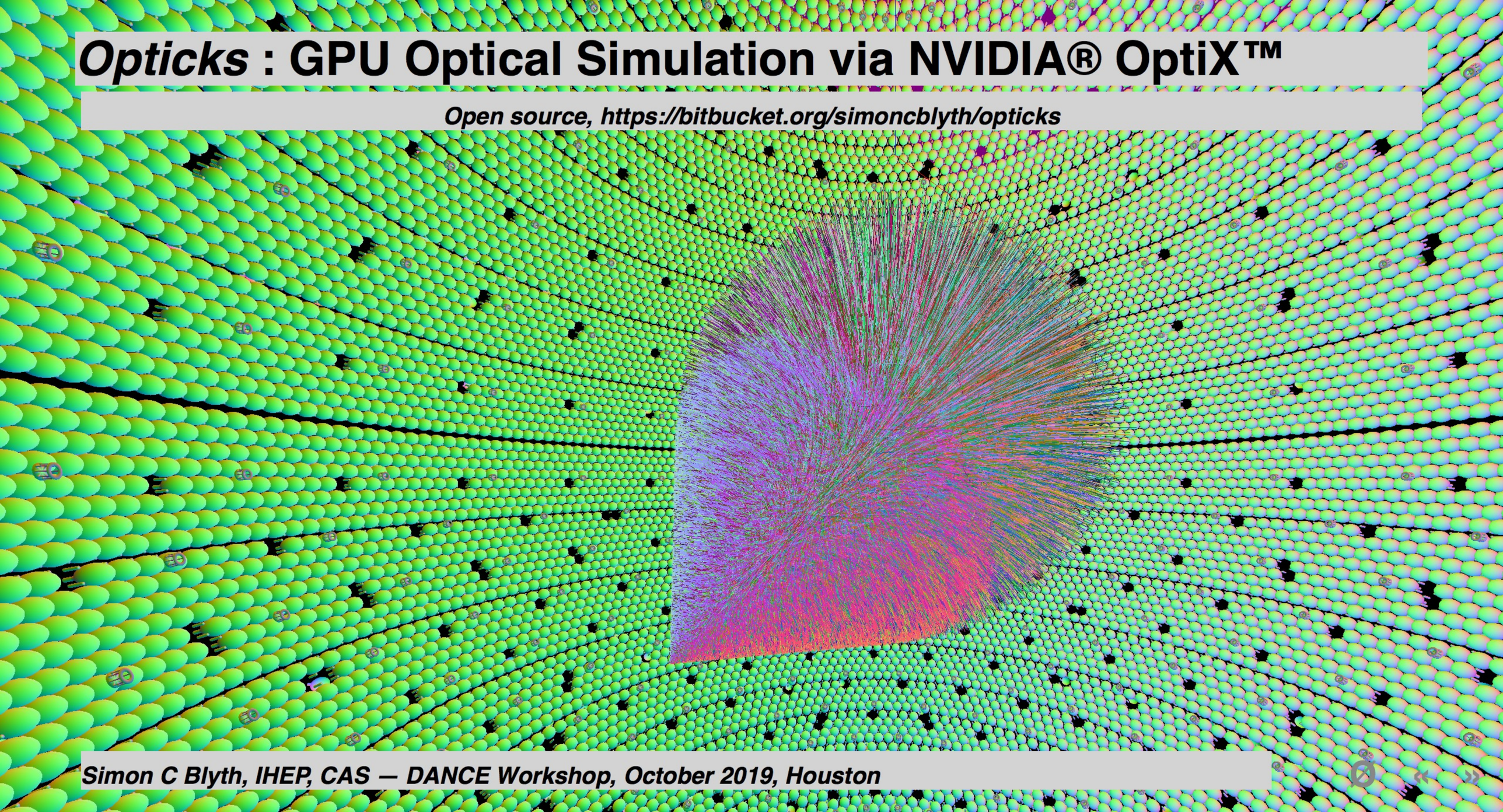
Opticks : GPU Optical Simulation via NVIDIA® OptiX™

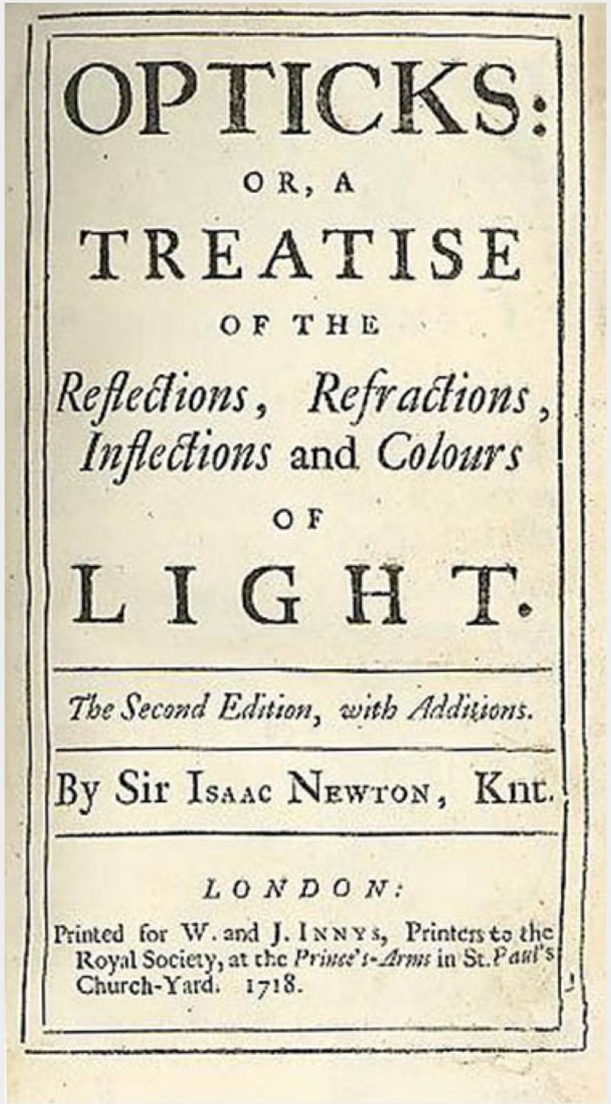
Open source, https://bitbucket.org/simoncblyth/opticks



Simon C Blyth, IHEP, CAS — DANCE Workshop, October 2019, Houston

Outline

- Optical Photon Simulation Problem...
- Tools to create Solution
  - Optical Photon Simulation ≈ Ray Traced Image Rendering
  - Rasterization and Ray tracing
  - Turing Built for RTX
  - BVH : Bounding Volume Hierarchy
  - NVIDIA OptiX Ray Tracing Engine
- Opticks : The Solution
  - Geant4 + Opticks Hybrid Workflow : External Optical Photon Simulation
  - Opticks : Translates G4 Optical Physics to CUDA/OptiX
  - Opticks : Translates G4 Geometry to GPU, Without Approximation
  - CUDA/OptiX Intersection Functions for ~10 Primitives
  - CUDA/OptiX Intersection Functions for Arbitrarily Complex CSG Shapes
- Validation and Performance
  - Random Aligned Bi-Simulation -> Direct Array Comparison
  - Perfomance Scanning from 1M to 400M Photons
- Overview + Links



OPTICKS:

OR, A

TREATISE

OF THE

Reflections, Refractions, Inflections and *Colours*

OF

LIGHT.

