# Opticks : GPU ray traced optical photon simulation

*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 NVIDIA GPU ray tracing, accessed via the NVIDIA OptiX API, with Geant4 toolkit based simulations. Optical photon simulation times of 10 seconds per 100 million photons have been measured with the full JUNO geometry and optical model with a 3rd RTX generation GPU. The JUNO optical model incorporates optical processes inside PMTs including thin layer interference effects. The GPU geometry is auto-translated from the Geant4 geometry. Optical physics processes of scattering, absorption, scintillator reemission and boundary processes are implemented in CUDA 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 performance measurements and new features developed to facilitate and optimize production usage of Opticks within the JUNO simulation framework. The features include geometry generalization enabling use of triangulated solids together with analytic geometry, out-of-core processing enabling simulation of events with more photons than can fit within available VRAM, interactive ray traced visualization and also the adoption of the Philox counter-based random number generator optimizing random number generation.

## 1 Introduction

Opticks[1-6] enables Geant4 [7] based simulations to benefit from high performance GPU ray tracing made accessible by NVIDIA® OptiX™ [8]. The Jiangmen Underground Neutrino Observatory (JUNO) [10] located in southern China is 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 in processing time and memory resources. Opticks eliminates these bottlenecks by offloading optical simulation to the GPU which 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.

Opticks was presented to the five prior CHEP conferences, with each contribution covering different aspects of its development. The first 2016 contribution [6] details the initial approach of using a mostly tesselated geometry with manual analytic PMTs, low resource usage of the curand random number generator within OptiX kernels as well as the CUDA port of photon generation and optical physics. The subsequent 2018 contribution [5] highlighted

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

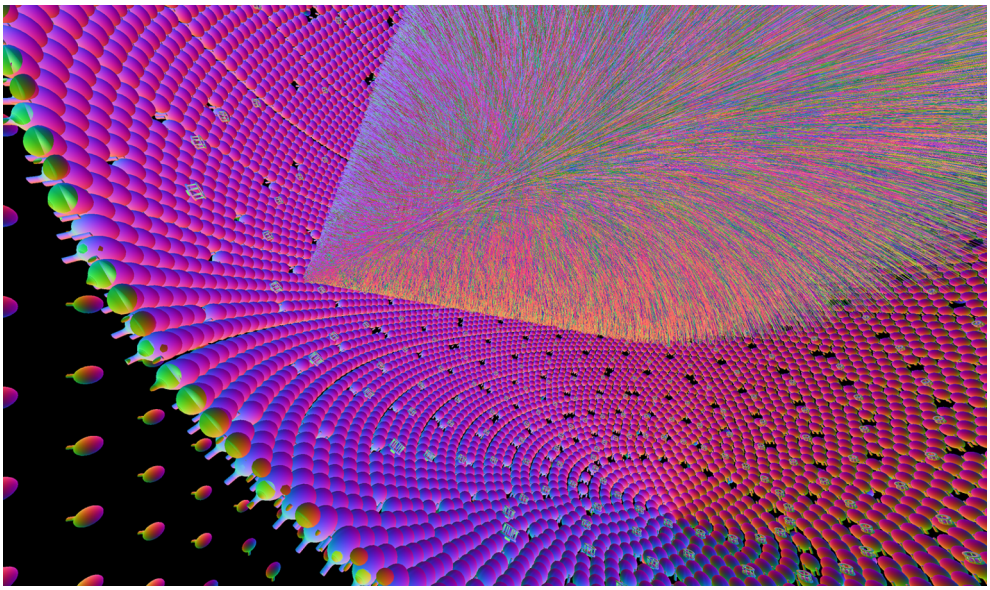

**Figure 1.** This figure is reproduced from [4]. Cutaway OpenGL rendering of millions of simulated optical photons from a 200 GeV muon crossing the JUNO liquid scintillator. Each line corresponds to a single photon with line colors representing the polarization direction. Primary particles are simulated by Geant4, "gensteps" are uploaded to the GPU and photons are generated, propagated and visualized all on the GPU.

development of a low resource CSG intersection algorithm. The 2019 plenary presentation and proceedings [4] focused on RTX [9] performance measurements, and the development of automated geometry translation. The 2021 contribution [3] covered initial stages of the transition to the NVIDIA OptiX 7+ API and the integration of Opticks with detector simulation frameworks. The 2023 contribution [2] covered the almost complete re-implementation of Opticks required to adopt the entirely new NVIDIA OptiX 7+ API with particular emphasis on the various geometry models and conversions between them.

