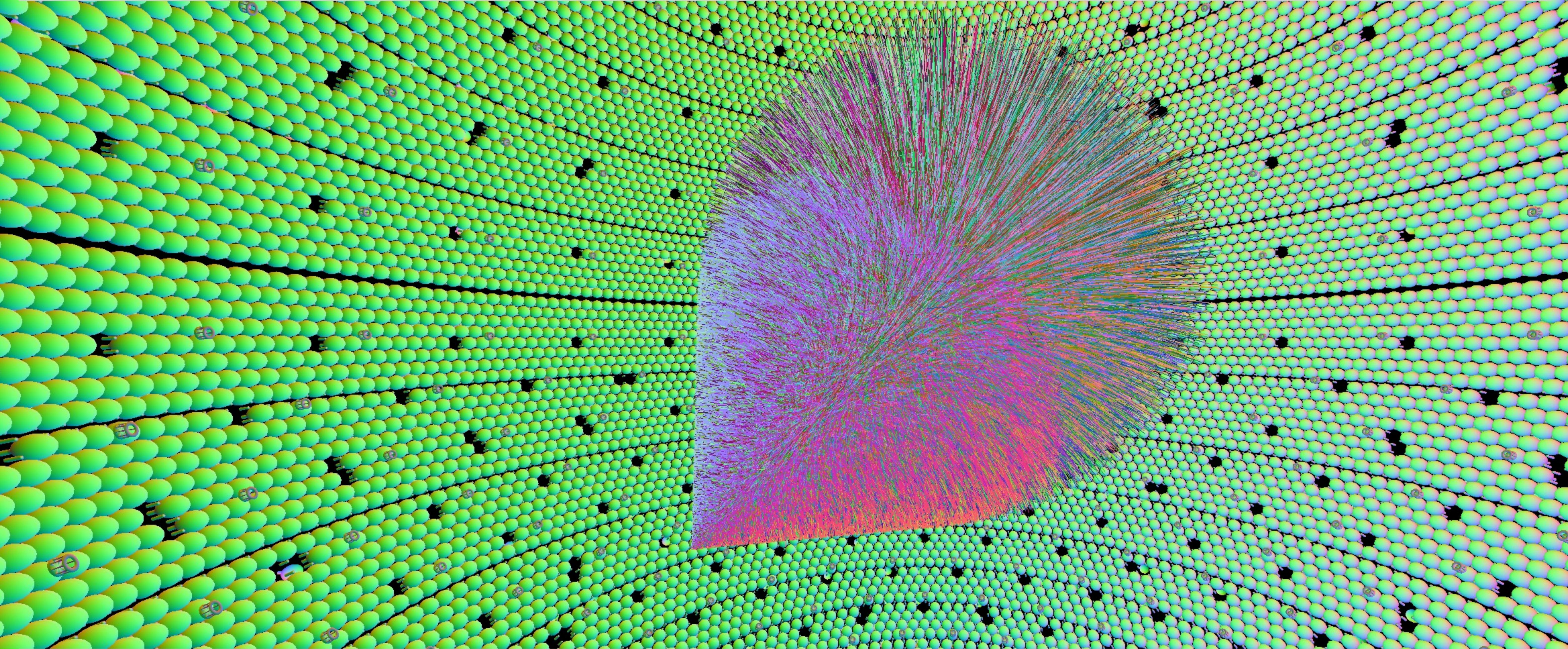


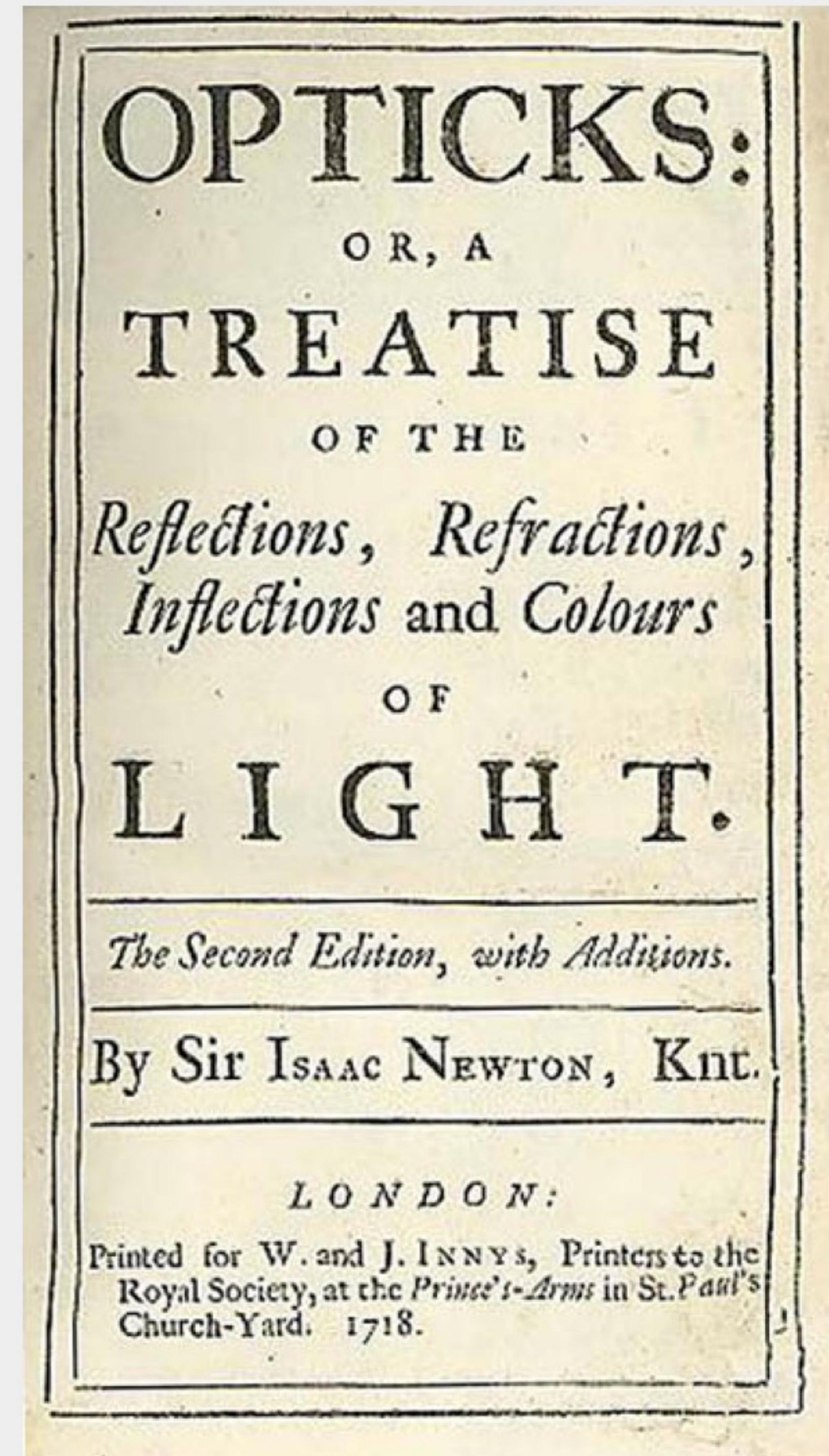
Opticks GPU Optical Simulation with NVIDIA® OptiX™ - Development Experience : Problems and Successes

Open source, <https://bitbucket.org/simoncblyth/opticks>

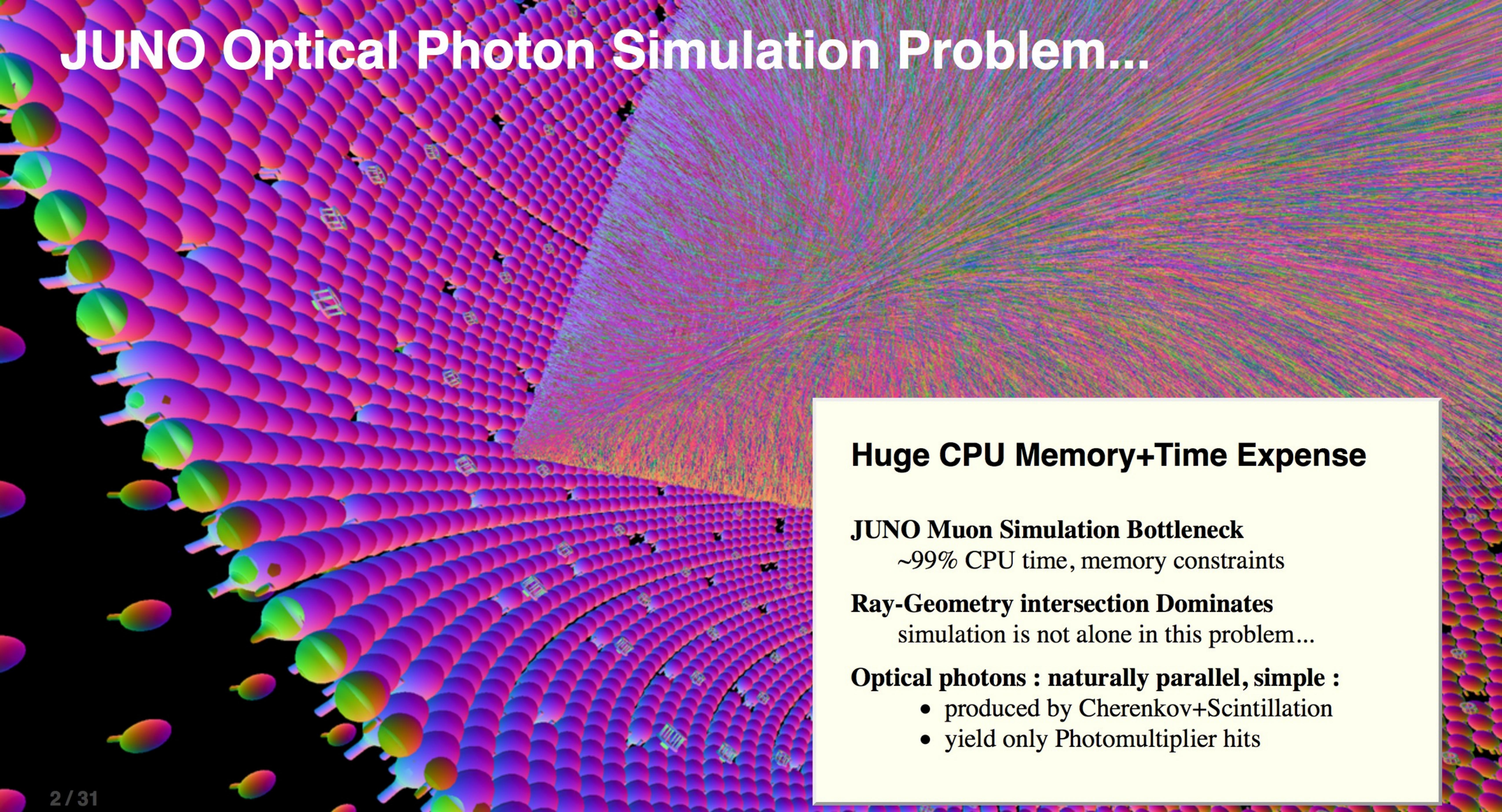


Outline

- JUNO Optical Photon Simulation Problem...
- GPU architecture and Ray tracing
 - CPU vs GPU architectures, Latency vs Throughput
 - Understanding GPU Graphical Origins -> Effective GPU Computation
 - Optical Photon Simulation \approx Ray Traced Image Rendering
 - Rasterization and Ray tracing
 - Turing Built for RTX, BVH : Bounding Volume Hierarchy
 - NVIDIA OptiX Ray Tracing Engine
- Opticks : Translate Geant4 Context to GPU
 - Geant4 + Opticks Hybrid Workflow : External Optical Photon Simulation
 - Opticks : Translates G4 Optical Physics to CUDA/OptiX
 - G4Solid -> CUDA Intersect Functions for ~10 Primitives
 - G4Boolean -> CUDA/OptiX Intersection Program Implementing CSG
 - Opticks : Translates G4 Geometry to GPU, Without Approximation
- Validation and Performance
 - Validation of Opticks Simulation by Comparison with Geant4
 - Performance Scanning from 1M to 400M Photons
- Opticks Experience : Problems and Successes
 - Main Operational Problem : Manpower
 - Main Technical Problem : Geometry Translation
 - Further Problems with using NVIDIA OptiX
 - Benefits from using NVIDIA OptiX



JUNO Optical Photon Simulation Problem...



Huge CPU Memory+Time Expense

JUNO Muon Simulation Bottleneck

~99% CPU time, memory constraints

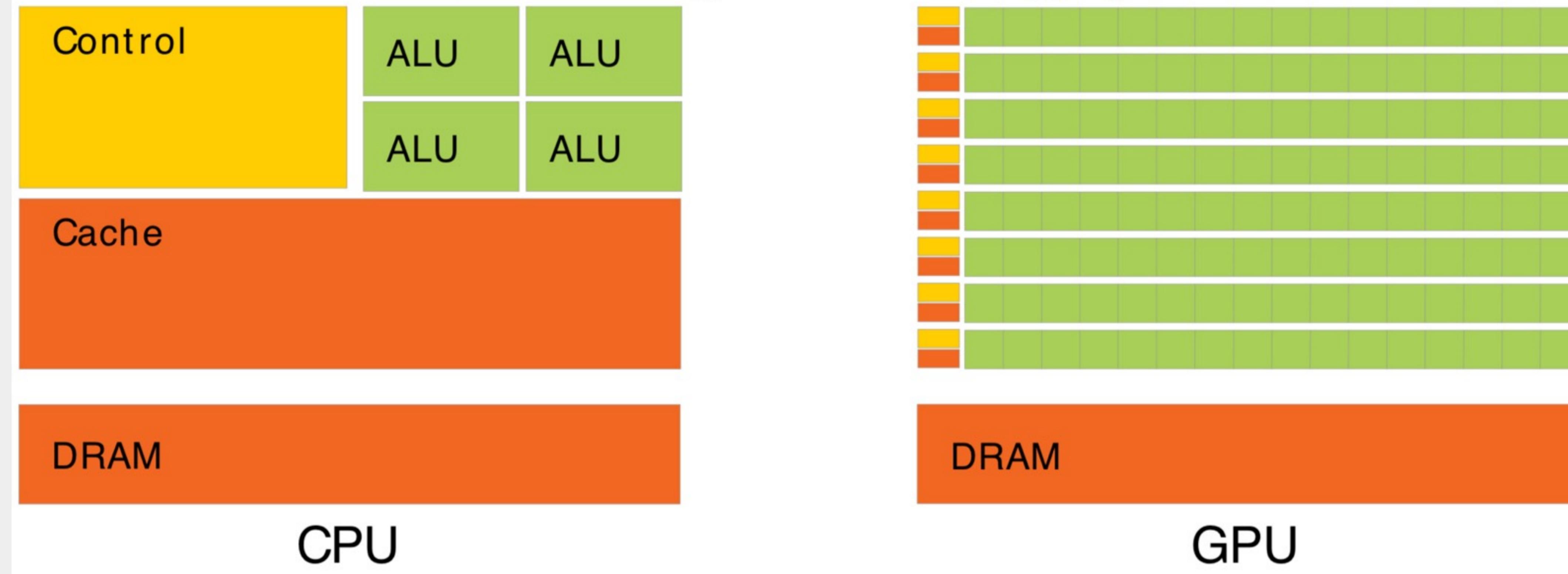
Ray-Geometry intersection Dominates

simulation is not alone in this problem...