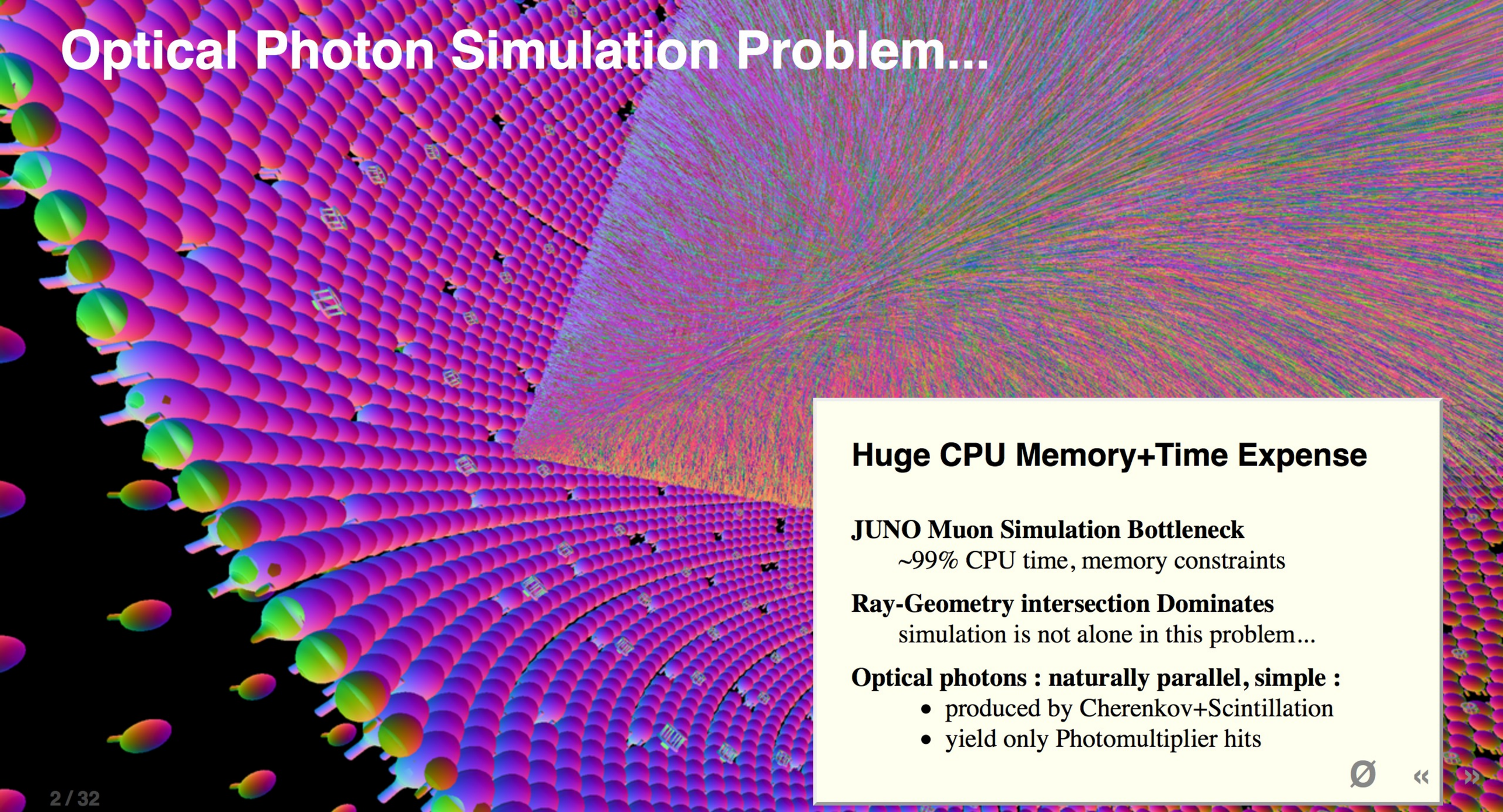
The Second Edition, with Additions.

By Sir Isaac Newton, Knt.

LONDON:

Printed for W. and J. Innys, Printers to the Royal Society, at the *Prince's-Arms* in St. *Paul's* Church-Yard. 1718.

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

Optical Photon Simulation ≈ Ray Traced Image Rendering

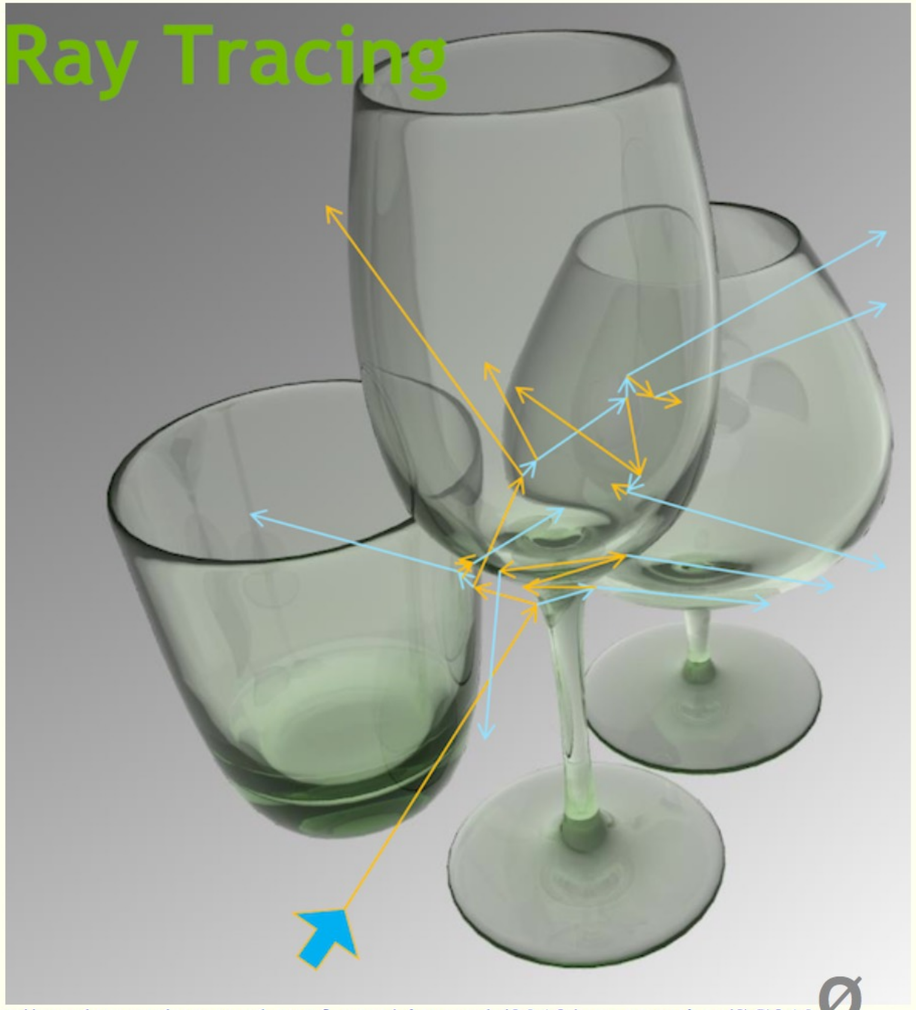
Much in common : geometry, light sources, optical physics

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

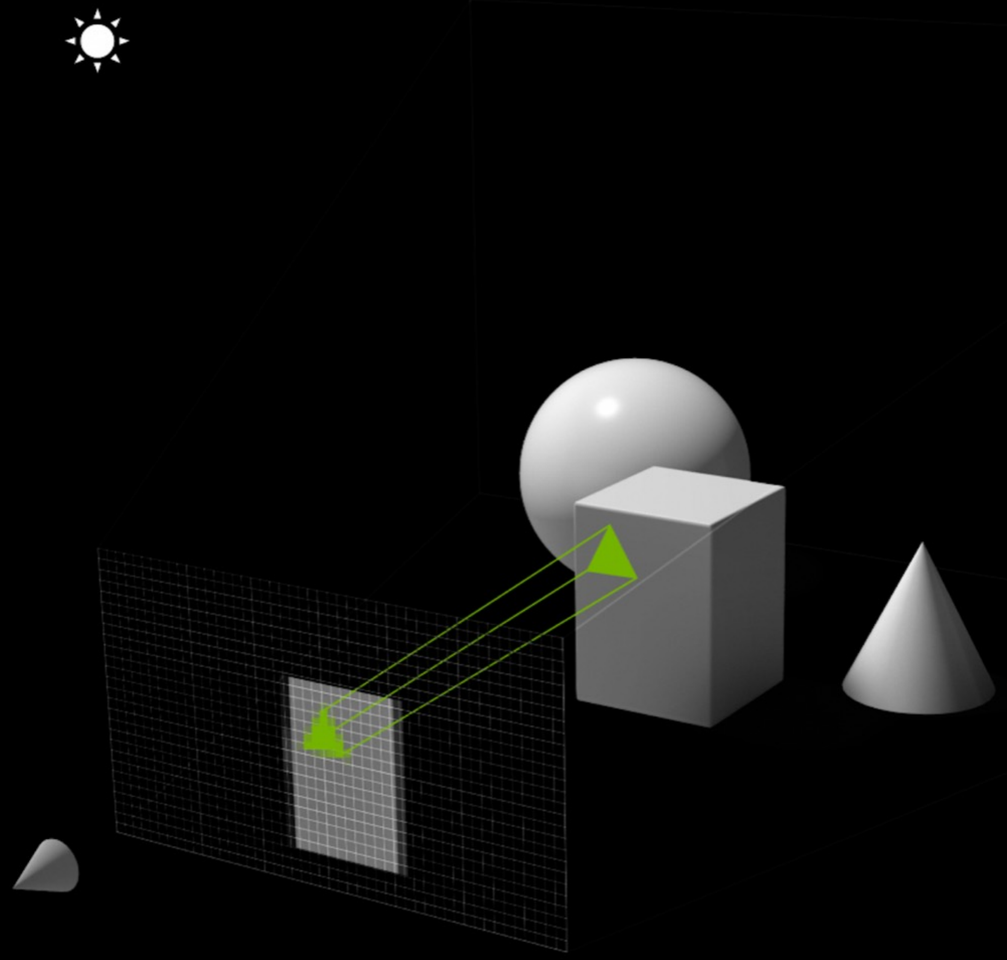
Many Applications of ray tracing :

