each propagation.

After extensive development guided by the next largest χ^2 contribution a match has been achieved between the two simulation implementations, validating the GPU implementations of all the optical photon propagation processes relevant to Daya Bay and JUNO: absorption, Rayleigh scattering, Fresnel reflection and refraction, diffuse reflection and scintillator reemission. Similarly optical photon generation from scintillation and Cerenkov processes using buffers of generation step parameters collected from Geant4 have been validated. These validations were performed with several geometries that can be fully analytically described.

The detailed point-by-point validation approach described above is not straightforwardly applicable with tessellated geometries. Opticks currently has analytic CUDA intersection implementations for only a few geometrical shapes and does not yet support general boolean CSG geometries. Development of an OptiX CSG boolean implementation using a single hit intersection approach is in progress. Generalized boolean CSG tree ray tracing together with a small number of additional shape intersection implementations may allow automated translation of Geant4 geometries into analytic OptiX geometries avoiding the tessellation approximation. Adaption of the VecGeom[18] solid CUDA implementations to work within the OptiX context is another possibility for investigation.

8. Summary

Photon propagation timings for validated geometries obtained with a mobile Kepler generation GPU with only 384 CUDA cores are listed in Table 1. A speedup factor of 200 between Opticks compute and Geant4 is apparent. Ray trace performance has been verified to scale linearly with CUDA cores, thus GPU workstations with 10-20 times more cores are expected by linear extrapolation to easily achieve speed up factors exceeding 1000.

This level of speed up means that optical photon simulation time will become effectively zero compared to the rest of the simulation and also the CPU memory required for photons is reduced to just that required for detected photons. Opticks implementations of GPU photon generation and propagation have been validated by detailed point by point comparisons with Geant4. Validations of full geometries requires further geometry modelling development.

Although Opticks was developed for simulations of neutrino detectors it can benefit any simulation limited by optical photons, and the more optical photons the greater the benefit.

Acknowledgements

The Daya Bay and JUNO collaborations are acknowledged for the use of detector geometries and simulation software. Dr Tao Lin is acknowledged for his assistance with the JUNO Offline software. The Chroma authors are thanked for an excellent learning environment and the NVIDIA OptiX team is acknowledged for providing a package that has been a joy to apply.

Table 1. Comparison of photon propagation times in seconds obtained with Macbook Pro(2013) NVIDIA GeForce GT 750M, 2048MB, 384 cores. The first column lists photon counts and geometries. Rainbow geometry is a spherical drop of water.

Geometry	Geant4 10.2	Opticks Interop	Opticks Compute
1M Rainbow (S-polarized)	56	1.62	0.28
1M Rainbow (P-polarized)	58	1.71	0.25
0.5M PMT in Mineral Oil	41	0.81	0.15

References

[1] Newton I 1704 Opticks: or, a treatise of the reflections, refractions, inflexions and colours of light.
[2] Opticks URL `https://bitbucket.org/simoncblyth/opticks/`
[3] Agostinelli S, Allison J, Amako K, Apostolakis J, Araujo H, Arce P, et al. 2003 Geant4–a simulation toolkit *Nucl Instrum Methods Phys Res* A **506** pp 250-303
[4] Allison J, Amako K, Apostolakis J, Araujo H, Dubois P, Asai M, et al. 2006 Geant4 developments and applications *IEEE Trans Nucl Sci* **53** pp 270–8
[5] Allison J, Amako K, Apostolakis J, Arce P, Asai M, Aso T, et al. 2016 Recent developments in Geant4 *Nucl Instrum Methods Phys Res* A **835** pp 186–225
[6] Parker S, Bigler J, Dietrich A, Friedrich H, Hoberock J, et al. 2010 OptiX: a general purpose ray tracing engine *ACM Trans. Graph. : Conf. Series* **29** p 66
[7] NVIDIA® OptiX™ webpage `https://developer.nvidia.com/optix`
[8] An F, et al. 2016 The detector system of the Daya Bay reactor neutrino experiment *Nucl Instrum Methods* A **811** pp 133–161
[9] An F et al. 2016 Neutrino physics with JUNO *J Phys G* **43** 030401
[10] An F, et al. 2015 The muon system of the Daya Bay Reactor antineutrino experiment *Nucl Instrum Methods* A **773** pp 8–20
[11] Laine S, Karras T and Aila T 2013 Megakernels considered harmful: wavefront path tracing on GPUs *Proceedings of the 5th High-Performance Graphics Conference* pp 137–143
[12] Aila T and Laine S 2009 Understanding the efficiency of ray traversal on GPUs *Proceedings of the Conference on High Performance Graphics* pp 145–149
[13] Aila T, Laine S and Karras T 2012 Understanding the Efficiency of Ray Tracing on GPUs Kepler & Fermi addendum. *NVIDIA Technical Report NVR201202*
[14] Chroma URL `http://chroma.bitbucket.org`
[15] Seibert S and LaTorre A 2011 Fast optical monte carlo simulation with surface-based geometries using Chroma `http://chroma.bitbucket.org/_downloads/chroma.pdf`
[16] Fork of Chroma `http://bitbucket.org/simoncblyth/chroma`
[17] Amadio G, Apostolakis J, Bandieramonte M, Bhattacharyya A, Bianchini C, Brun R, Canal P, et al. 2015 The GeantV project : preparing the future of simulation *J. Phys.: Conf. Series* **664** 072006
[18] Apostolakis J, Bandieramonte M, Bitzes G, Brun R, Canal P, Carminati F, et al. 2015 Towards a high performance geometry library for particle-detector simulations *J. Phys.: Conf. Series* **608** 012023
[19] Stich M, Friedrich H and Dietrich A 2009 Spatial Splits in Bounding Volume Hierarchies *Proceedings of the Conference on High Performance Graphics* pp 7–13
[20] G4DAE URL `https://bitbucket.org/simoncblyth/g4dae/`
[21] Fork of Assimp supporting G4DAE extra optical properties `https://github.com/simoncblyth/assimp`
[22] Assimp URL `http://www.assimp.org`
[23] Kern R 2007 A Simple File Format for NumPy Arrays `https://docs.scipy.org/doc/numpy/neps/npy-format.html`
[24] IPython URL `https://ipython.org`
[25] Van der Walt S, Colbert S, Varoquaux G 2011 The NumPy array: a structure for efficient numerical computation *Comput. Sci. Eng.* **13** pp 22–30
[26] Bell N and Hoberock J 2011 *Thrust: a Productivity-Oriented Library for CUDA* (GPU Computing Gems Jade Edition) ed W W Hwu, Chapter 26
[27] cuRAND URL `http://docs.nvidia.com/cuda/curand/index.html`
[28] ImGui URL `https://github.com/ocornut/imgui`