Optical photons : naturally parallel, simple :

- produced by Cherenkov+Scintillation
- yield only Photomultiplier hits

CPU vs GPU architectures, Latency vs Throughput



Waiting for memory read/write, is major source of latency...

CPU : latency-oriented : Minimize time to complete single task : *avoid latency with caching*

- complex : caching system, branch prediction, speculative execution, ...

GPU : throughput-oriented : Maximize total work per unit time : *hide latency with parallelism*

- many simple processing cores, hardware multithreading, SIMD (single instruction multiple data)
- simpler : **lots of compute (ALU)**, at expense of cache+control
- design assumes **abundant parallelism**

Effective use of **Totally different processor architecture** -> Total reorganization of data and computation

Understanding GPU Graphical Origins -> Effective GPU Computation

GPUs evolved to rasterize 3D graphics at 30/60 fps

- 30/60 "launches" per second, each handling millions of items
- literally billions of small "shader" programs run per second

Simple Array Data Structures (N-million,4)

- millions of vertices, millions of triangles
- vertex: (x y z w)
- colors: (r g b a)

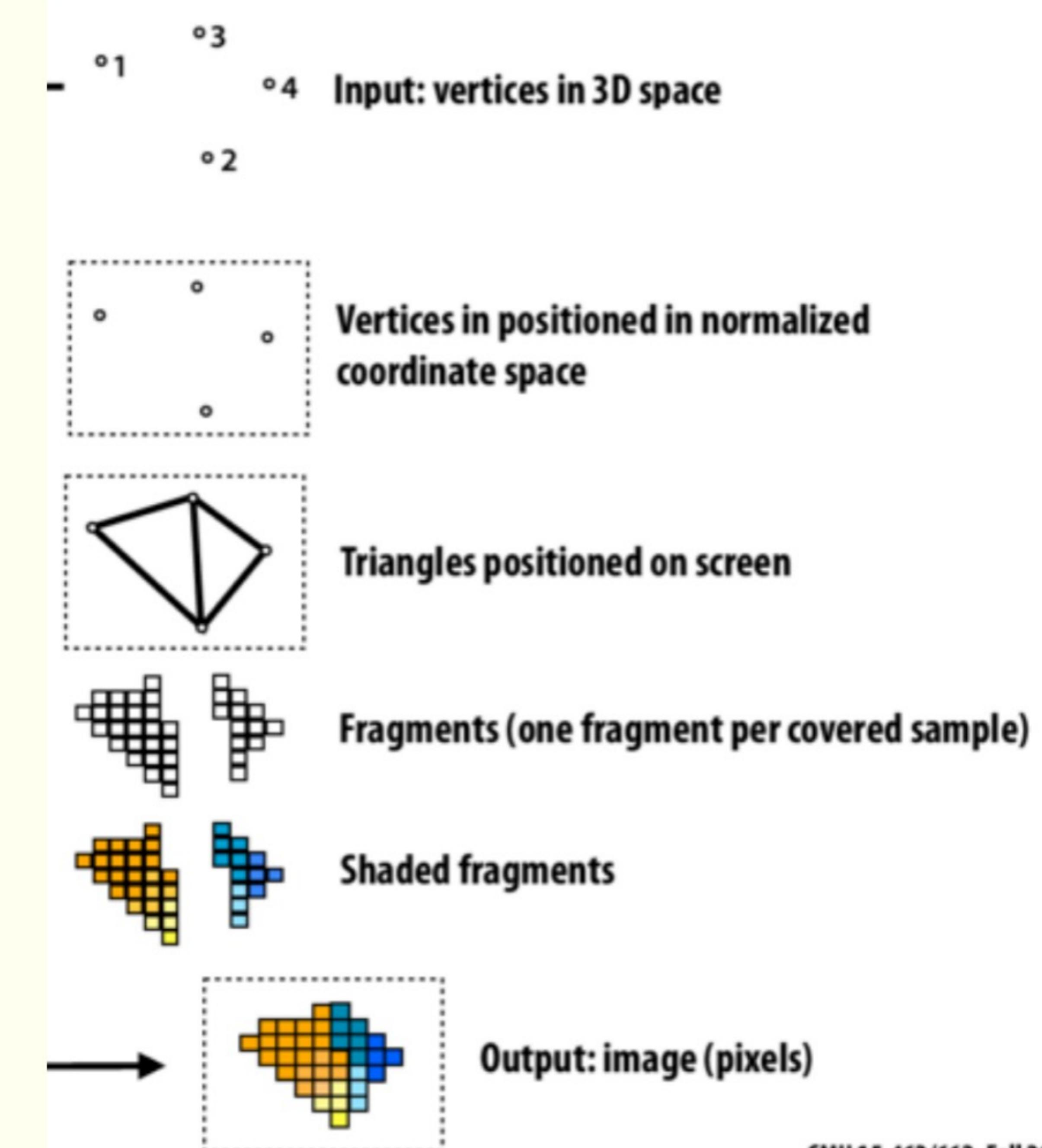
Constant "Uniform" 4x4 matrices : scaling+rotation+translation

- 4-component homogeneous coordinates -> easy projection

Graphical Experience Informs Fast Computation on GPUs

- array shapes similar to graphics ones are faster
 - "float4" 4*float(32bit) = 128 bit memory reads are favored
 - Opticks photons use "float4x4" just like 4x4 matrices
- GPU Launch frequency < ~30/60 per second
 - avoid copy+launch overheads becoming significant
 - ideally : handle millions of items in each launch

OpenGL Rasterization Pipeline



Optical Photon Simulation ≈ Ray Traced Image Rendering

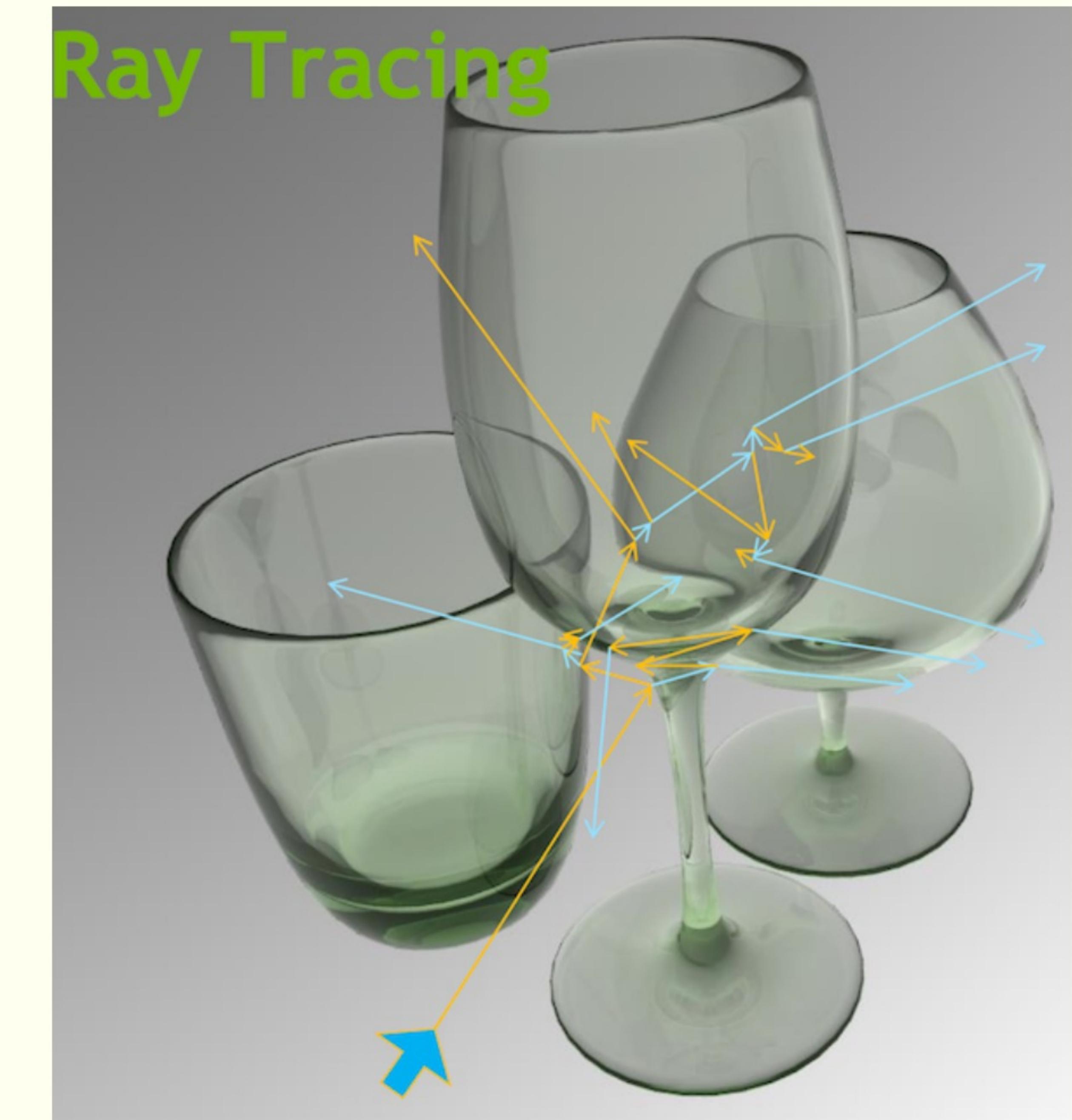
Much in common : geometry, light sources, optical physics

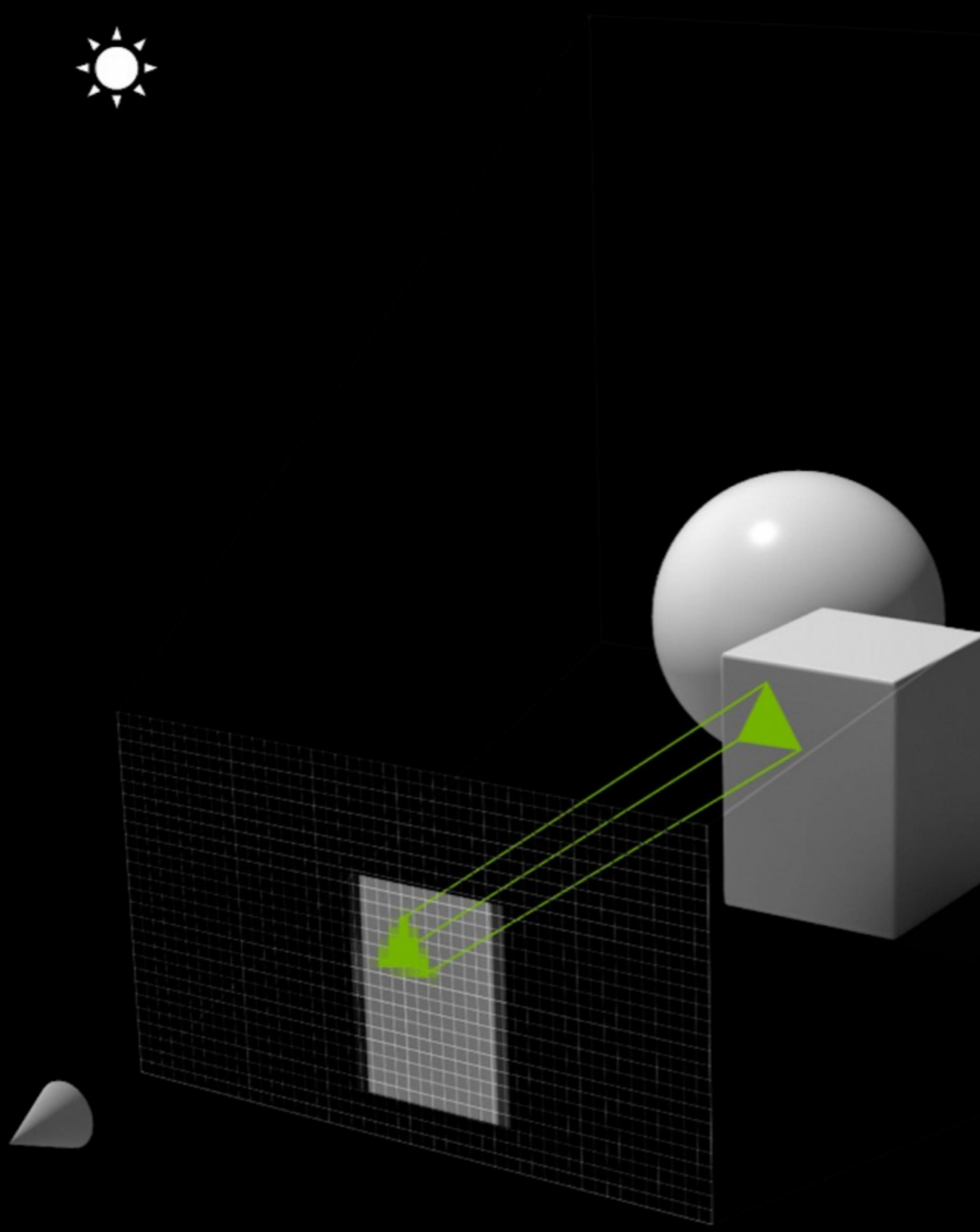
- simulation : photon parameters at PMT detectors
- rendering : pixel values at image plane
- both limited by ray geometry intersection, aka ray tracing

Many Applications of ray tracing :