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

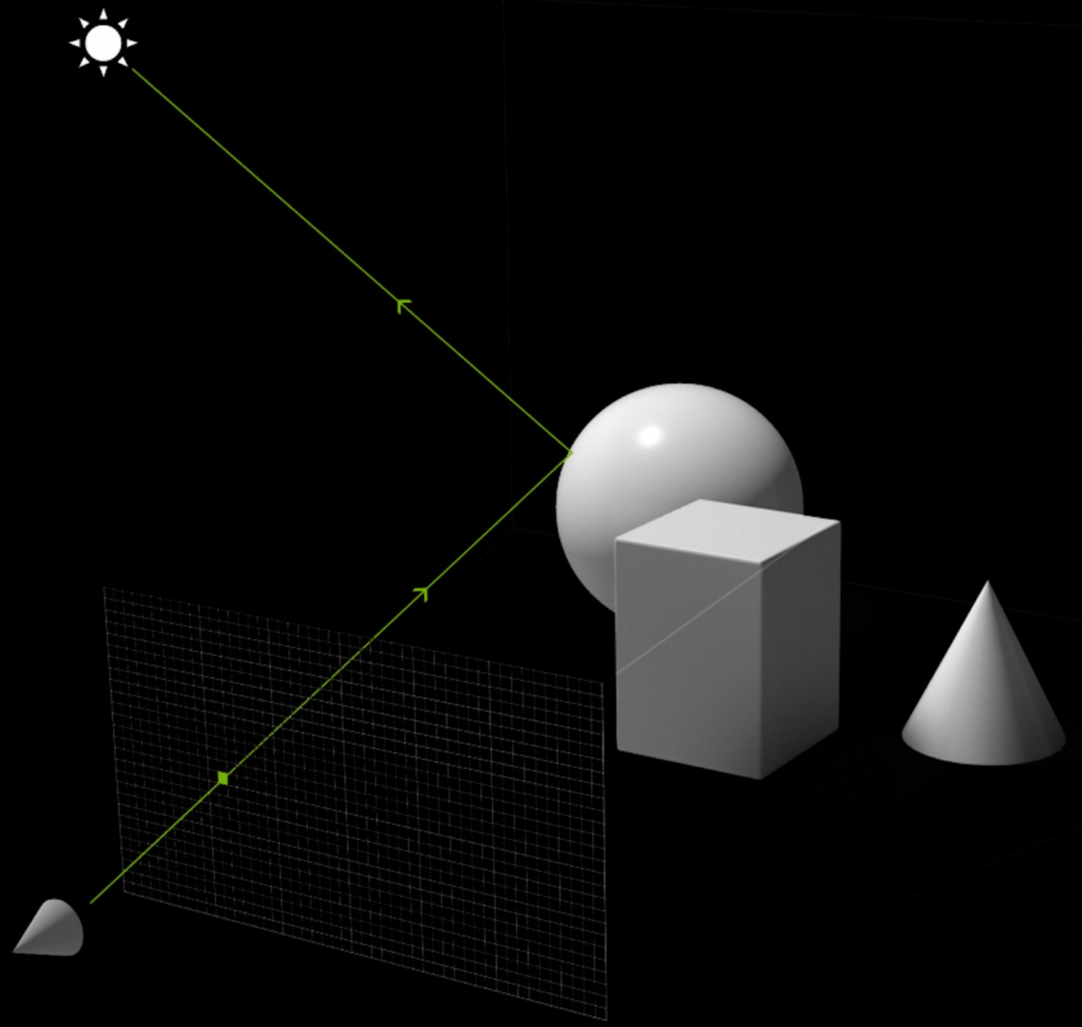
Not a Photo, a Calculation



http://on-demand.gputechconf.com/siggraph/2013/presentation/SG3106-Building-Ray-Tracing-Applications-OptiX.pdf



RASTERIZATION



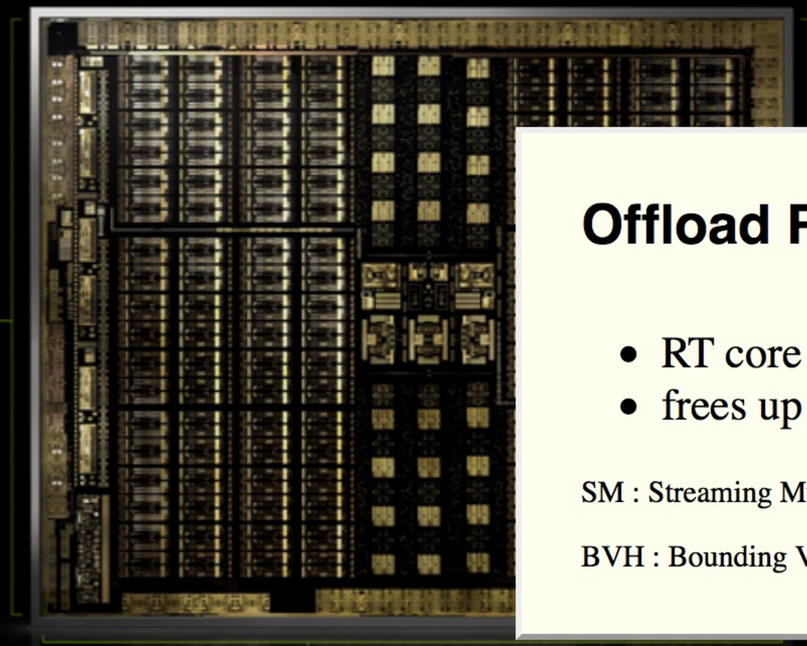
RAY TRACING

SIGGRAPH 2018

TURING BUILT FOR RTX

GREATEST LEAP SINCE 2006 CUDA GPU

Turing SM
14 TFLOPS + 14 TIPS
Concurrent FP & INT Execution
Variable Rate Shading