These proceedings describe new features developed to facilitate and optimize production usage of Opticks within the JUNO simulation framework. The features include generalizations enabling integrated use of analytic and triangulated geometry representations and the automated handling of events with more photons than can fit within VRAM, interactive ray traced visualization and also the adoption of the Philox counter-based random number generator optimizing random number generation. In addition a simulation performance comparison between GPUs from the 1st and 3rd RTX generations is provided.

### 1.1 GPU ray tracing

GPUs evolved to perform rasterized rendering, optimizing throughput [11] rather than minimizing latency. GPUs are suited to problems with 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

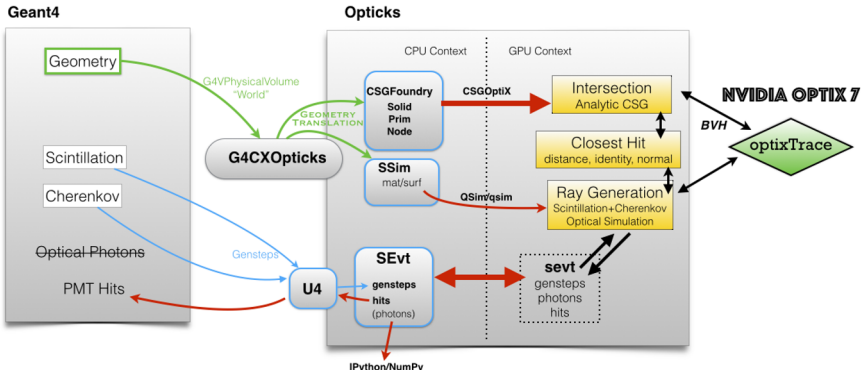


**Figure 2.** This figure is reproduced from [2], which provides further details. Illustration of the hybrid Geant4 + Opticks workflow : `G4CXOpticks` translates Geant4 geometry to GPU appropriate form. `U4` collects "gensteps" enabling GPU generation of scintillation and Cerenkov photons.

graphics. NVIDIA RTX [9] GPUs include hardware RT cores dedicated to ray geometry intersection. The first generation of RTX GPUs was introduced in 2018, each of the subsequent RTX generations have approximately doubled ray tracing performance with the 4th generation introduced in 2025 providing approximately 8 times faster ray tracing.

### 1.2 NVIDIA® OptiX™ ray tracing engine

OptiX [8] 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 allows user-defined primitives bounded by specified axis-aligned bounding to be implemented with CUDA intersection functions. Alternatively geometry can be defined with a set of triangles that use built-in triangle intersection. In August 2019 NVIDIA introduced the OptiX 7 API, 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. The latest OptiX 9.0.0 release from January 2025 has a very similar API to OptiX 7.

## 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. At each Geant4 simulated step of applicable particles the `G4Scintillation` and `G4Cerenkov` 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. Details on the GPU implementation of the simulation can be found in prior proceedings [2].

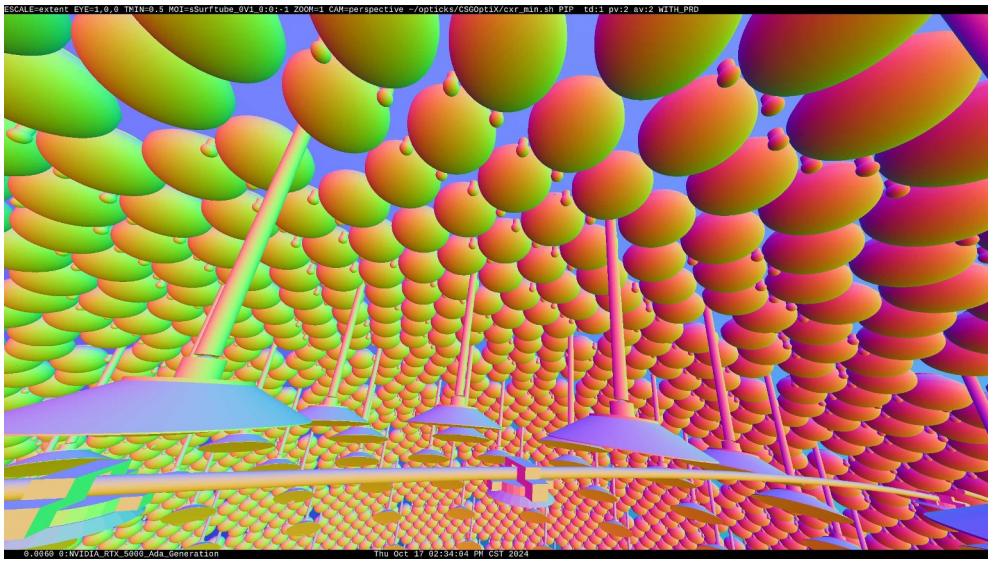


**Figure 3.** Screenshot from a smoothly interactive ray trace rendering application of the JUNO detector geometry comprising 2560x1440 ray traced pixels created with a CUDA launch of 6 ms using a single NVIDIA RTX 5000 Ada Generation GPU (3rd generation) with NVIDIA OptiX 8.0 and CUDA 12.4. The geometry is mostly analytic CSG with a few user selected solids, such as the guide tube torus, using a tesselated representation.