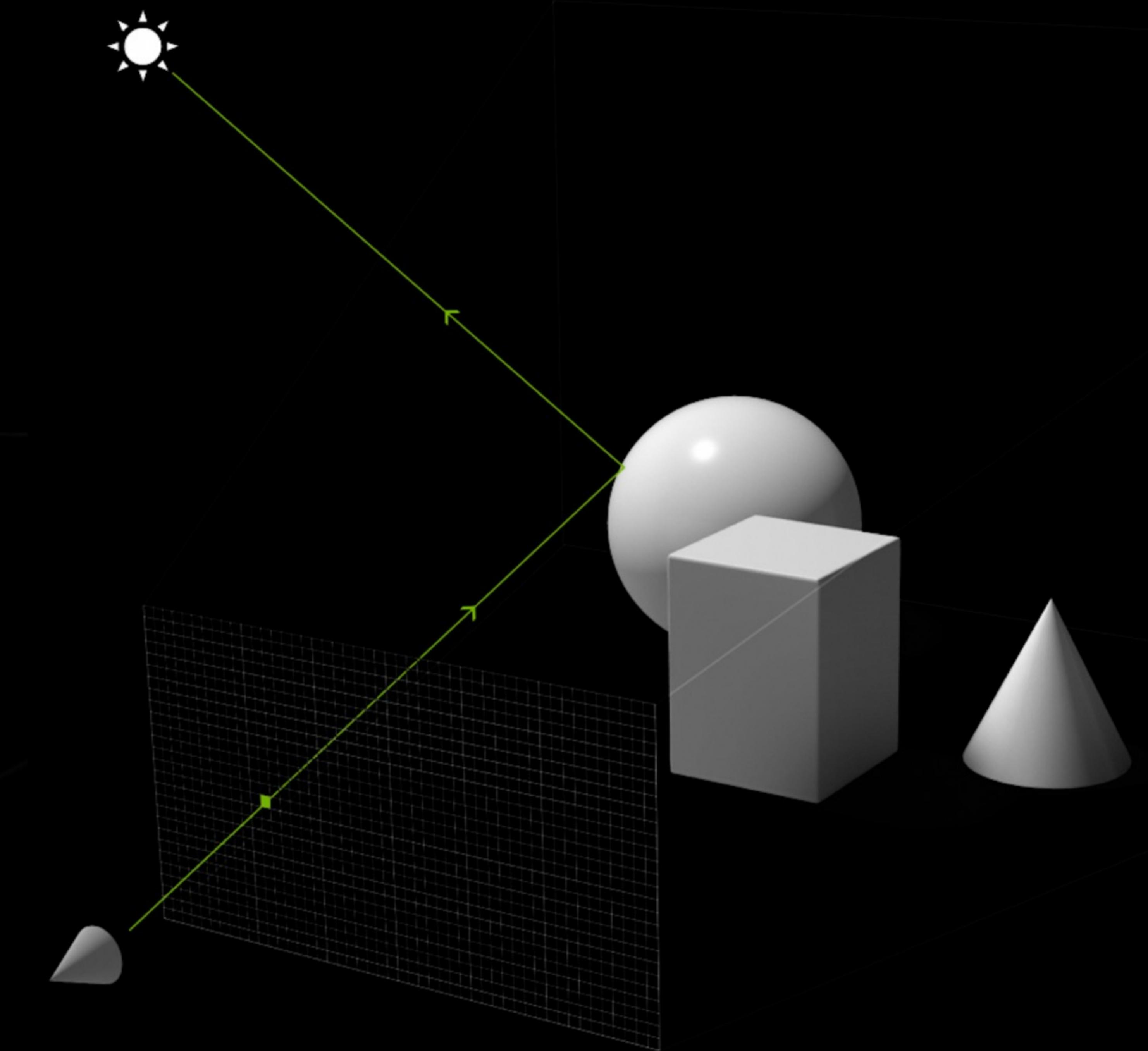
- advertising, design, architecture, films, games,...
- -> huge efforts to improve hw+sw over 30 yrs

Not a Photo, a Calculation





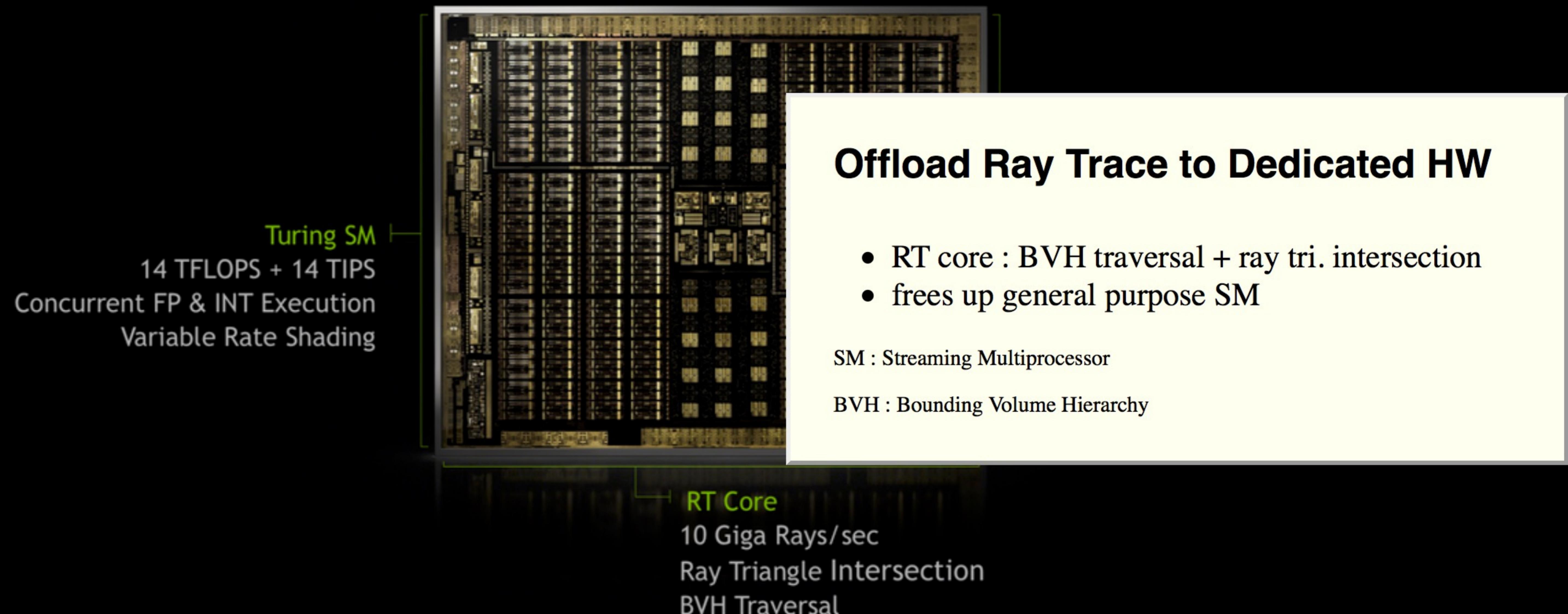
RASTERIZATION



RAY TRACING

TURING BUILT FOR RTX

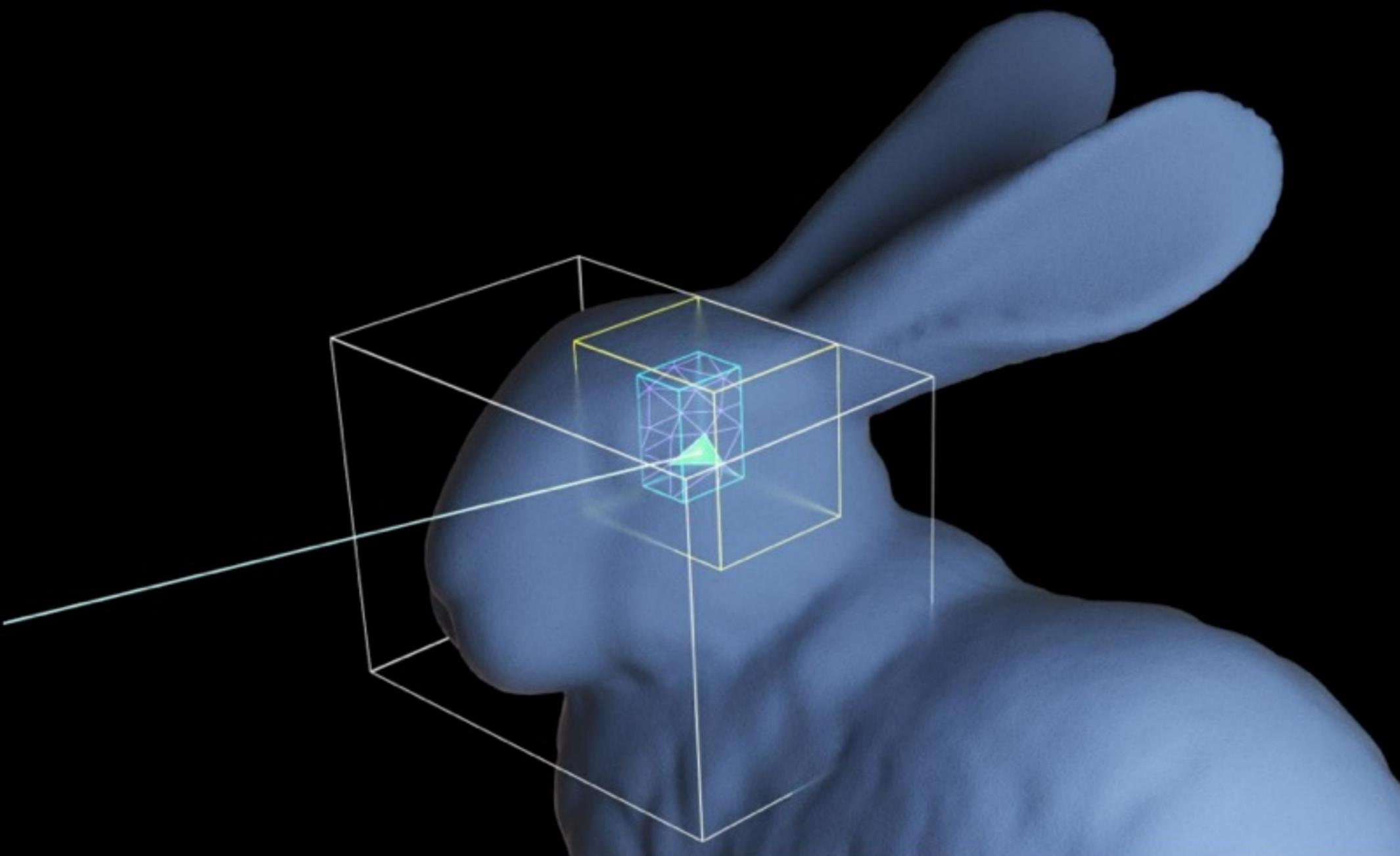
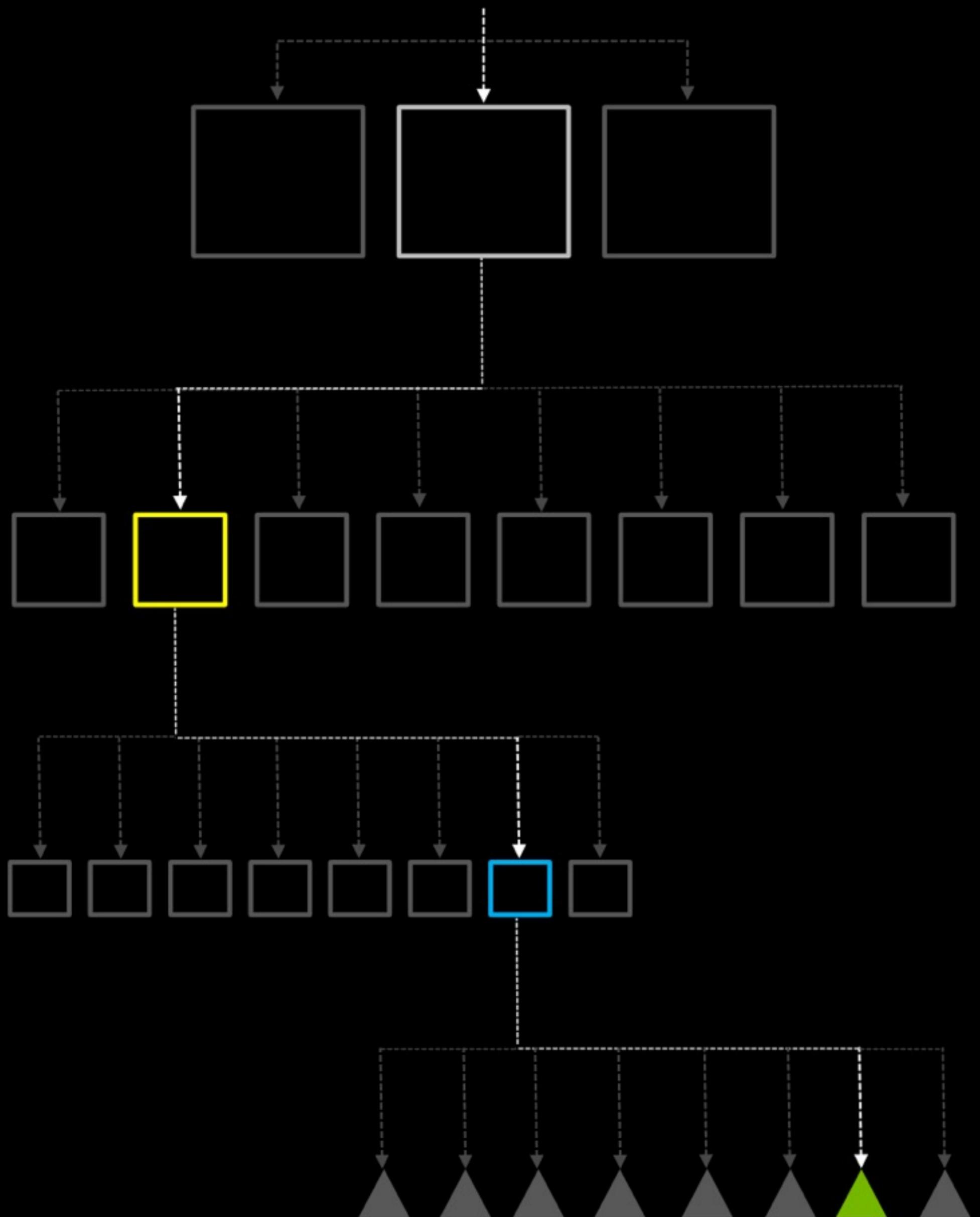
GREATEST LEAP SINCE 2006 CUDA GPU



Spatial Index Acceleration Structure

BVH ALGORITHM

Massive Improvement in Search Efficiency



Tree of Bounding Boxes (bbox)

- aims to minimize bbox+primitive intersects
- accelerates ray-geometry intersection

NVIDIA® OptiX™ Ray Tracing Engine -- <http://developer.nvidia.com/optix>

OptiX makes GPU ray tracing accessible

- accelerates ray-geometry intersections
- simple : single-ray programming model
- "...free to use within any application..."
- access RT Cores[1] with OptiX 6.0.0+ via RTX™ mode