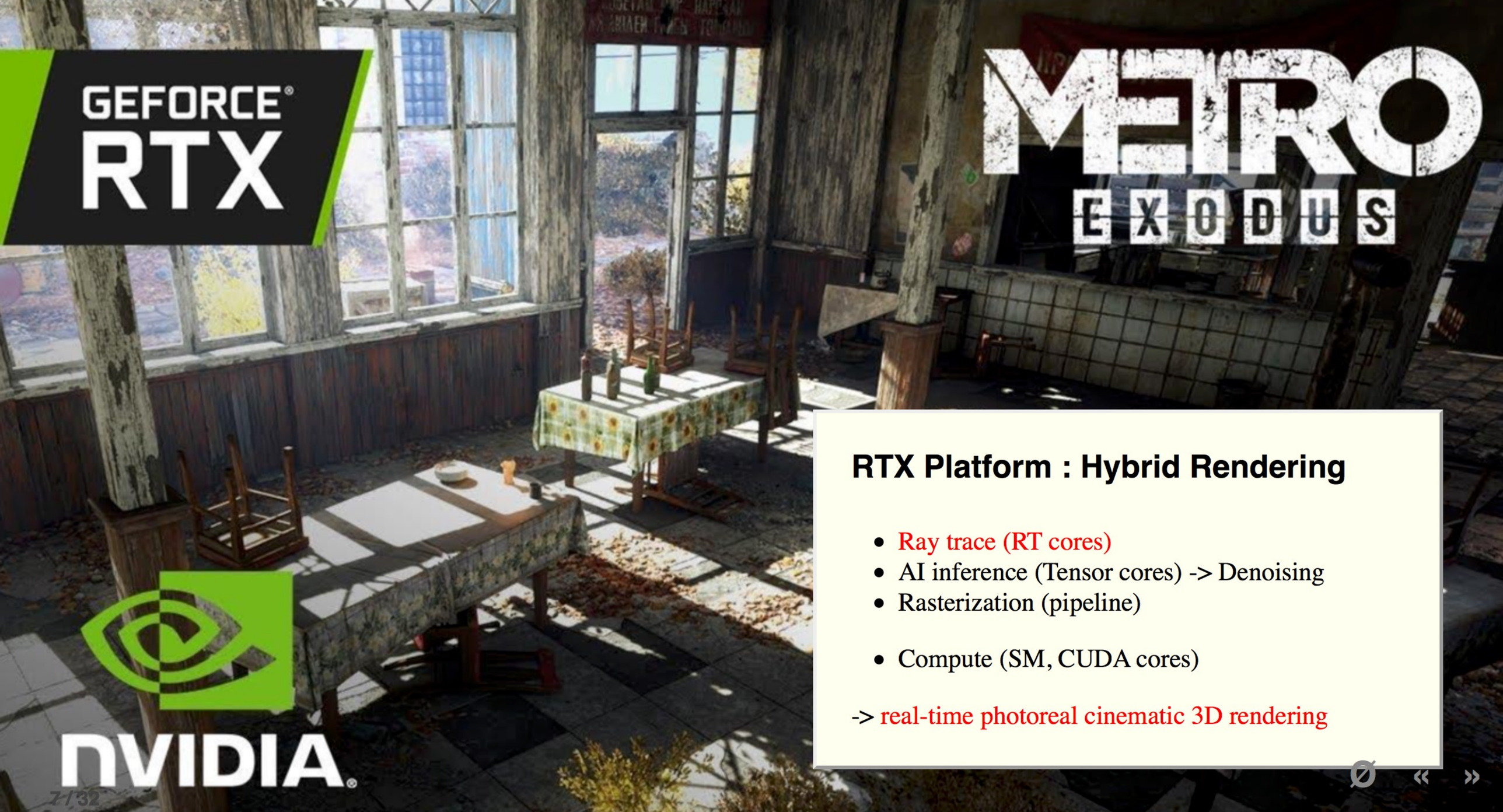
RT Core
10 Giga Rays/sec
Ray Triangle Intersection
BVH Traversal

Offload Ray Trace to Dedicated HW

- RT core : BVH traversal + ray tri. intersection
- frees up general purpose SM

SM : Streaming Multiprocessor

BVH : Bounding Volume Hierarchy



RTX Platform : Hybrid Rendering

- Ray trace (RT cores)
- AI inference (Tensor cores) -> Denoising
- Rasterization (pipeline)

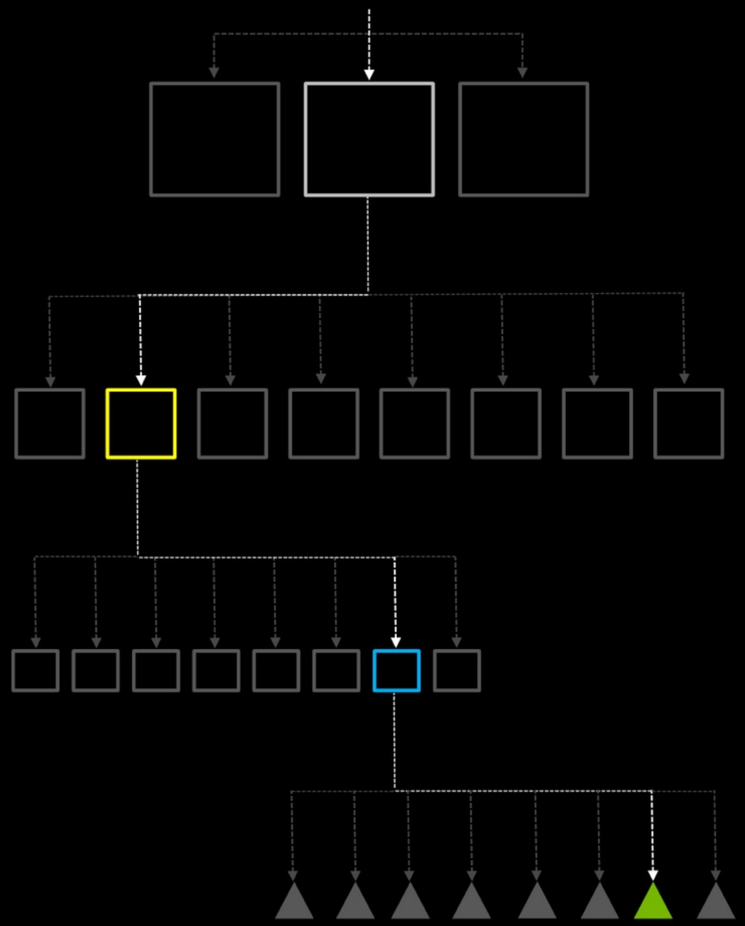
- Compute (SM, CUDA cores)

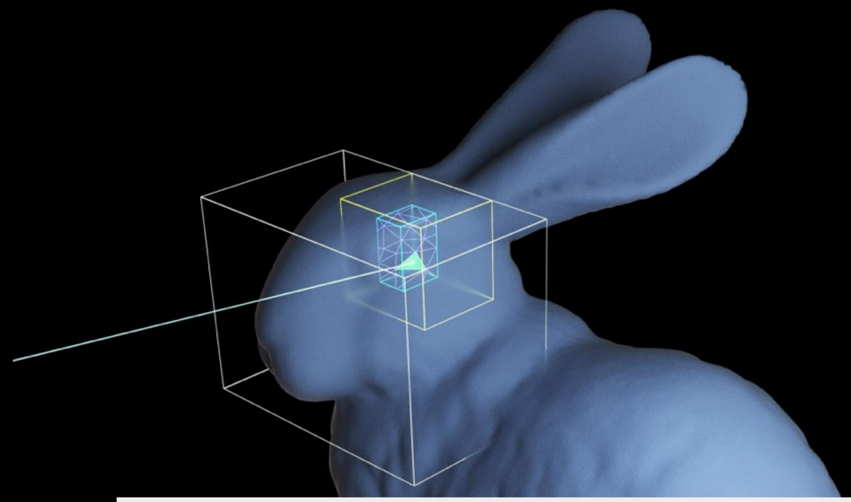
-> real-time photoreal cinematic 3D rendering

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:

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

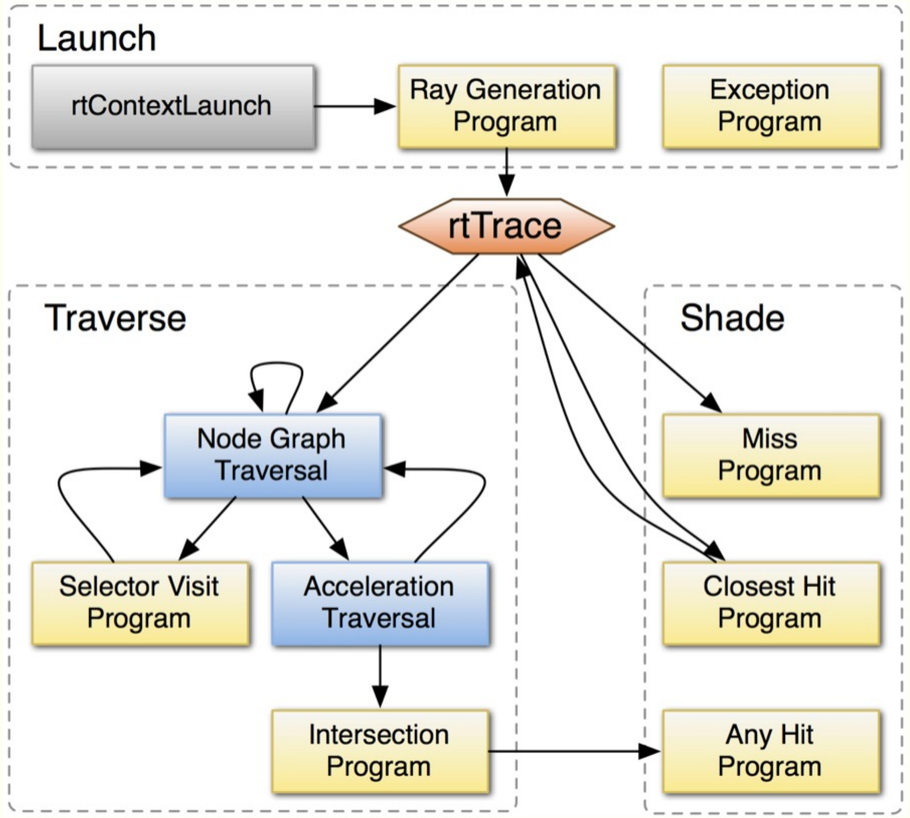
Opticks provides (Yellow):