## 3 Detector geometry

At initialization Opticks translates the geometry through the below sequence of models:

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` : CPU/GPU model with `CSGSolid`, `CSGPrim` and `CSGNode`
4. NVIDIA OptiX 7+ : Instance and Geometry Acceleration Structures (IAS, GAS)

Both the `stree` and `CSGFoundry` geometry models are independent of Geant4 and can be persisted into directories of NumPy [12] binary files. The prior proceedings [2] provide a detailed description of the geometry models and conversion between them, including the crucial geometry factorization into repeated groups of volumes and a remainder of other insufficiently repeated volumes that is done within the intermediate `stree` model. Also the representation of solid shapes using n-ary trees of `sn.h` CSG nodes within the `stree` intermediate model are detailed.

### 3.1 Integrated Analytic and Triangulated geometry

Use of analytic CSG geometry typically allows float precision intersection positions to very closely match the double precision intersection positions that Geant4 provides. Intersection of a ray with a torus requires solution of a quartic equation with coefficients of greatly varying magnitude that result in unacceptably poor numerical precision for some rays when using

float precision. Replacing the analytic torus geometry with an approximate tesselated geometry is found to yield more robust intersect positions and avoids performance reductions from resorting to double precision.

During geometry factorization the `stree::collectGlobalNodes` method assigns non-instanced structural nodes as analytic or triangulated according to a user provided list of solid names to use a tesselated representation. This assignment is communicated via the `CSGSolid` to the geometry acceleration structure creation of `SBT::createGAS`.

## 4 Visualization

Opticks provides OpenGL based visualization using GLFW, a lightweight Graphics Library Framework, which manages OpenGL windows as well as keyboard and mouse inputs. Interoperation between OpenGL and CUDA provided by the CUDA runtime API allows CUDA/OptiX ray trace render kernels to write to OpenGL pixel buffer objects on the GPU which are subsequently accessed from the OpenGL rasterization pipeline via texture samplers. This GPU resource sharing enables smoothly interactive rendering of exactly the same detector geometry as that used by the simulation, as illustrated in Figure 3. Navigation in 3D via keys and mouse together with viewpoint bookmarking have enabled geometry overlap issues to be found by visual inspection. Initialization time for visualization of the full JUNO geometry is about two seconds to load and upload the binary NumPy arrays of the persisted CSGFoundry geometry.

The Opticks render kernel computes `uchar4` values for each pixel in the image plane with RGB color values based on the surface normal direction at geometry intersects. The fourth component of the pixel values is set to the z-depth in the eye frame mapped appropriately for the effective perspective or orthographic projection matrix, as shown in Figure 4. This depth component is used from the OpenGL shader rasterization pipeline to set the so called fragment depth of the ray traced pixel. This allows compositing of OptiX ray trace rendered pixels together with OpenGL rasterized fragments such that ray traced geometry can be drawn together with representations of photon positions or gensteps.

## 5 Optical physics

Opticks optical photon simulation is implemented in the QUDARap package, depending only on the SysRap base package and CUDA, with no dependency on OptiX, as detailed in the prior proceedings [2].

### 5.1 Optimizing random number generation

The curand library [13] is used for concurrent generation of pseudorandom numbers with separate curandState within each thread enabling reproducible simulation. As detailed in prior proceedings [6], with the old OptiX 5 API it was found that a large stack size was required to successfully initialize curand with the default XORWOW generator. This large stack size resulted in poor ray tracing performance, presumably due to larger thread resource usage limiting parallelism. This issue motivated install time creation of `curandStateXORWOW` struct for all photon "slots" which are persisted to binary files. These state files are loaded and uploaded to device at initialization, allowing each simulation kernel thread to use the XORWOW generator with its prepared state without the expense of initializing the state. This approach limits the maximum number of photons that can be simulated to the number of persisted states and also the initialization time to load and upload the states and required