NVIDIA expertise:

- compiler optimized for GPU ray tracing
- ~linear scaling up to 4 GPUs
- acceleration structure creation + traversal (Blue)
- instanced sharing of geometry + acceleration structures

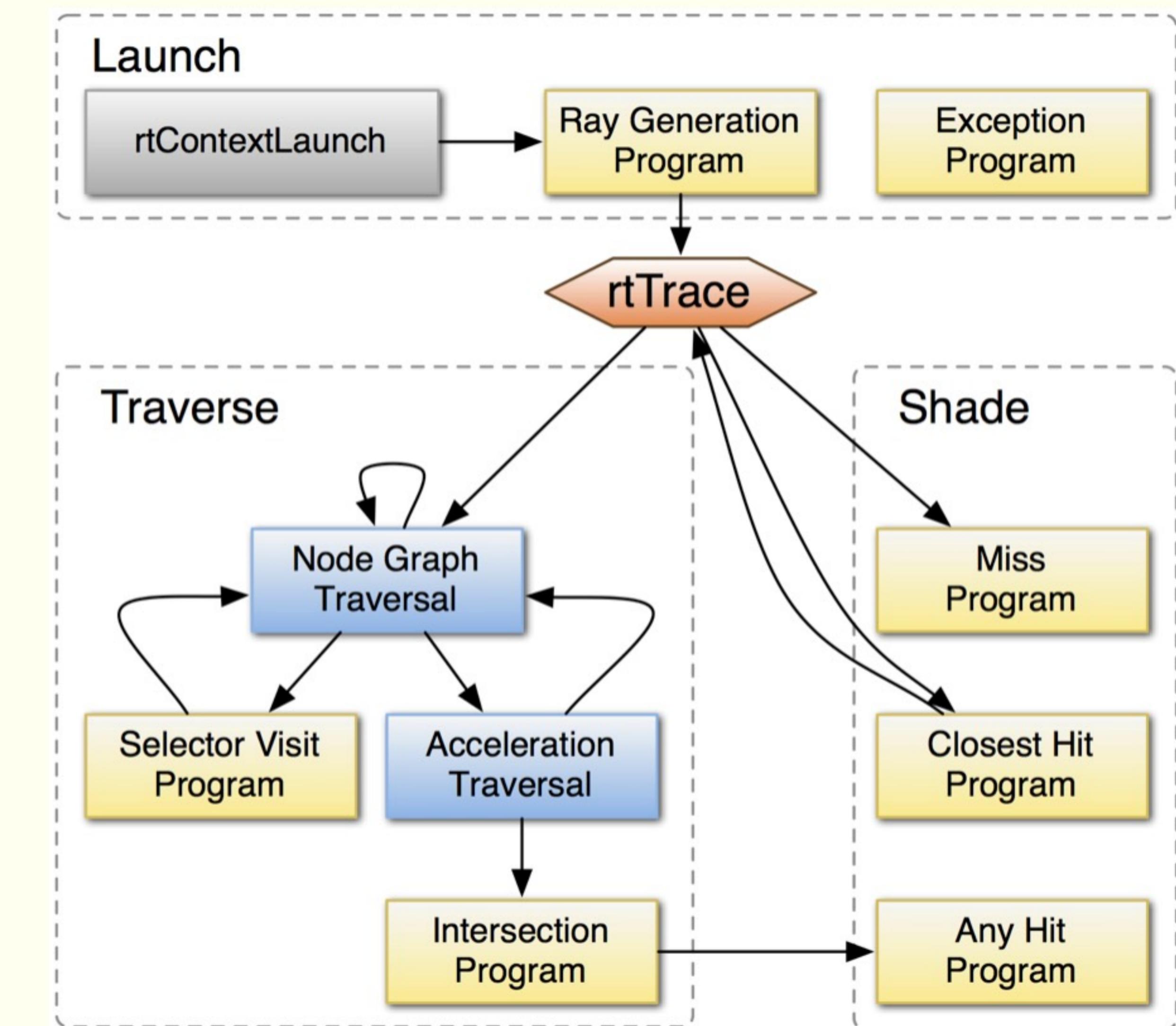
Opticks provides (Yellow):

- ray generation program
- ray geometry intersection+bbox programs

[1] Turing RTX GPUs

OptiX Raytracing Pipeline

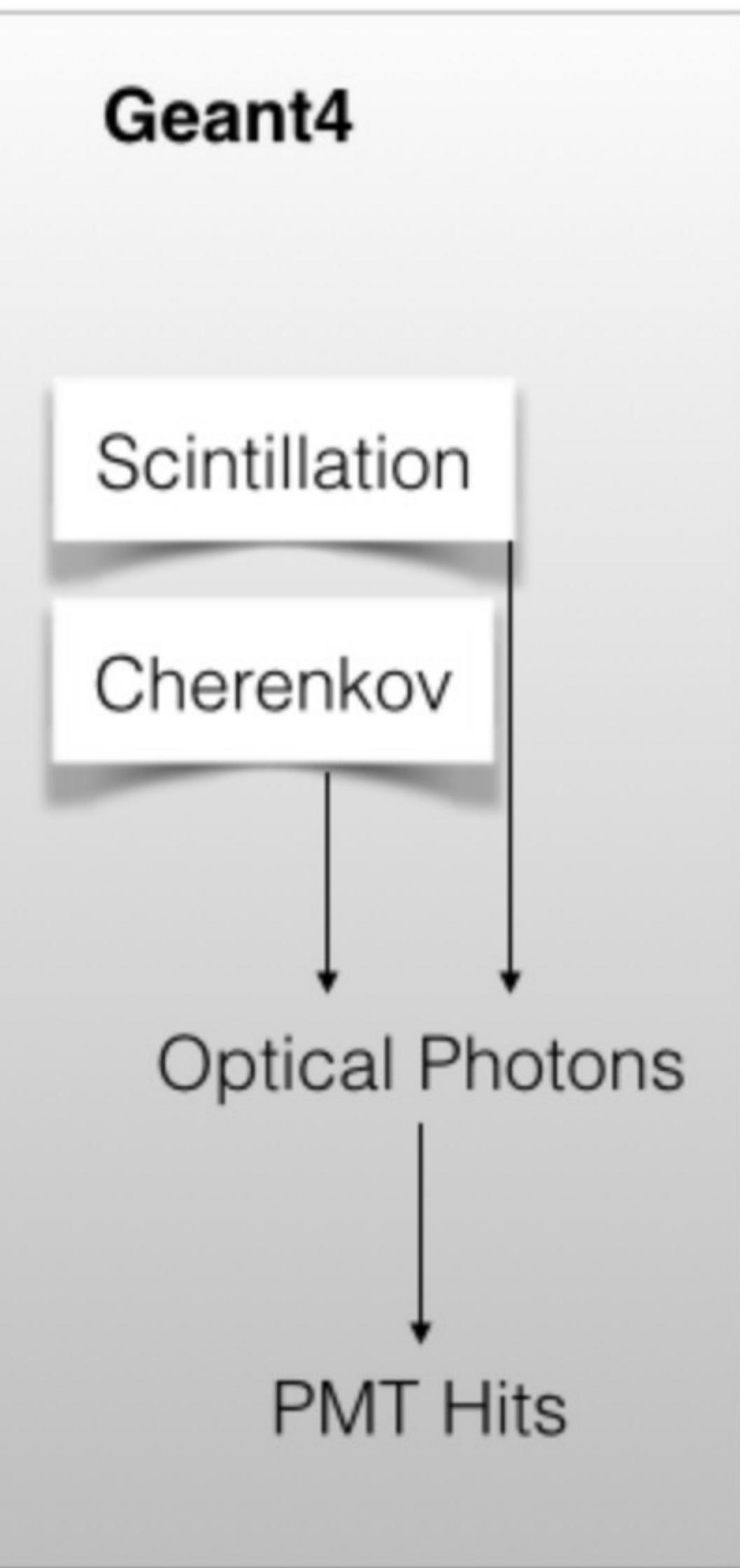
Analogous to OpenGL rasterization pipeline:



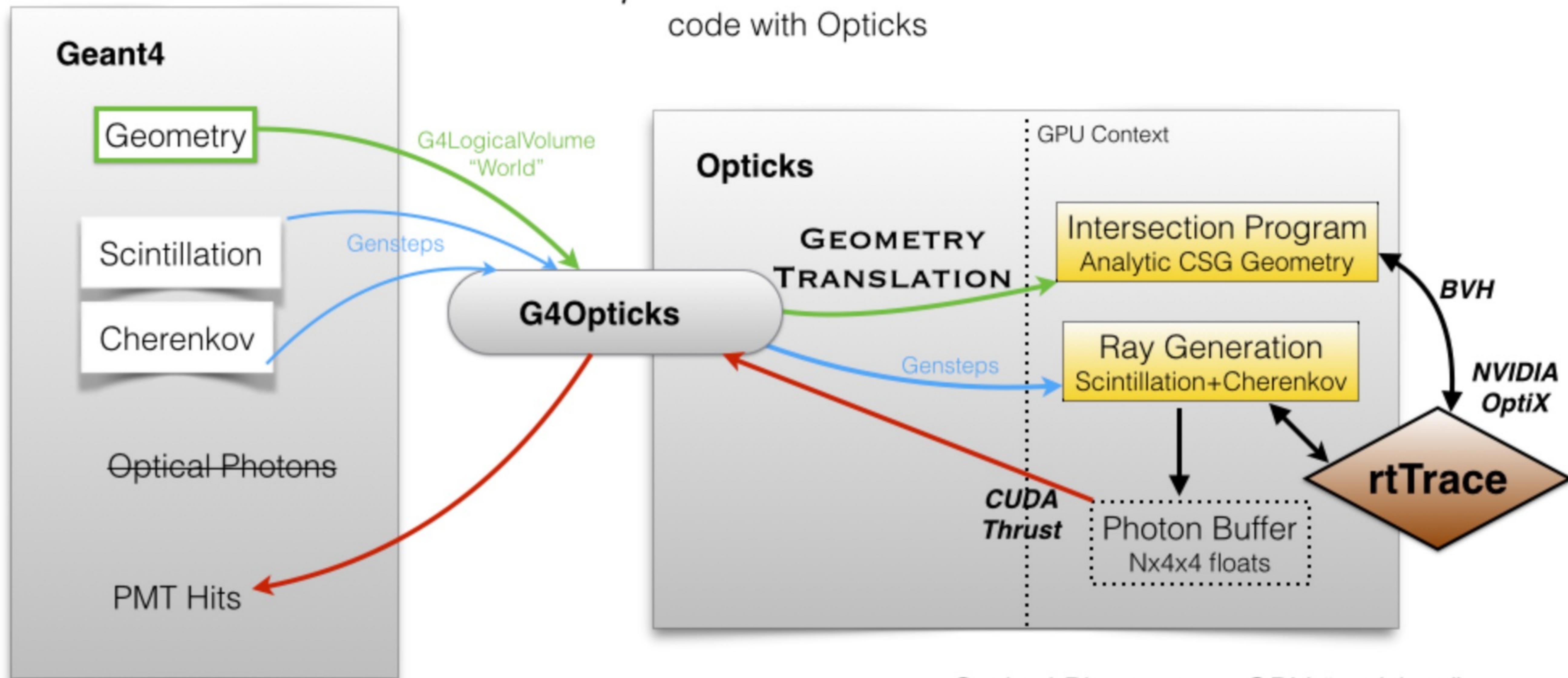
Geant4 + Opticks Hybrid Workflow : External Optical Photon Simulation

<https://bitbucket.org/simoncblyth/opticks>

Standard Workflow



Hybrid Workflow



G4Opticks interfaces Geant4 user code with Opticks

Optical Photons are GPU “resident”,
only hits are copied to CPU memory

Opticks : Translates G4 Optical Physics to CUDA/OptiX

OptiX : single-ray programming model -> line-by-line translation

CUDA Ports of Geant4 classes

- G4Cerenkov (only generation loop)
- G4Scintillation (only generation loop)
- G4OpAbsorption
- G4OpRayleigh
- G4OpBoundaryProcess (only a few surface types)

Modify Cherenkov + Scintillation Processes

- collect *genstep*, copy to GPU for generation
- avoids copying millions of photons to GPU

Scintillator Reemission

- fraction of bulk absorbed "reborn" within same thread
- wavelength generated by reemission texture lookup

Opticks (OptiX/Thrust GPU interoperation)

- OptiX : upload gensteps
- Thrust : seeding, distribute genstep indices to photons
- OptiX : launch photon generation and propagation
- Thrust : pullback photons that hit PMTs
- Thrust : index photon step sequences (optional)

GPU Resident Photons

Seeded on GPU

associate photons -> *gensteps* (via seed buffer)

Generated on GPU, using genstep param:

- number of photons to generate
- start/end position of step

Propagated on GPU

Only photons hitting PMTs copied to CPU

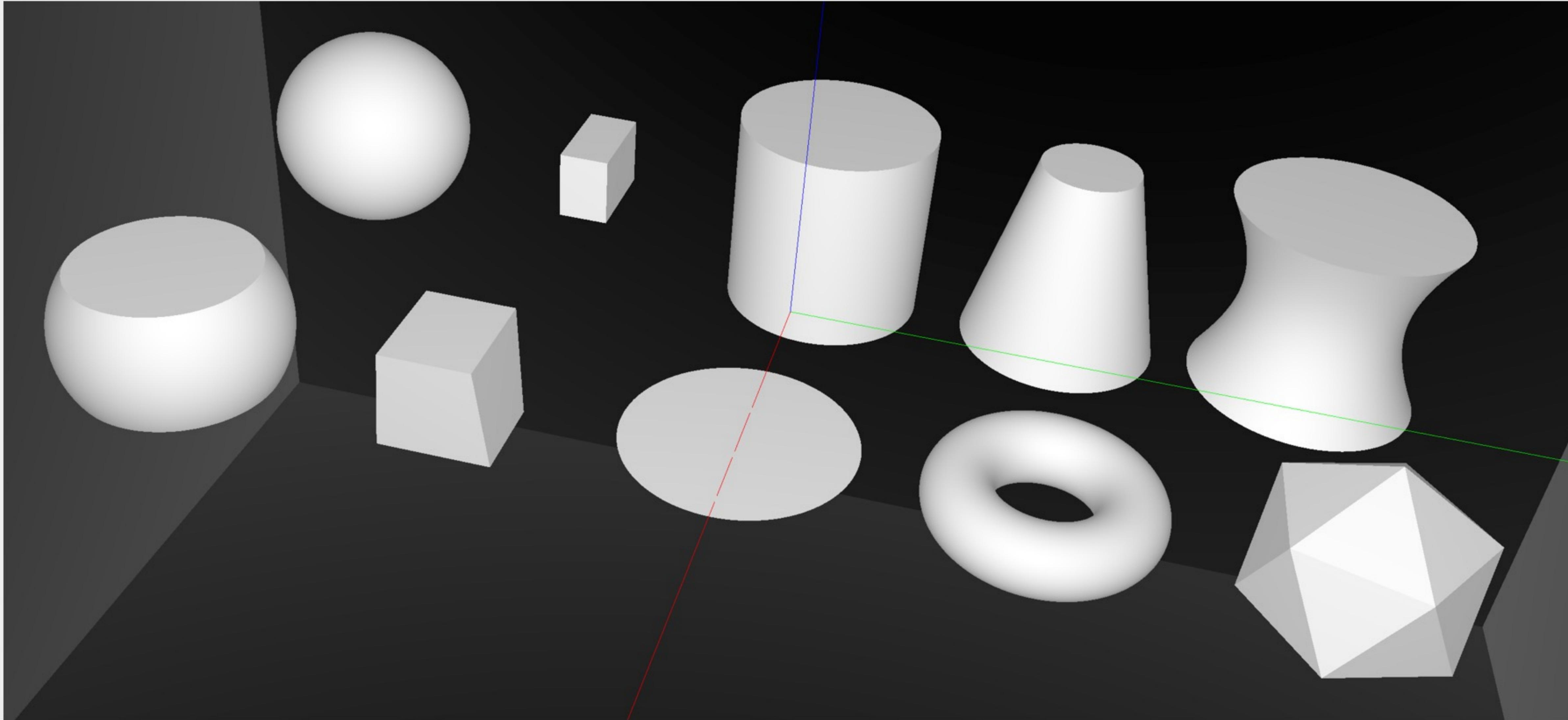
Thrust: high level C++ access to CUDA



- <https://developer.nvidia.com/Thrust> □

G4Solid -> CUDA Intersect Functions for ~10 Primitives

- 3D parametric ray : $\text{ray}(x,y,z;t) = \text{rayOrigin} + t * \text{rayDirection}$
- implicit equation of primitive : $f(x,y,z) = 0$
- -> polynomial in t , roots: $t > t_{\min}$ -> intersection positions + surface normals



*Sphere, Cylinder, Disc, Cone, Convex Polyhedron, Hyperboloid, **Torus**, ...*

G4Boolean -> CUDA/OptiX Intersection Program Implementing CSG

Complete Binary Tree, pick between pairs of nearest intersects:

$UNION \ tA < tB$	Enter B	Exit B	Miss B
Enter A	ReturnA	LoopA	ReturnA
Exit A	ReturnA	ReturnB	ReturnA
Miss A	ReturnB	ReturnB	ReturnMiss

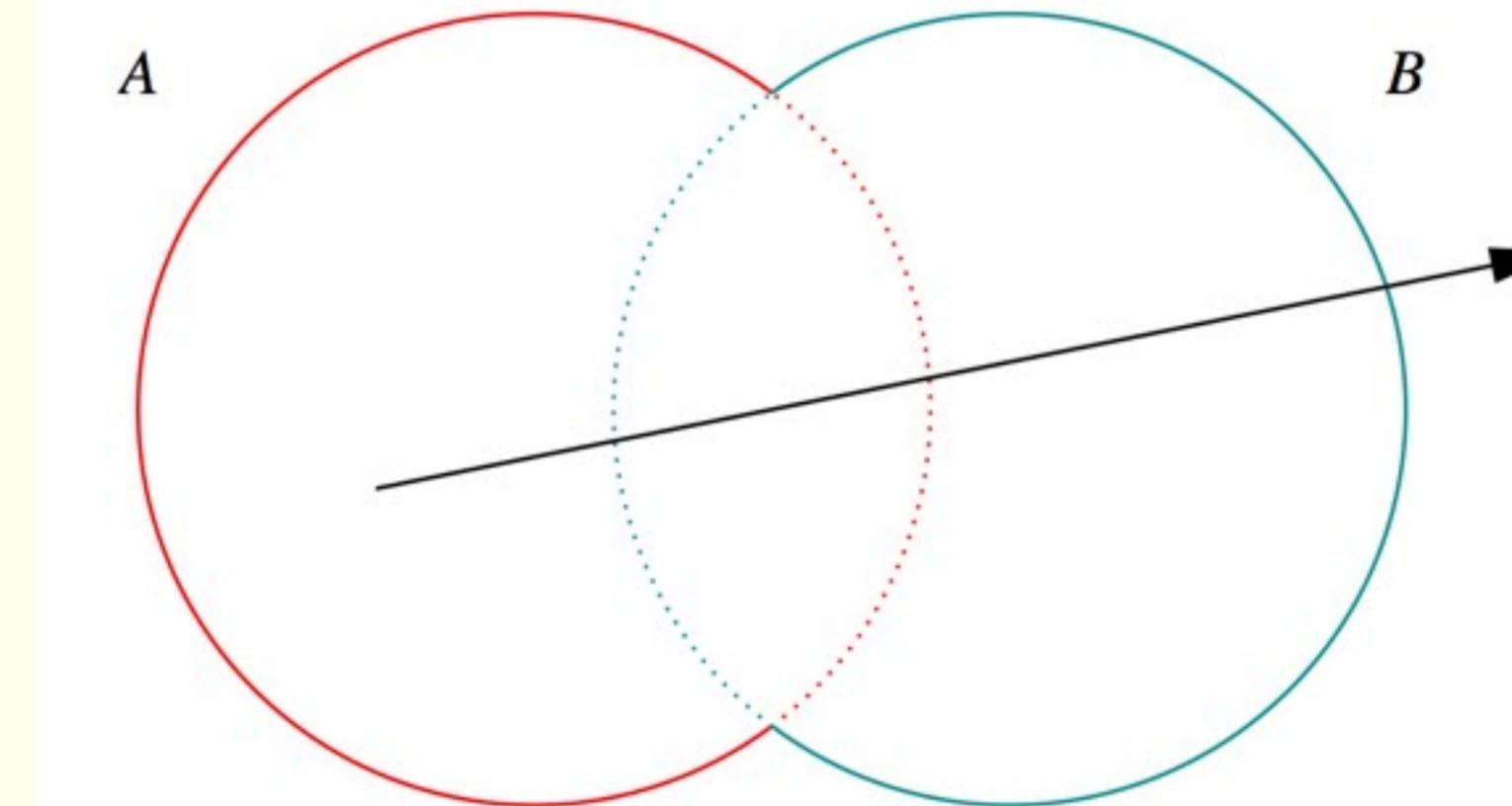
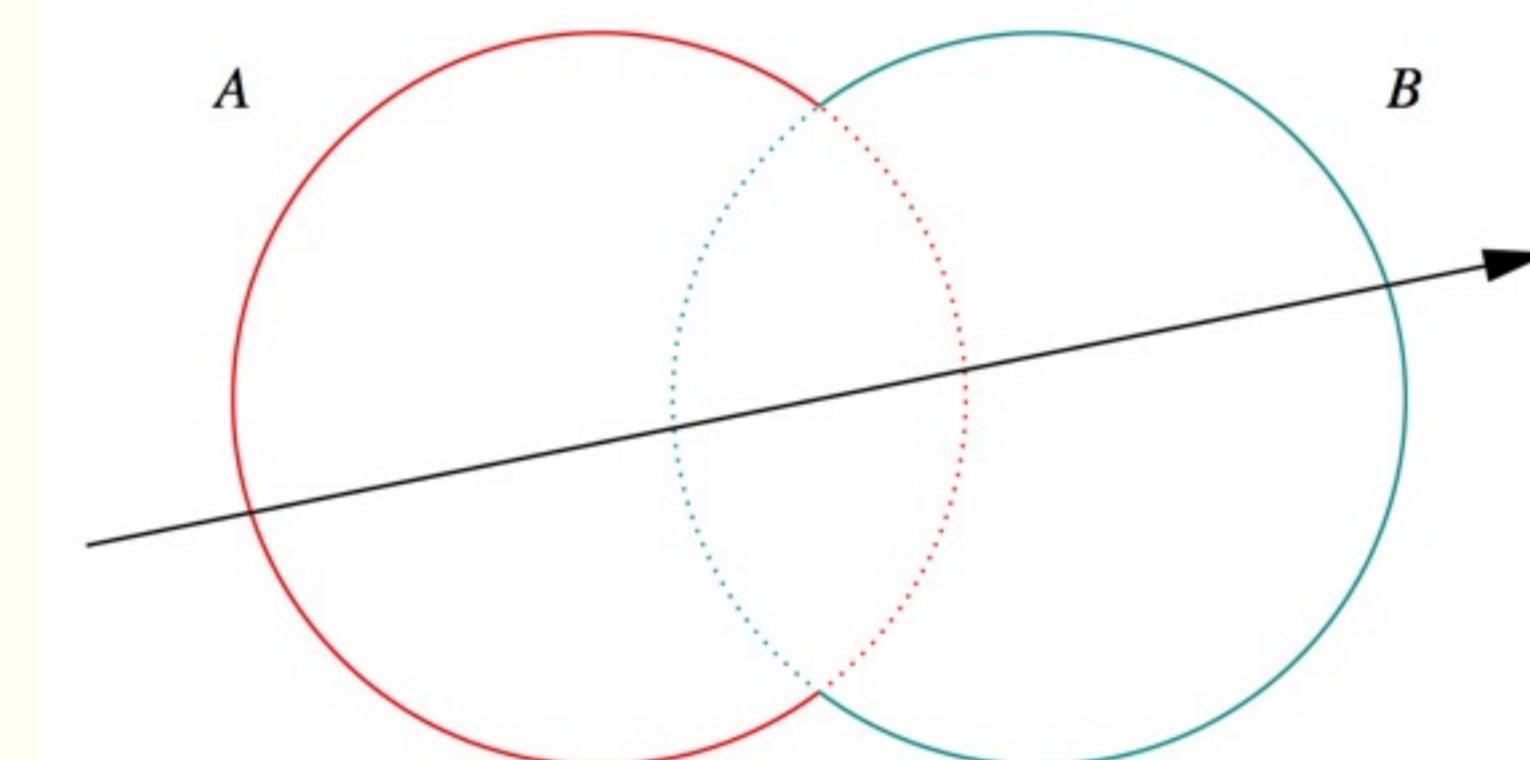
- *Nearest hit intersect algorithm* [1] avoids state
 - sometimes Loop : advance t_{min} , re-intersect both
 - classification shows if inside/outside
- *Evaluative* [2] implementation emulates recursion:
 - recursion not allowed in OptiX intersect programs
 - bit twiddle traversal of complete binary tree
 - stacks of postorder slices and intersects
- **Identical geometry to Geant4**
 - solving the same polynomials
 - near perfect intersection match

[1] Ray Tracing CSG Objects Using Single Hit Intersections, Andrew Kensler (2006)
with corrections by author of XRT Raytracer <http://xrt.wikidot.com/doc:csg> □

[2] https://bitbucket.org/simoncblyth/opticks/src/master/optixrap/cu/csg_intersect_boolean.h □
Similar to binary expression tree evaluation using postorder traverse.

Outside/Inside Unions

dot(normal,rayDir) -> Enter/Exit



- **A + B** boundary not inside other
- **A * B** boundary inside other

CSG Complete Binary Tree Serialization -> simplifies GPU side

Geant4 solid -> CSG binary tree (leaf primitives, non-leaf operators, 4x4 transforms on any node)

Serialize to **complete binary tree** buffer:

- no need to deserialize, no child/parent pointers
- bit twiddling navigation **avoids recursion**
- simple approach profits from small size of binary trees
- BUT: very inefficient when unbalanced

Height 3 complete binary tree with level order indices:

				depth	elevation				
	1			0	3				
10		11		1	2				
100	101	110	111	2	1				
1000	1001	1010	1011	1100	1101	1110	1111	3	0

postorder_next(i,elevation) = i & 1 ? i >> 1 : (i << elevation) + (1 << elevation) ; // from pattern of bits

Bit Twiddling Navigation

- $\text{parent}(i) = i/2 = i \gg 1$
- $\text{leftchild}(i) = 2*i = i \ll 1$
- $\text{rightchild}(i) = 2*i + 1 = (i \ll 1) + 1$
- $\text{leftmost}(\text{height}) = 1 \ll \text{height}$

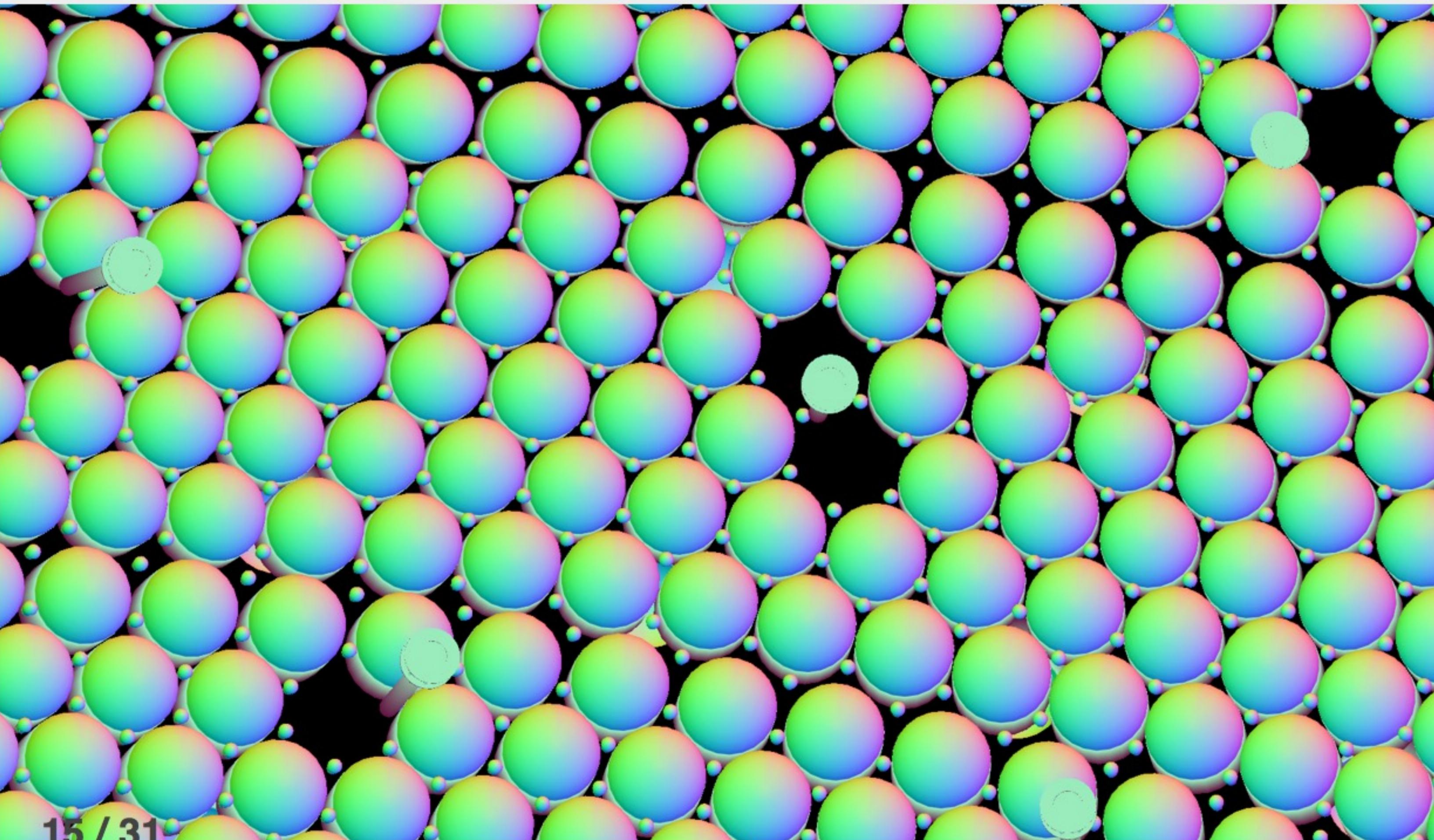
Opticks : Translates G4 Geometry to GPU, Without Approximation