- ray generation program
- ray geometry intersection+bbox programs

[1] Turing RTX GPUs

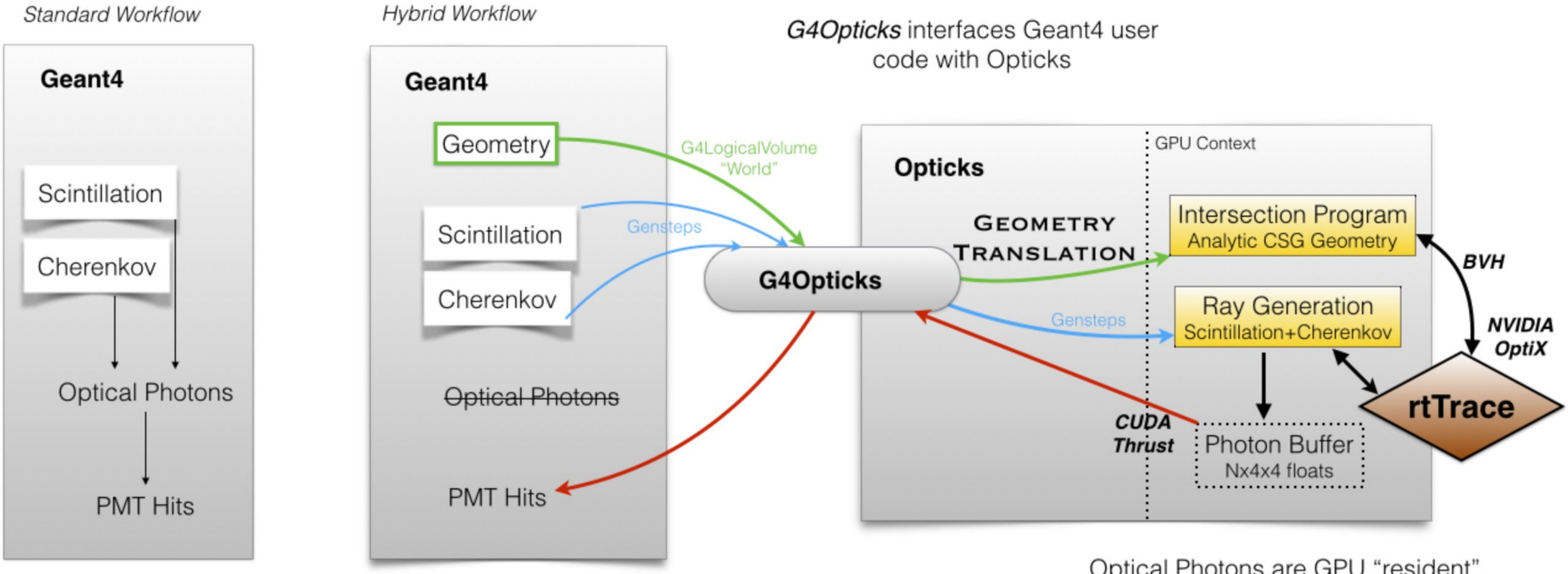
OptiX Raytracing Pipeline

Analogous to OpenGL rasterization pipeline:



Geant4 + Opticks Hybrid Workflow : External Optical Photon Simulation

https://bitbucket.org/simoncblyth/opticks



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)

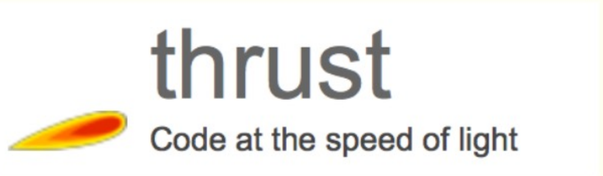
Generated on GPU, using genstep param:

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

Propagated on GPU

Only photons hitting PMTs copied to CPU

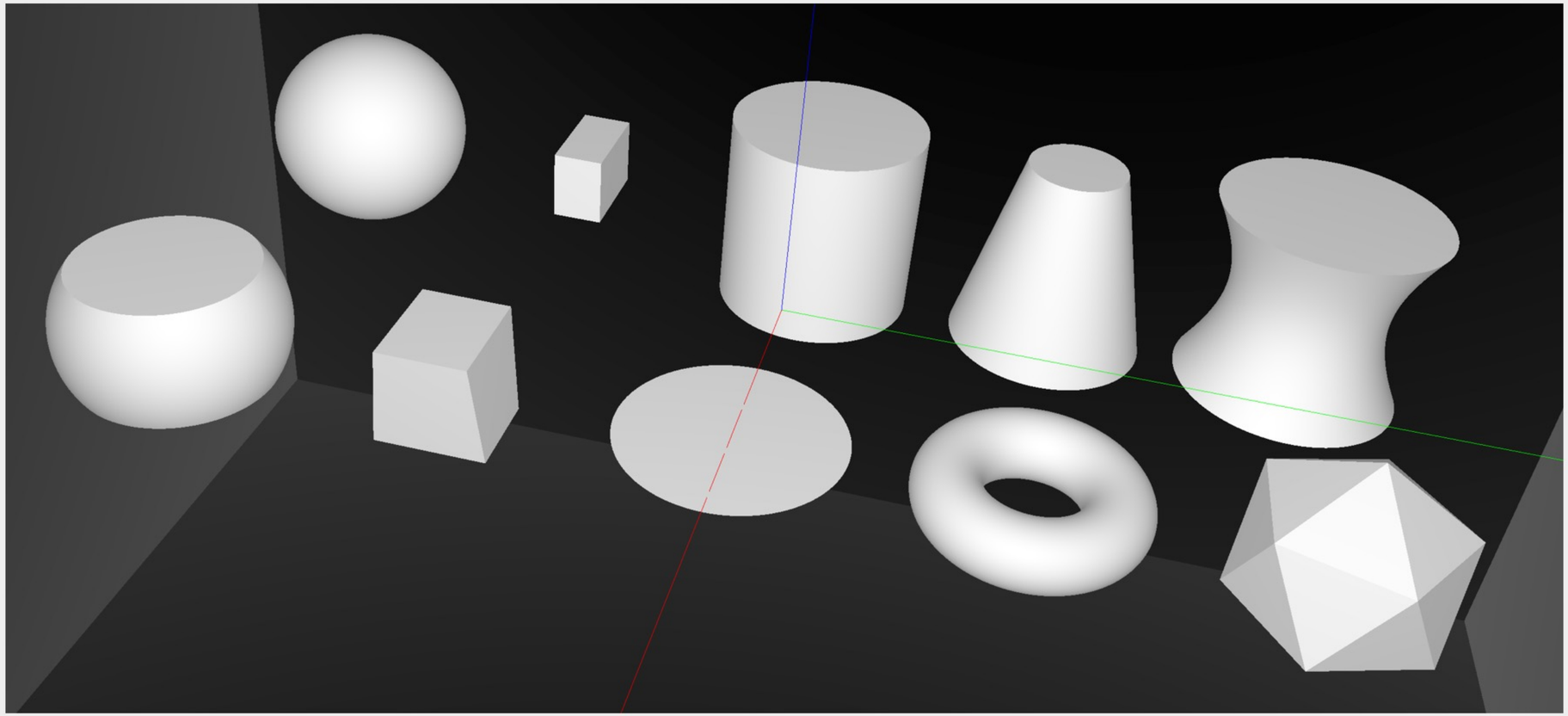
Thrust: **high level C++ access to CUDA**