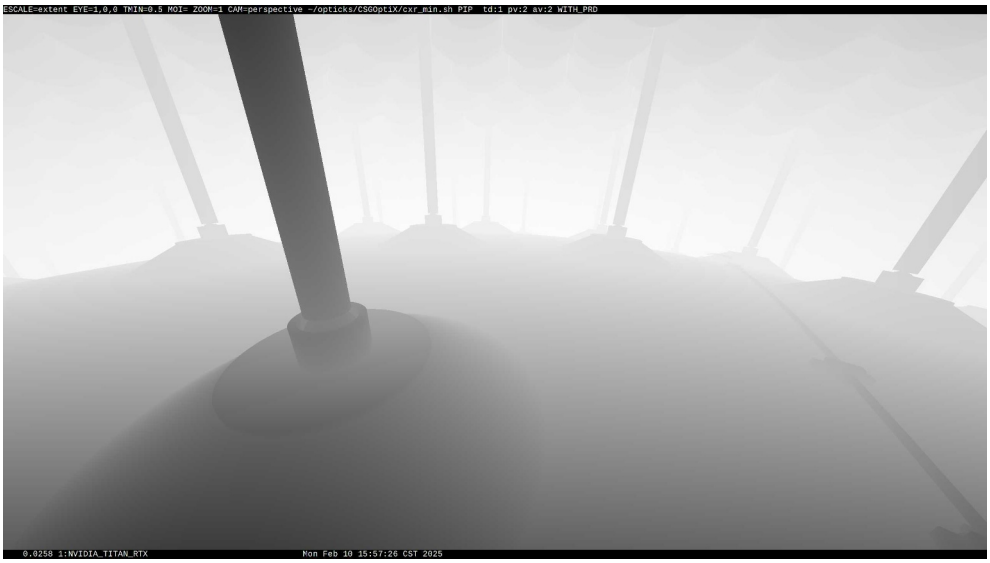


**Figure 4.** Render of eye frame z-depth from within JUNO detector geometry, used to enable compositing of ray traced geometry and rasterized event representations.

global memory becomes significant when simulating hundreds of millions of photons. The workaround of dividing curand initialization and generation into separate kernel launches is described with example code in the performance notes section of the curand manual [13].

In addition to the default `XORWOW` generator, curand provides several pseudorandom number generators including the `Philox4_32_10` generator. Philox is a counter based random number generator which may be implemented to use only integer counters for internal state. The statistical quality of Philox generated uniform random numbers are comparable to those from XORWOW [13]. Adopting the Philox random number generator within Opticks allows direct initialization within the simulation kernels avoiding the use of state files and hence removing the limitation on the number of photons that can be simulated and greatly reducing initialization time.

### 5.2 Out-of-core optical photon simulation

Simulation of more photons than can fit within VRAM is implemented using multiple kernel launches invoked from `QSim::simulate`. Following genstep collection index range slices into the genstep array are chosen such that the number of photons within each slice is less than the configured maximum number of CUDA thread "slots" within a single launch. The default maximum number of slots is determined based upon the total global memory of the GPU. Result arrays such as photons and hits are gathered from device to host into separate `NPFold` instances for each kernel launch which are subsequently concatenated. To check this functionality an event with one billion photons was simulated with an NVIDIA RTX 5000 Ada generation GPU with VRAM of 32GB in four kernel launches in a total time under 100 seconds.

## 6 Simulation performance and validation techniques

Using artificial point sources of photons known as "torch" gensteps with increasing numbers of photons in a sequence of purely optical fabricated events provides a convenient way to

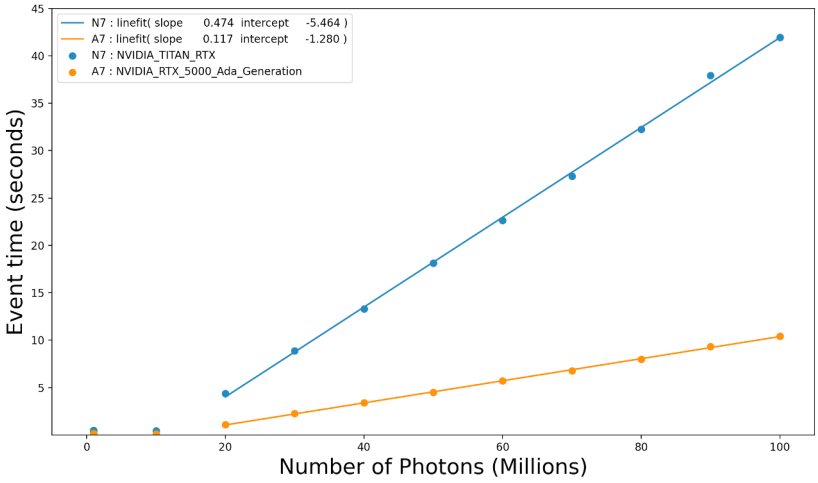


**Figure 5.** Optical photon simulation times for a sequence of purely optical events with numbers of photons in the range from one million to one hundred million. The blue and orange points are measurements from two Dell workstations with NVIDIA Titan RTX (1st gen.) and NVIDIA RTX 5000 Ada (3rd gen.) respectively.