G4 Structure Tree -> Instance+Global Arrays -> OptiX

Group structure into repeated instances + global remainder:

- auto-identify repeated geometry with "progeny digests"
 - JUNO : 5 distinct instances + 1 global
- instance transforms used in OptiX/OpenGL geometry

instancing -> huge memory savings for JUNO PMTs



Materials/Surfaces -> GPU Texture

Material/Surface/Scintillator properties

- interpolated to standard wavelength domain
- interleaved into "boundary" texture
- "reemission" texture for wavelength generation

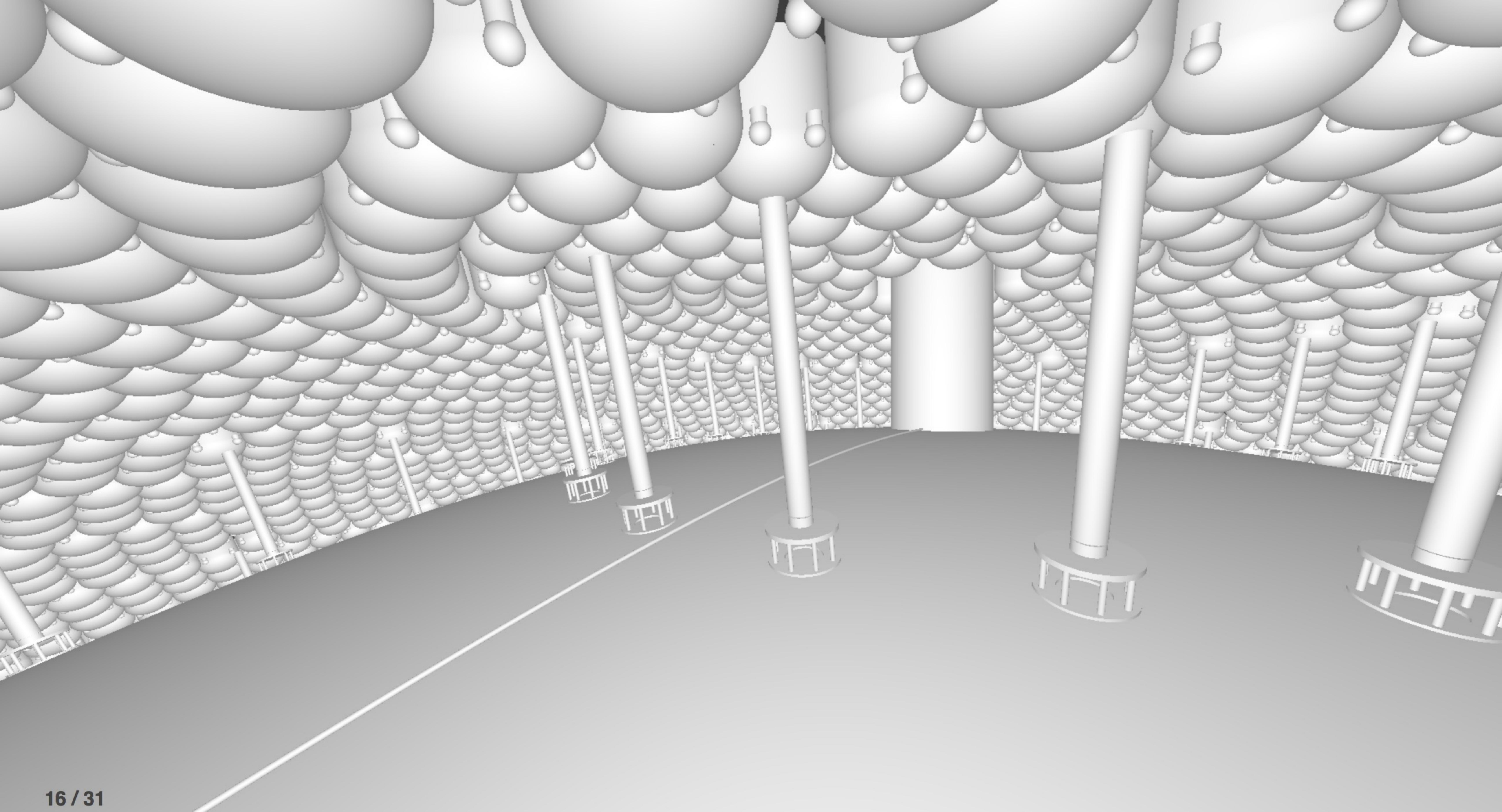
Material/surface boundary : 4 indices

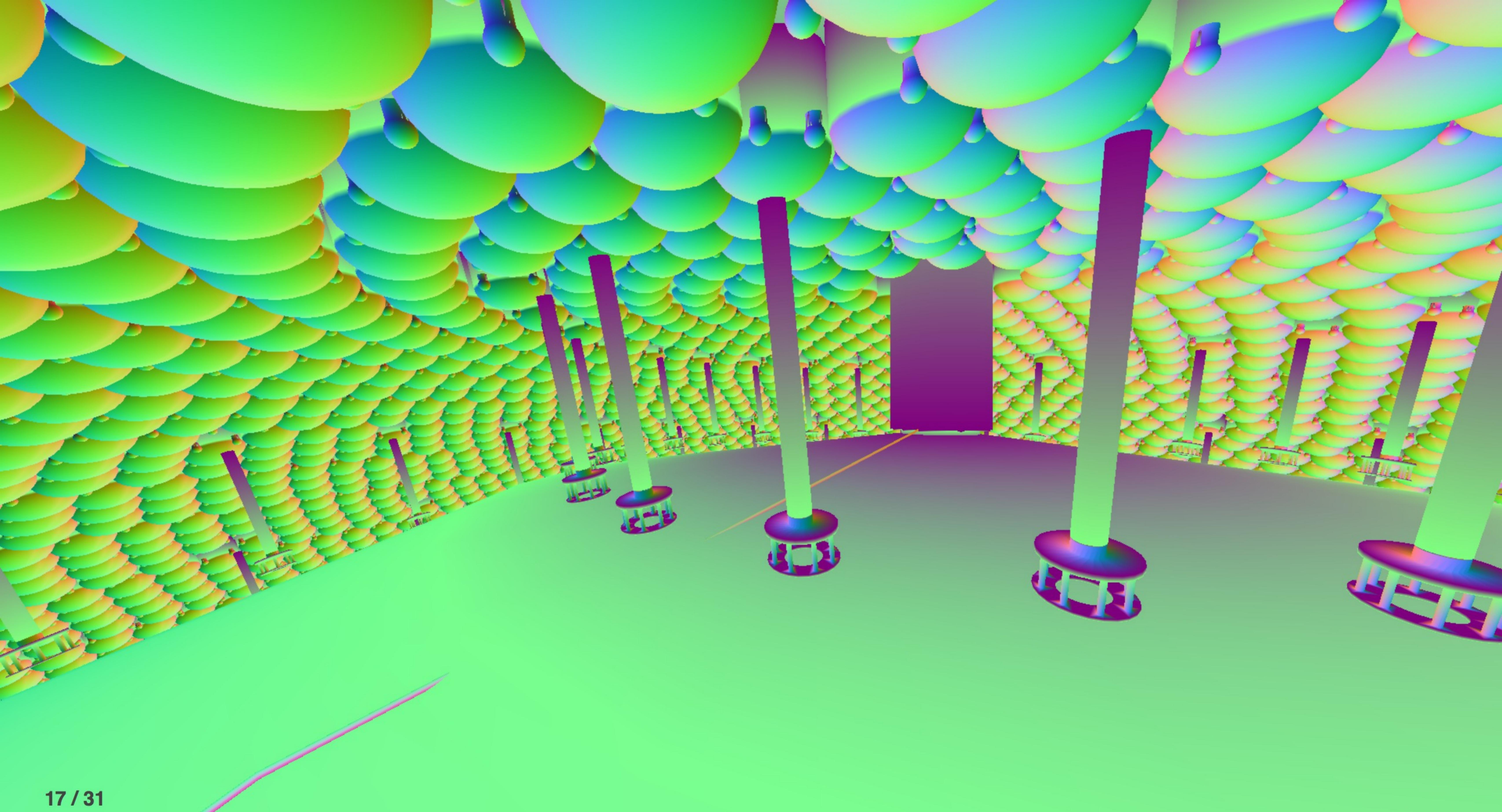
- outer material (parent)
- outer surface (inward photons, parent -> self)
- inner surface (outward photons, self -> parent)
- inner material (self)

Primitives labelled with unique boundary index

- ray primitive intersection -> boundary index
- texture lookup -> material/surface properties

simple/fast properties + reemission wavelength





Validation of Opticks Simulation by Comparison with Geant4

Bi-simulations of all JUNO solids, with millions of photons

mis-aligned histories

mostly < 0.25%, < 0.50% for largest solids

deviant photons within matched history

< 0.05% (500/1M)

Primary sources of problems

- grazing incidence, edge skimmers
- incidence at constituent solid boundaries

Primary cause : float vs double

Geant4 uses *double* everywhere, *Opticks* only sparingly (observed *double* costing 10x slowdown with RTX)

Conclude

- neatly oriented photons more prone to issues than realistic ones
- perfect "technical" matching not feasible
- instead shift validation to more realistic full detector "calibration" situation

Random Aligned Bi-Simulation

Same inputs to *Opticks* and *Geant4*:

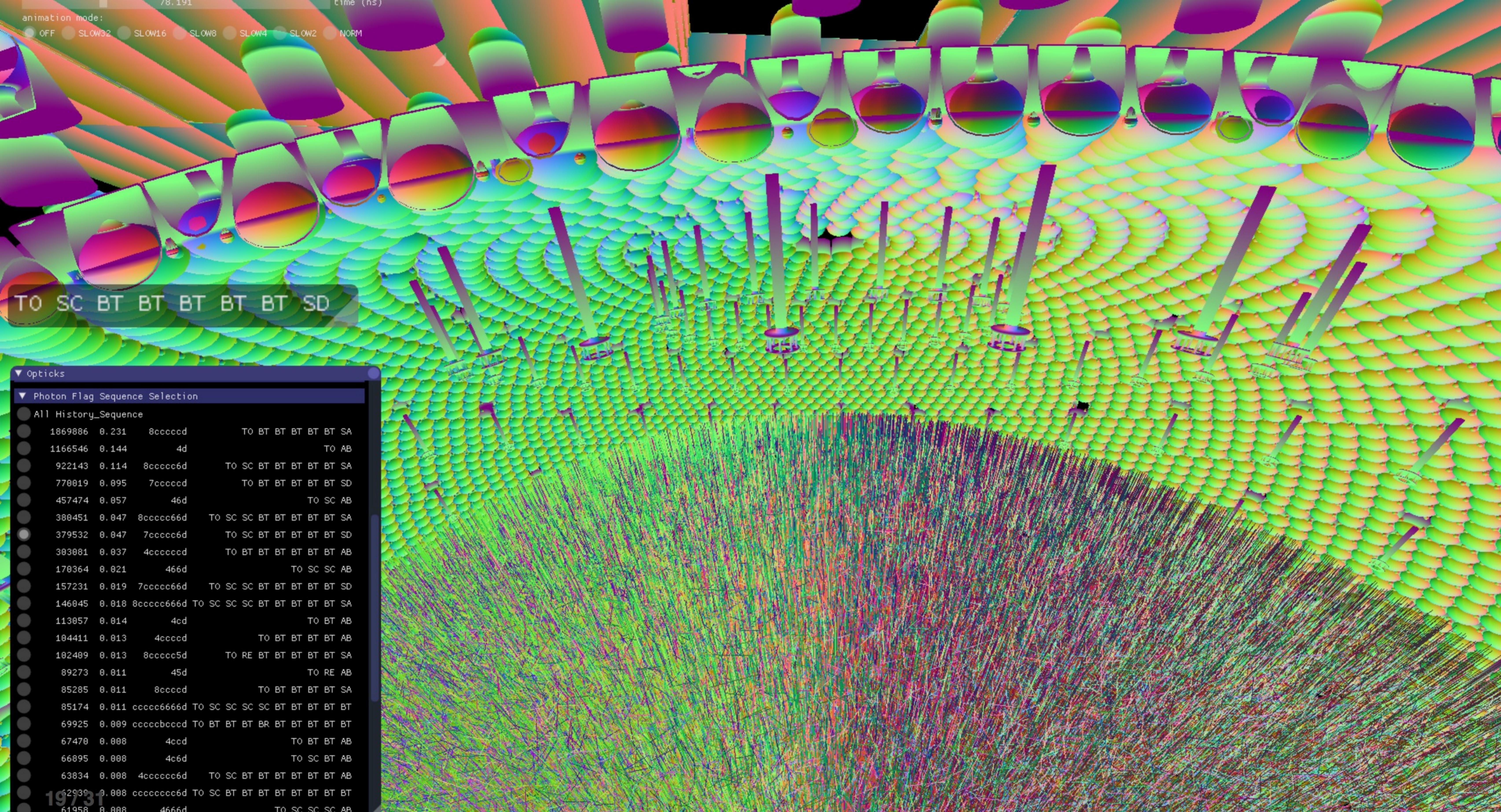
- CPU generated photons
- GPU generated randoms, fed to *Geant4*