- https://developer.nvidia.com/Thrust ❑

G4Solid -> CUDA Intersect Functions for ~10 Primitives

- 3D parametric ray : **ray(x,y,z;t) = rayOrigin + t * 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

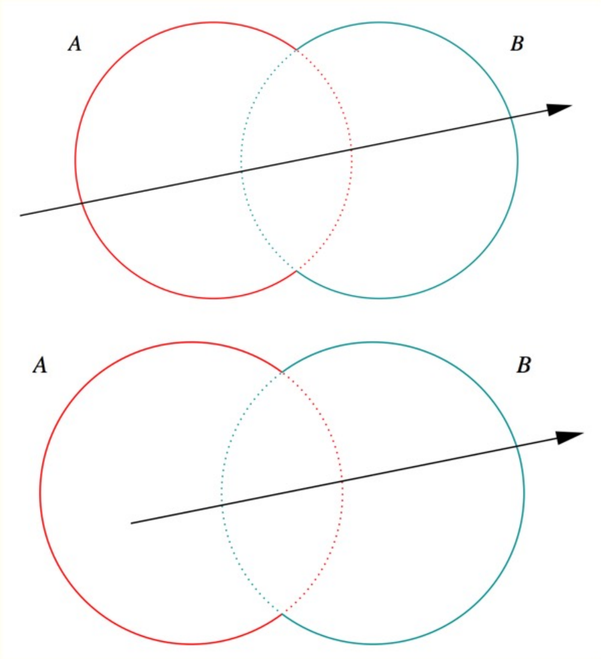
- *Nearest hit intersect algorithm* [1] avoids state
  - sometimes Loop : advance **t_min** , re-intersect both
  - classification shows if inside/outside
- *Evaluative* [2] implementation emulates recursion:
  - recursion not allowed in OptiX intersect programs
  - bit twiddle traversal of complete 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/tip/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

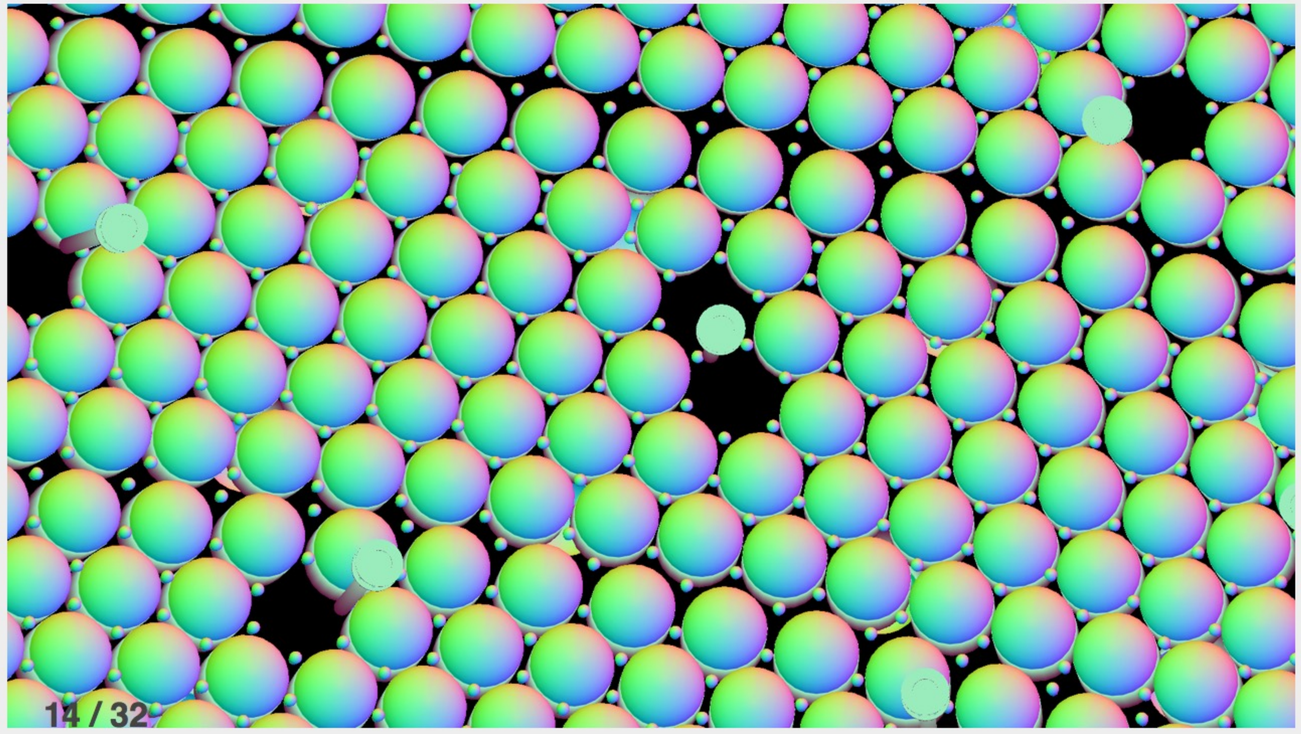
Opticks : Translates G4 Geometry to GPU, Without Approximation