scan performance as a function of photon count. Measurements of Opticks optical simulation times with 1st and 3rd generation RTX GPUs are shown in Figure 5. For both sets of measurements the simulation time is found to scale linearly with photon counts above 20M photons. The performance of the 3rd generation RTX GPU is found to be approximately a factor of four faster than the 1st generation GPU. For these measurements the default curand XORWOW generator was used with state loading as described in section 5.1. It is notable that the measured times at 1M and 10M photons are almost the same at 0.45(0.14) seconds with 1st(3rd) gen. RTX GPUs. Potentially a fixed overhead is dominating processing time. That overhead may be the copying of a large fixed number of curandState from global to local memory of all threads. This copying is done in order to avoid initialization of the XORWOW generator, as described in section 5.1. Further scans comparing performance with the XORWOW and Philox generators with and without curandState loading are needed to test this idea.

The Opticks optical simulation is validated by comparisons to Geant4 with several approaches. One approach uses common input photons and aligns the consumption of randoms by the two optical only simulations providing direct comparison unclouded by statistics, as described in [5]. In addition a statistical approach is used which contrasts the histories of photons between the two simulations using a Chi-squared comparison of the frequencies of different histories. Each step of every photon is characterized with a 4 bit integer enumeration representing reflection, transmission, bulk absorption, surface absorption, surface detection, scattering, re-emission etc.. Histories of up to 32 steps of the photon are collected into 128 bit integers for each photon. Arrays of these photon histories are recorded for both optical simulations. CUDA Thrust is used to sort the arrays of photon history integers and total the frequencies of each. These history frequencies are then compared using a Chi-squared. Issues with the simulations such as geometry overlaps, coincidences or close coincidences lead to large Chi-squared deviations in certain photon histories which typically require changes to the geometry or the modelling of the geometry to resolve.

## 7 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. Recent feature additions such as out-of-core optical photon simulation and the adoption of the curand Philox random number generator remove limits on the number of photons that can be simulated and make Opticks easier to use. Comparisons between different GPUs shows optical simulation performance to closely follow ray tracing performance with a factor of four improvement between GPUs from the first and third RTX generations.

## Acknowledgements

The JUNO collaboration is acknowledged for the use of detector geometries and simulation software. Dr. Tao Lin is acknowledged for his JUNO software assistance over many years. NVIDIA is acknowledged for extensive support for the OptiX 7+ API transition. 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/
Opticks References, https://simoncblyth.bitbucket.io
Opticks Group, https://groups.io/g/opticks
[2] S. Blyth, EPJ Web Conf. **295**, 11014 (2024)
https://doi.org/10.1051/epjconf/202429511014
[3] S. Blyth, EPJ Web Conf. **251**, 03009 (2021)
https://doi.org/10.1051/epjconf/202125103009
[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 et al., Nucl. Instrum. Methods. Phys. Res. A **506**, 250 (2003)
J. Allison et al., IEEE Trans Nucl Sci, **53**, 270 (2006)
J. Allison et al., Nucl. Instrum. Methods. Phys. Res. A **835**, 186 (2016)
[8] S. Parker et al., ACM Trans. Graph.: Conf. Series **29**, 66 (2010)
OptiX introduction, https://developer.nvidia.com/optix
OptiX API, https://raytracing-docs.nvidia.com/optix8/index.html
[9] NVIDIA RTX™ Platform, https://developer.nvidia.com/rtx
[10] Neutrino physics with JUNO
F. An et al., J. Phys. G. **43**, 030401 (2016)
[11] Understanding Throughput Oriented Architectures
M. Garland, D.B. Kirk, Commun. ACM **53**(11), 58 (2010)
[12] The NumPy array: a structure for efficient numerical computation
S. Van der Walt, S. Colbert, G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011)
[13] cuRAND, http://docs.nvidia.com/cuda/curand/index.html
Generator tests, https://docs.nvidia.com/cuda/curand/testing.html
https://docs.nvidia.com/cuda/curand/device-api-overview.html#performance-notes