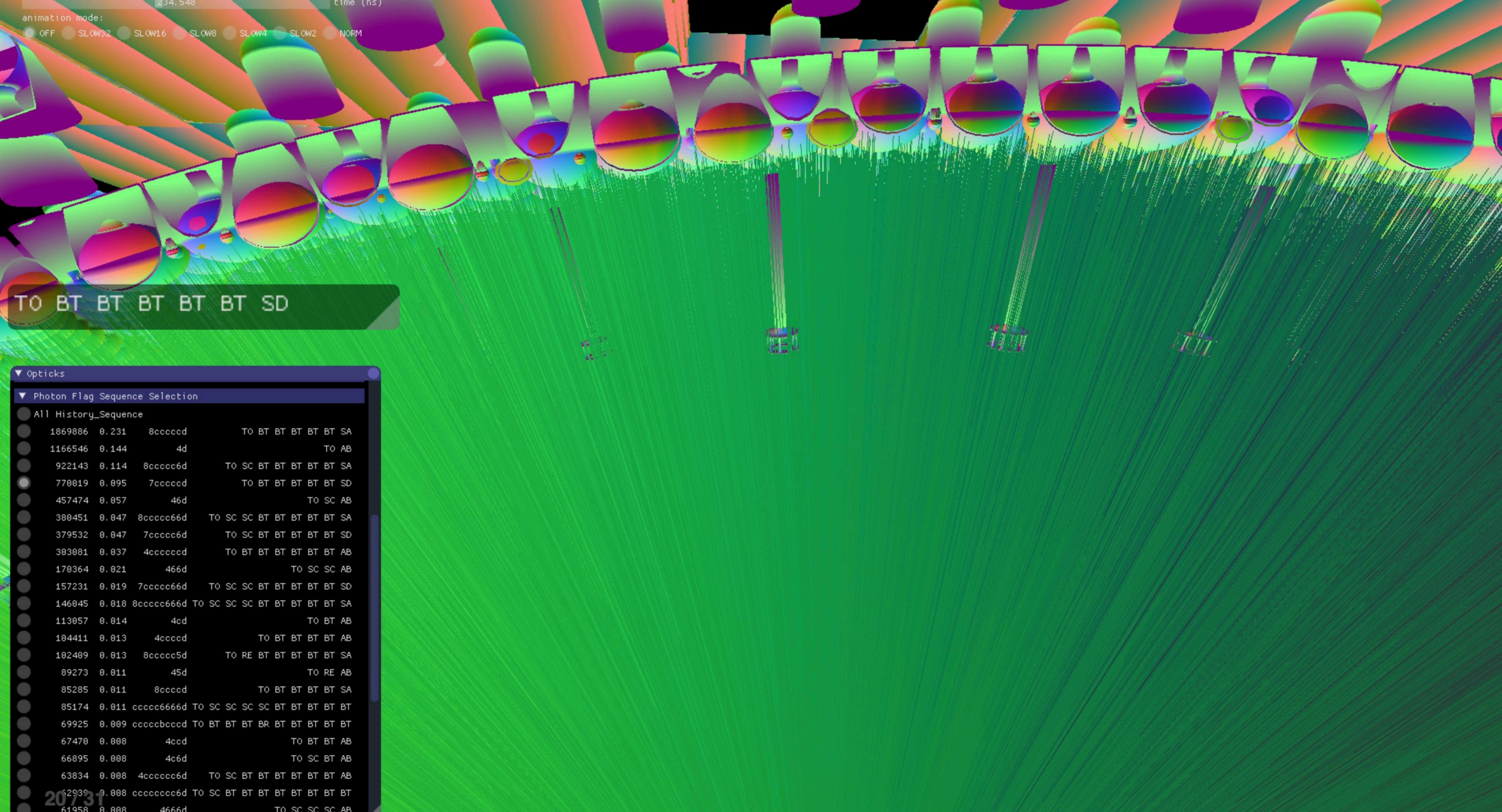
Common recording into *OpticksEvents*:

- compressed photon step record, up to 16 steps
- persisted as *NumPy* arrays for python analysis

Aligned random consumption, direct comparison:

- ~every **scatter, absorb, reflect, transmit** at matched positions, times, polarization, waven





Performance : Scanning from 1M to 400M Photons

Full JUNO Analytic Geometry j1808v5

- "calibration source" genstep at center of scintillator

Production Mode : does the minimum

- only saves hits
- skips : genstep, photon, source, record, sequence, index, ..
- no *Geant4* propagation (other than at 1M for extrapolation)

Multi-Event Running, Measure:

interval

avg time between successive launches, including overheads:
(upload gensteps + **launch** + download hits)

launch

avg of 10 OptiX launches

- overheads < 10% beyond 20M photons

Test Hardware + Software

Workstation

- DELL Precision 7920T Workstation
- Intel Xeon Gold 5118, 2.3GHz, 48 cores, 62G
- NVIDIA Quadro RTX 8000 (48G)

Software

- Opticks 0.0.0 Alpha
- Geant4 10.4p2
- NVIDIA OptiX 6.5.0
- NVIDIA Driver 435.21
- CUDA 10.1

IHEP GPU Cluster

- 10 nodes of 8x NVIDIA Tesla GV100 (32G)

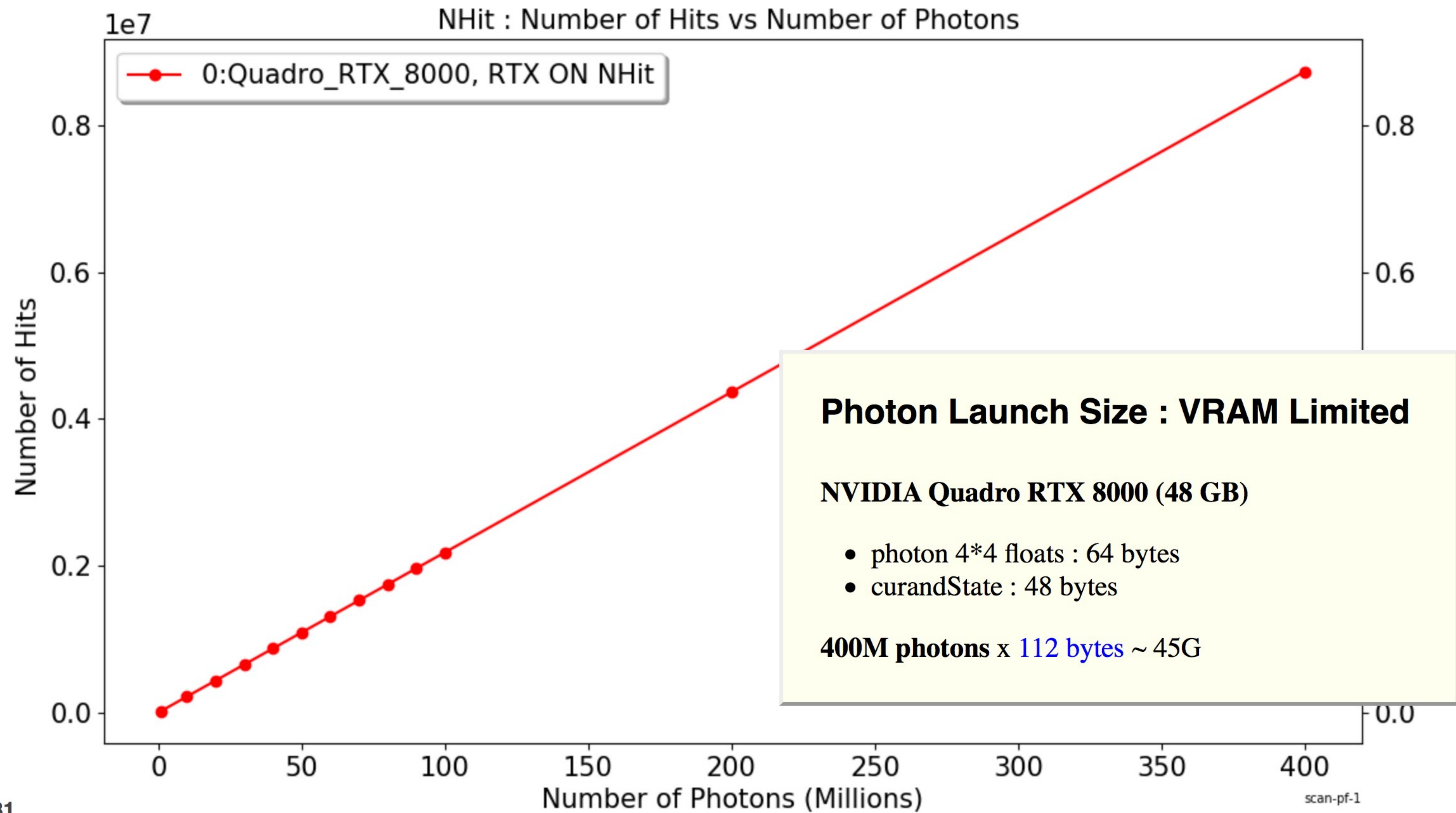
NVIDIA Quadro RTX 8000 (48G)