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

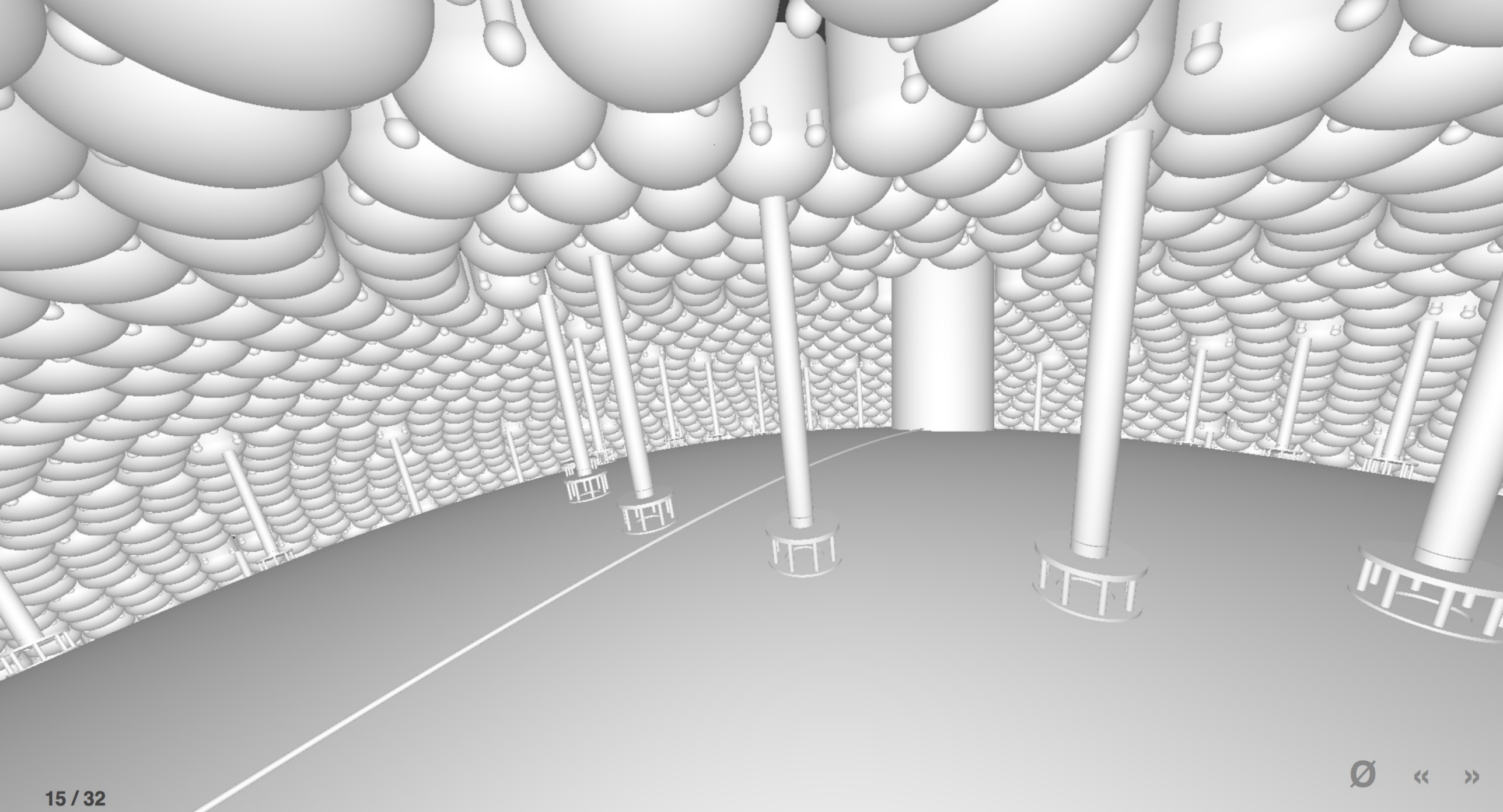
Material/surface boundary : 4 indices

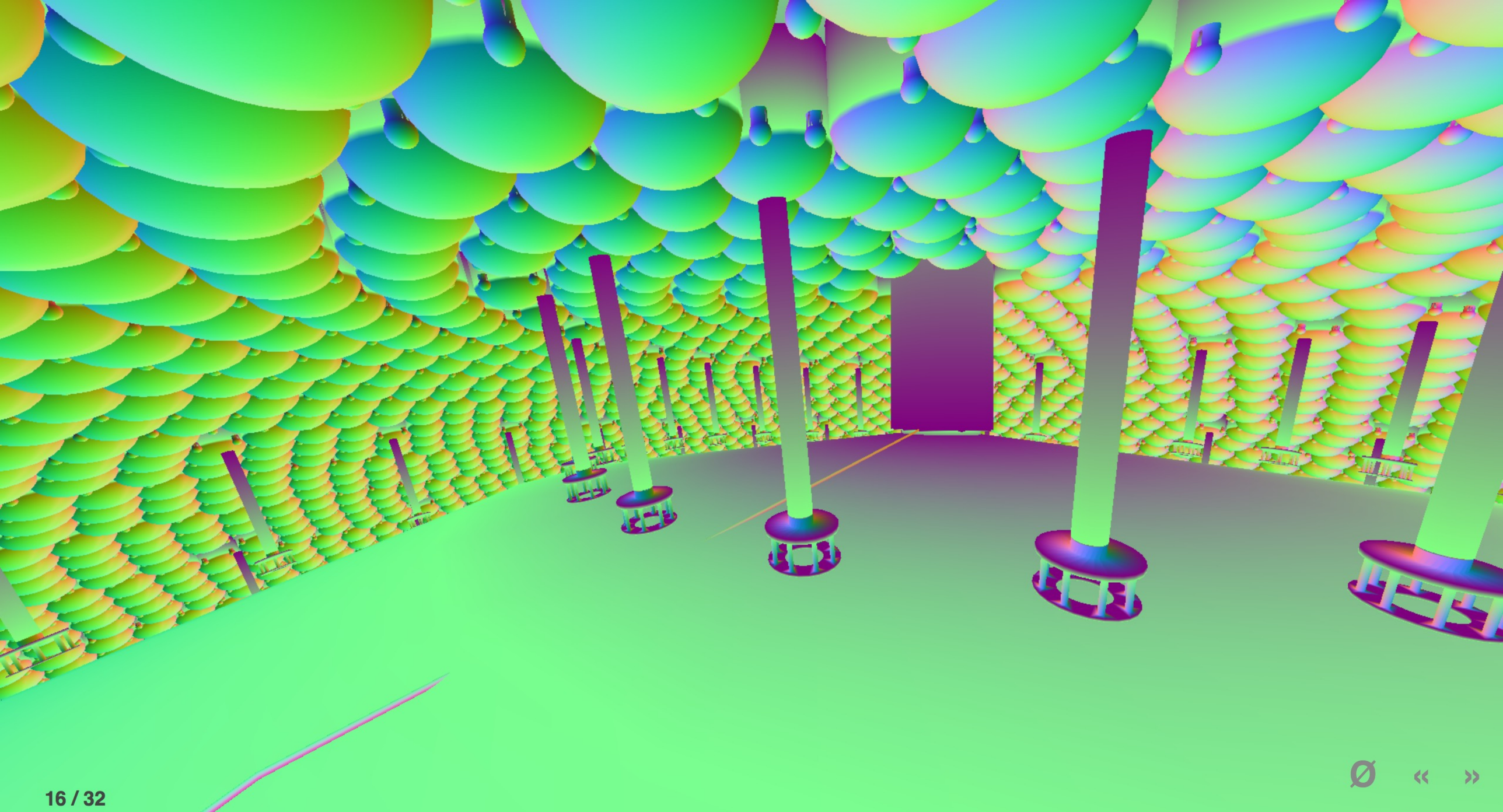
- outer material (parent)
- outer surface (inward photons, parent -> self)
- inner surface (outward photons, self -> parent)
- inner material (self)

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*:

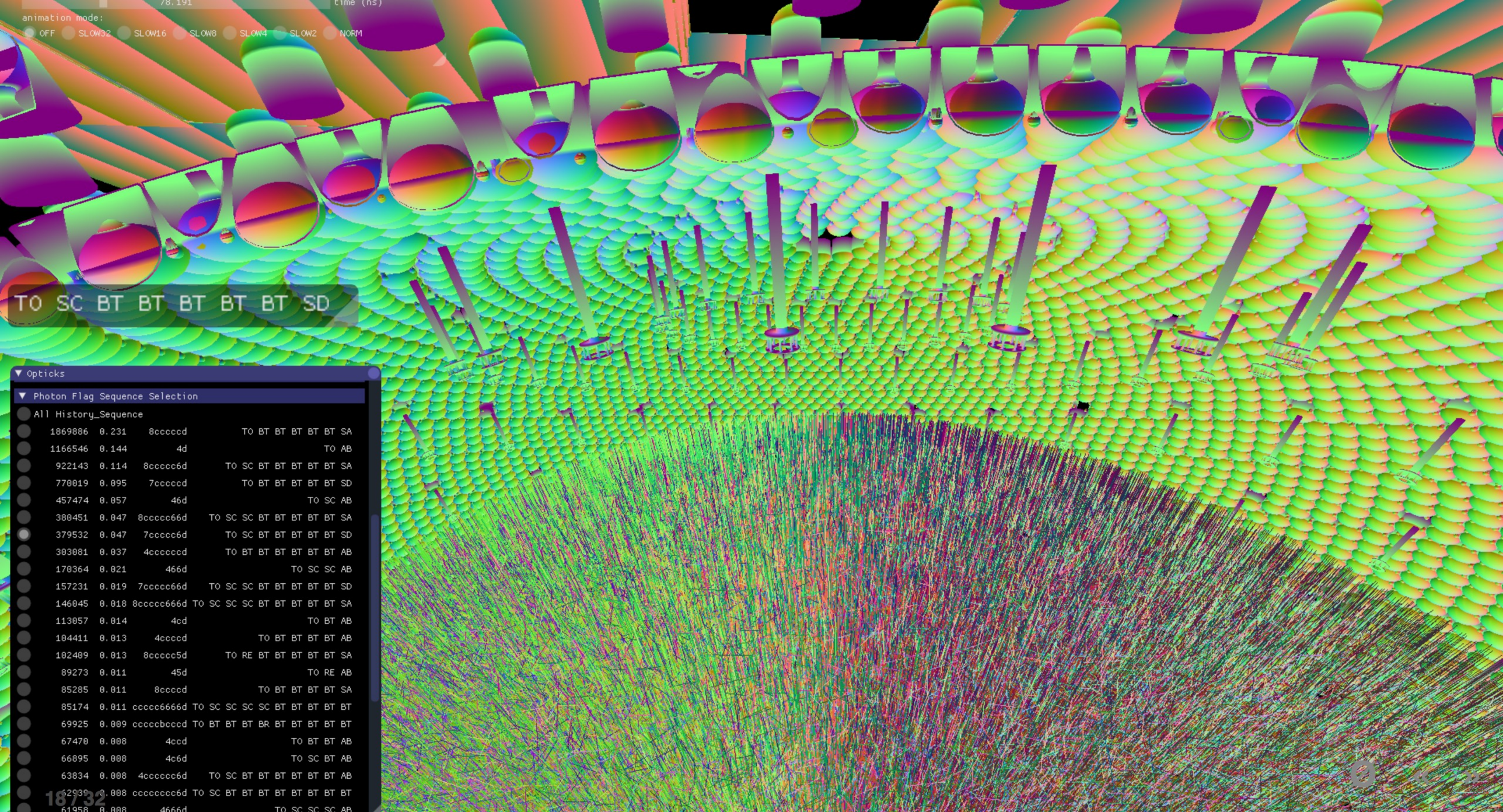
- CPU generated photons
- GPU generated randoms, fed to *Geant4*