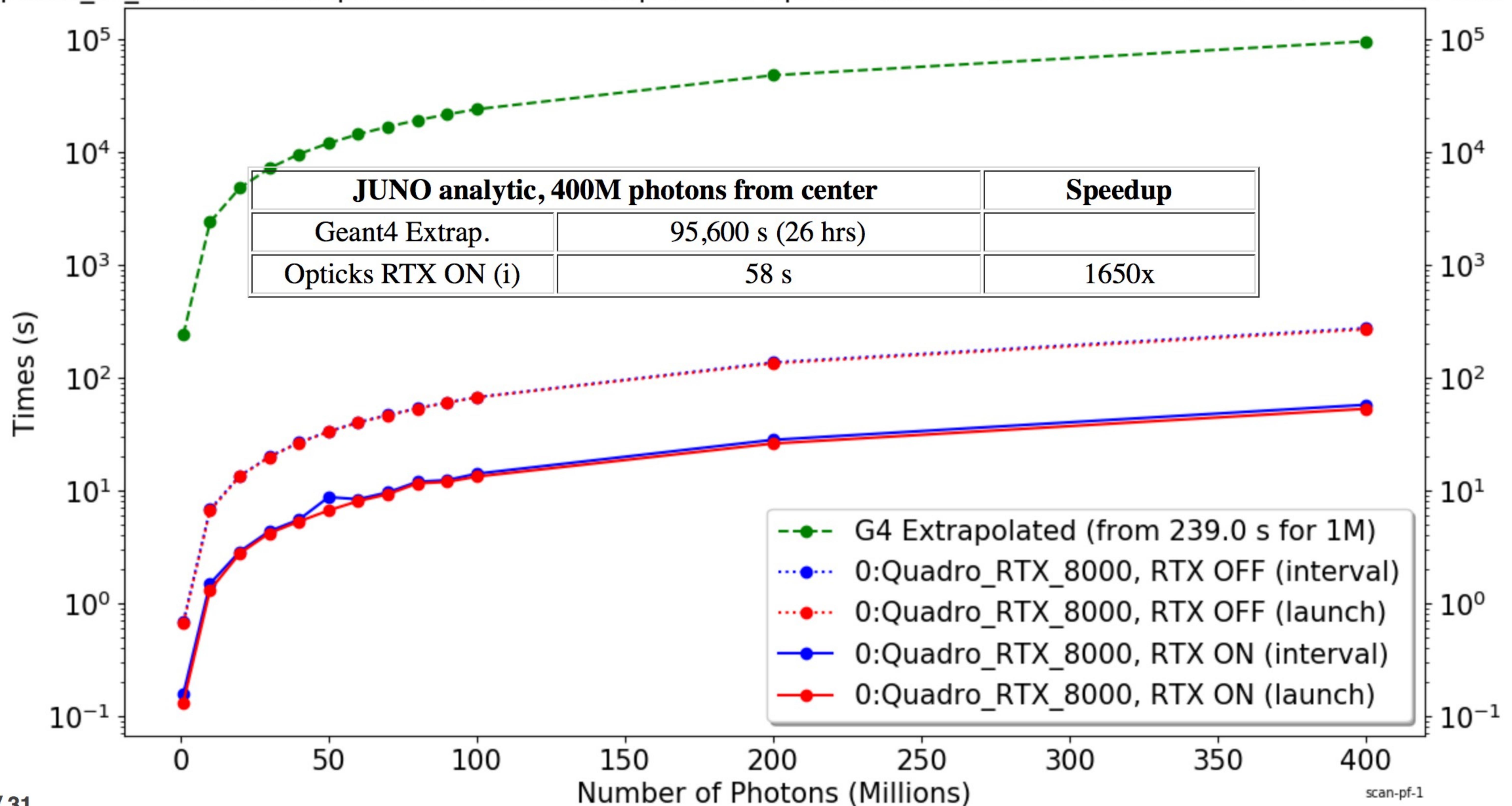
RTX



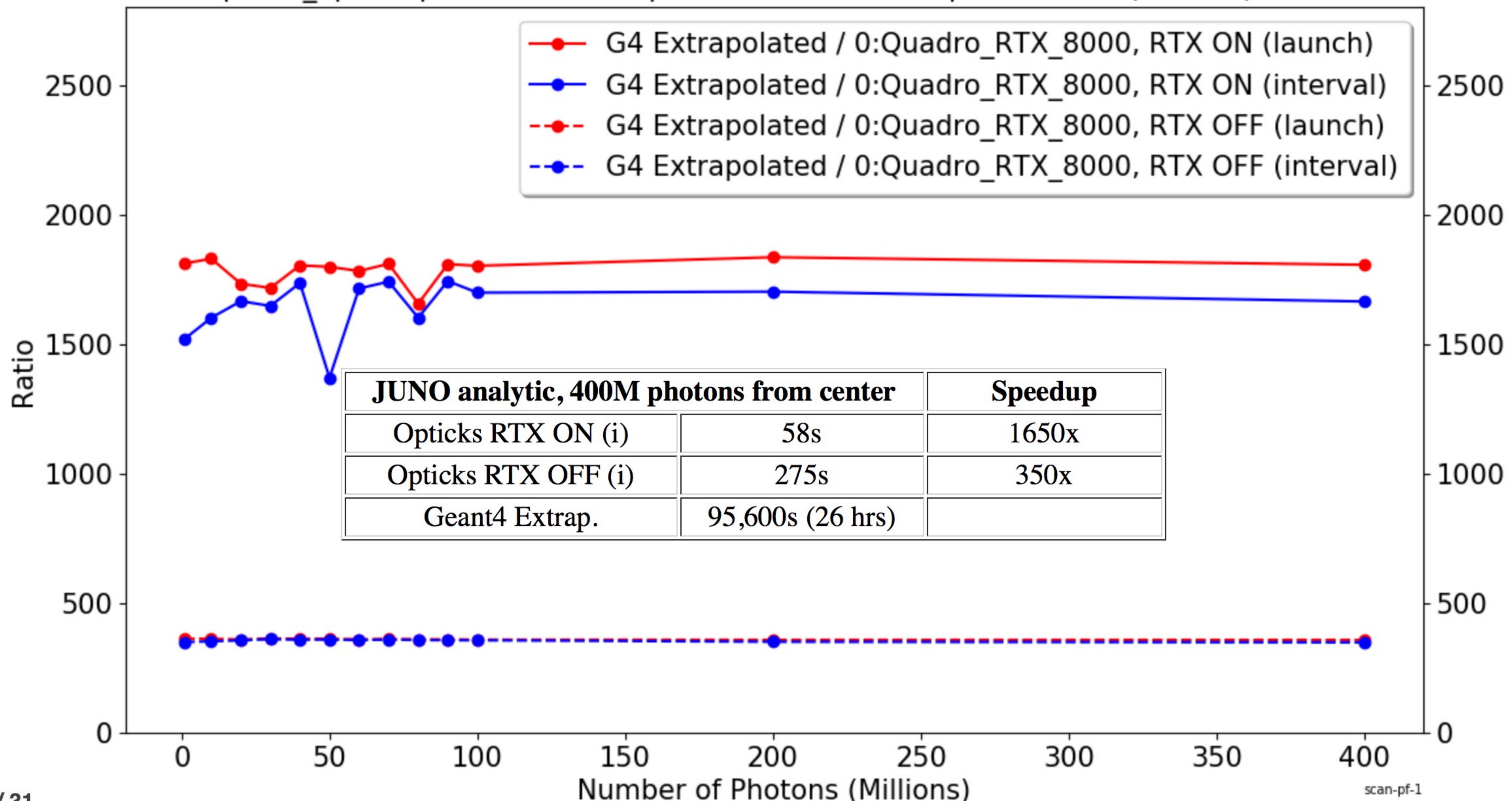
谢谢 NVIDIA China
for loaning the card

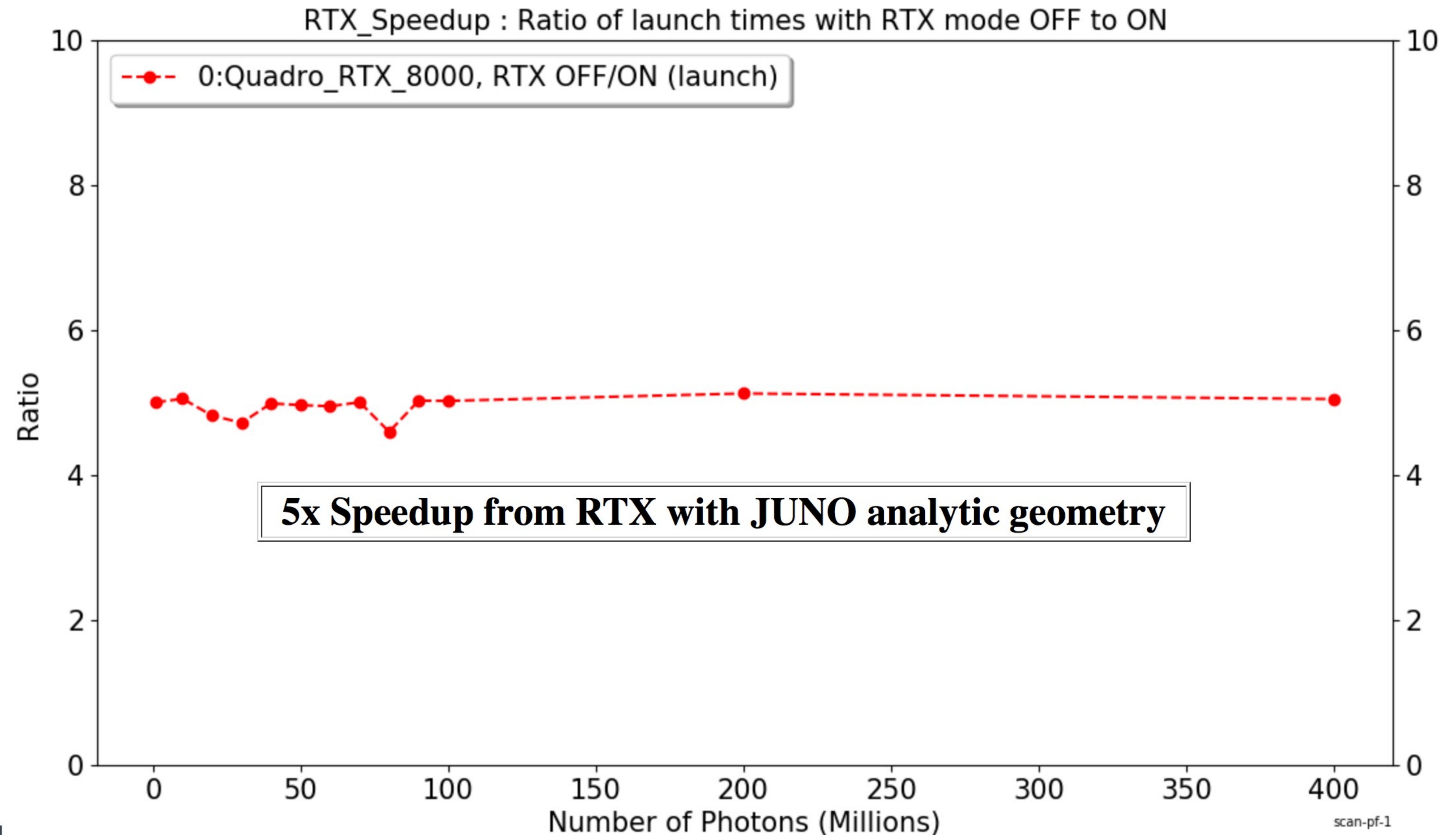


Opticks_vs_Geant4 : Extrapolated G4 times compared to Opticks launch+interval times with RTX mode ON and OFF



Opticks_Speedup : Ratio of extrapolated G4 times to Opticks launch(interval) times





Opticks Experience : Main Operational Problem : Manpower

Lots of interest, very little contribution, why ?

Tool Innovation is Disincentivized ?

- students/postdocs interested in Opticks
- advisors steer them to analysis : less risky, better for career

Why GPU simulation development difficult ?

- totally different geometry model
 - ~~tree of C++ objects~~ -> arrays, textures
 - ~~solid primitives~~ -> intersection by solving polynomials
- totally different development model
 - ~~complex libraries~~ -> simple headers
 - simpler -> smaller stack -> more threads in flight
 - low level CUDA development, eg CSG from first principals
 - very few libs
 - restricted CUDA environment
 - no recursion in intersect
 - no shared memory/synchronizations/barriers
 - double precision problematic, performance hit

Atypical Experience Needed

- ~~difficult for students/postdocs to help~~
- Opticks is a one person project

Typical NVIDIA OptiX Users:

- **software company teams** building commercial renderers for design/architecture/visual-effects
- experienced graphics developers
- Windows workstations

Typical Opticks Users (Candidate Devs):

- neutrino and dark matter search experiments with 0-0.25 FTE trying to improve Geant4 simulation performance
- no graphics experience, little GPU experience
- Linux servers

Opticks Experience : Main Technical Problem : Geometry Translation

Intersection Performance -> Simulation Performance, Drivers:

- acceleration structure (AS) eg BVH
- geometry model input to AS

Analytic Geometry : translate volume -> surface based model

Coincident faces (even in CSG boolean constituents)

- very common problem, causes spurious intersects
 - manual modelling changes : avoiding coincidence
- CSG serialized using complete binary tree
 - simple+convenient, **very inefficient for unbalanced trees**
 - balancing enables support for more complex trees
 - v.complicated solids (G4Boolean abuse) still problematic

Analytic Torus Intersection

- double precision quartic root solving
- very large coefficient range, robust solution difficult
 - many techniques tried (numerical/computer science papers)
- very heavy kernel : 10x performance impact even when unused
- pragmatic solution : avoid torus, or use triangulated

Approximate Triangulated geometry

GPUs love triangles, BUT polygonization not easy

- G4Polyhedron issue (black box code)
 - polycones yield cleaved meshes
 - just visualization issue for Geant4
 - but breaks the geometry
- Opticks(OpenMesh) fixes meshes, BUT fragile
- Failed to find/develop better polygonization
- Triangulation too approximate for PMTs
- Approx. tris : limits Opticks-Geant4 matching

Opticks Experience : Problems with using NVIDIA OptiX

- NVIDIA GPUs only
- No influence on direction of NVIDIA OptiX (eg OptiX 7)
- immature OptiX support for Linux debug/profiling
 - most OptiX users develop on Windows ?
 - cuda-gdb not to level of Nsight VSE

Optimization Issues

- difficult to approach 10 GigaRays/s
- closed source : black box BVH
 - blind experimentation

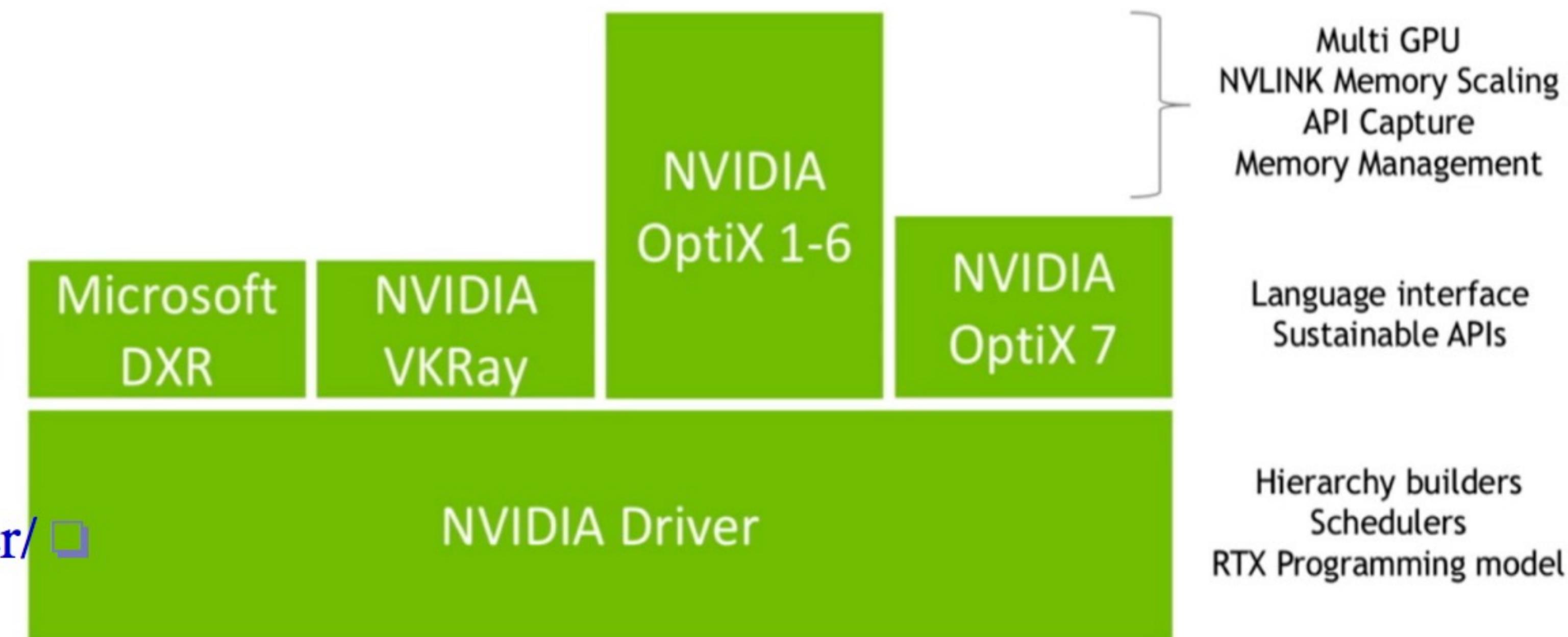
Linux GPU Cluster (eg Tesla V100) Deployment Issues

- OptiX releases demand lastest **short lived branch** driver
- GPU clusters typically use **long lived branch** drivers
 - drivers appropriate for OptiX slow to appear
 - must use older OptiX release for many months
- RTX performance, RT cores ~not available in server GPUs[1,2]
 - 4 non-RTX GPUs ~ single RTX GPU performance
- <https://www.nvidia.com/en-us/design-visualization/quadro-data-center/>

NVIDIA OptiX 7 : Entirely new API

- introduced August 2019
- low-level CUDA-centric thin API (Vulkan-ized)
- ~~near perfect scaling to 4 GPUs, for free~~
- Major effort needed to support in Opticks

Introducing OptiX 7



[1] NVIDIA RTX Server with 8x NVIDIA Quadro RTX 8000 : probably restricted to car, design, film companies ... [2] NVIDIA Quadro RTX 8000 PCIe Server Card (Passive)

Opticks Experience : Benefits from using NVIDIA OptiX

NVIDIA OptiX 3,4,5,6

- excellent easy to use API
- useful shared host/device context
- flexible geometry : implement intersection code
- automated acceleration structure (AS) building/traversal
- instancing of geometry + AS, essential for JUNO PMTs
- transparent ~linear scaling up to 4 GPUs
- graphics interop buffer sharing CUDA/Thrust/OptiX/OpenGL
 - in-situ visualization with no data movement
- straightforward port of Geant4 optical physics

NVIDIA OptiX 6

- accelerated with ray trace dedicated hardware (RT Cores)
 - 5x performance from RTX

1 or 2 Releases per Year

- until OptiX 7, fairly compatible API changes
- tuning for new GPUs

State-of-the-art GPU ray tracing

OptiX makes this performance accessible

- drastic performance factors > 1000x Geant4
- GPU offloading
 - eliminates memory, time bottlenecks
- substantial improvements with each release
- benefit from new GPU features

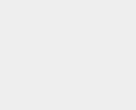
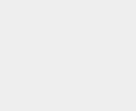
Summary

Opticks : state-of-the-art GPU ray tracing applied to optical photon simulation and integrated with *Geant4*, giving a leap in performance that eliminates memory and time bottlenecks.

- Drastic speedup -> better detector understanding -> greater precision
 - **any simulation limited by optical photons can benefit**
 - more photon limited -> more overall speedup (99% -> 100x)

Innovation = Problems...

- Problems Inevitable
- Ample benefits reward efforts

https://bitbucket.org/simoncblyth/opticks 	code repository
https://simoncblyth.bitbucket.io 	presentations and videos
https://groups.io/g/opticks 	forum/mailing list archive
email: opticks+subscribe@groups.io	subscribe to mailing list