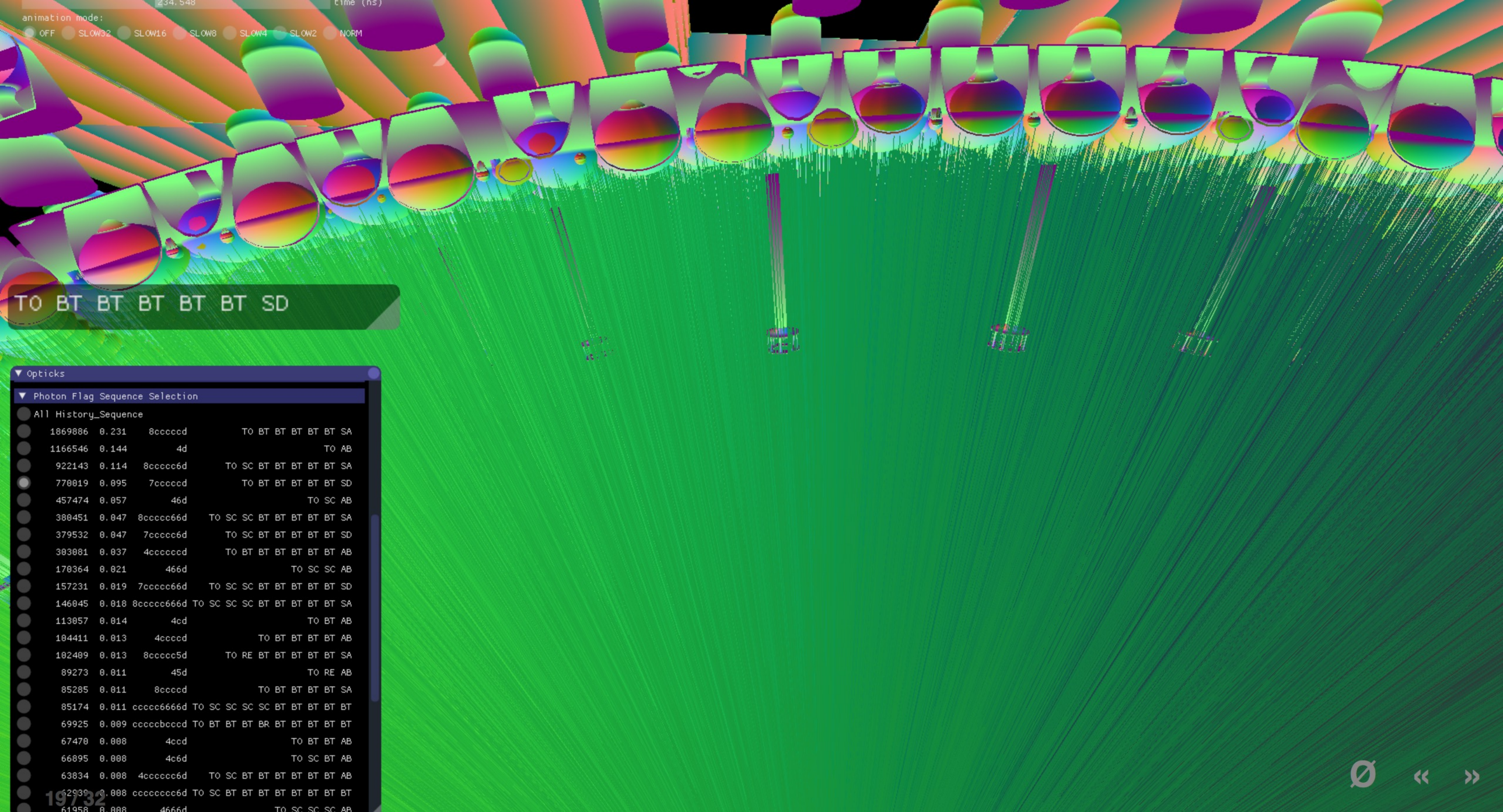
Common recording into *OpticksEvents*:

- 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, wavlen


78.191
time (ns)
animation mode:
OFF
SLOW32
SLOW16
SLOW8
SLOW4
SLOW2
NORM
TO SC BT BT BT BT BT SD
Opticks
Photon Flag Sequence Selection
All History_Sequence
1869886 0.231 8ccccd TO BT BT BT BT BT SA
1166546 0.144 4d TO AB
922143 0.114 8ccccc6d TO SC BT BT BT BT BT SA
770019 0.095 7ccccd TO BT BT BT BT BT SD
457474 0.057 46d TO SC AB
380451 0.047 8ccccc66d TO SC SC BT BT BT BT BT SA
379532 0.047 7ccccc6d TO SC BT BT BT BT BT SD
303081 0.037 4cccccd TO BT BT BT BT BT BT AB
170364 0.021 466d TO SC SC AB
157231 0.019 7ccccc66d TO SC SC BT BT BT BT BT SD
146045 0.018 8ccccc666d TO SC SC SC BT BT BT BT BT SA
113057 0.014 4cd TO BT AB
104411 0.013 4ccccd TO BT BT BT BT AB
102409 0.013 8ccccc5d TO RE BT BT BT BT BT SA
89273 0.011 45d TO RE AB
85285 0.011 8ccccd TO BT BT BT BT BT SA
85174 0.011 ccccc6666d TO SC SC SC SC BT BT BT BT BT
69925 0.009 cccccbcccd TO BT BT BT BR BT BT BT BT BT
67470 0.008 4ccd TO BT BT AB
66895 0.008 4c6d TO SC BT AB
63834 0.008 4ccccccd TO SC BT BT BT BT BT BT AB
62939 0.008 cccccccc6d TO SC BT BT BT BT BT BT BT BT
61958 0.008 4666d TO SC SC SC AB
187/32


234.548
time (ns)
animation mode:
OFF
SLOW32
SLOW16
SLOW8
SLOW4
SLOW2
NORM
TO BT BT BT BT BT SD
Opticks
Photon Flag Sequence Selection
All History_Sequence
1869886 0.231 8ccccd TO BT BT BT BT BT SA
1166546 0.144 4d TO AB
922143 0.114 8ccccc6d TO SC BT BT BT BT BT SA
770019 0.095 7ccccd TO BT BT BT BT BT SD
457474 0.057 46d TO SC AB
380451 0.047 8ccccc66d TO SC SC BT BT BT BT BT SA
379532 0.047 7ccccc6d TO SC BT BT BT BT BT SD
303081 0.037 4cccccd TO BT BT BT BT BT BT AB
170364 0.021 466d TO SC SC AB
157231 0.019 7ccccc66d TO SC SC BT BT BT BT BT SD
146045 0.018 8ccccc666d TO SC SC SC BT BT BT BT BT SA
113057 0.014 4cd TO BT AB
104411 0.013 4ccccd TO BT BT BT BT AB
102409 0.013 8ccccc5d TO RE BT BT BT BT BT SA
89273 0.011 45d TO RE AB
85285 0.011 8ccccd TO BT BT BT BT SA
85174 0.011 ccccc6666d TO SC SC SC SC BT BT BT BT BT
69925 0.009 cccccbcccd TO BT BT BT BR BT BT BT BT BT
67470 0.008 4ccd TO BT BT AB
66895 0.008 4c6d TO SC BT AB
63834 0.008 4ccccccd TO SC BT BT BT BT BT BT AB
61958 0.008 4666d TO SC SC SC AB
19732

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

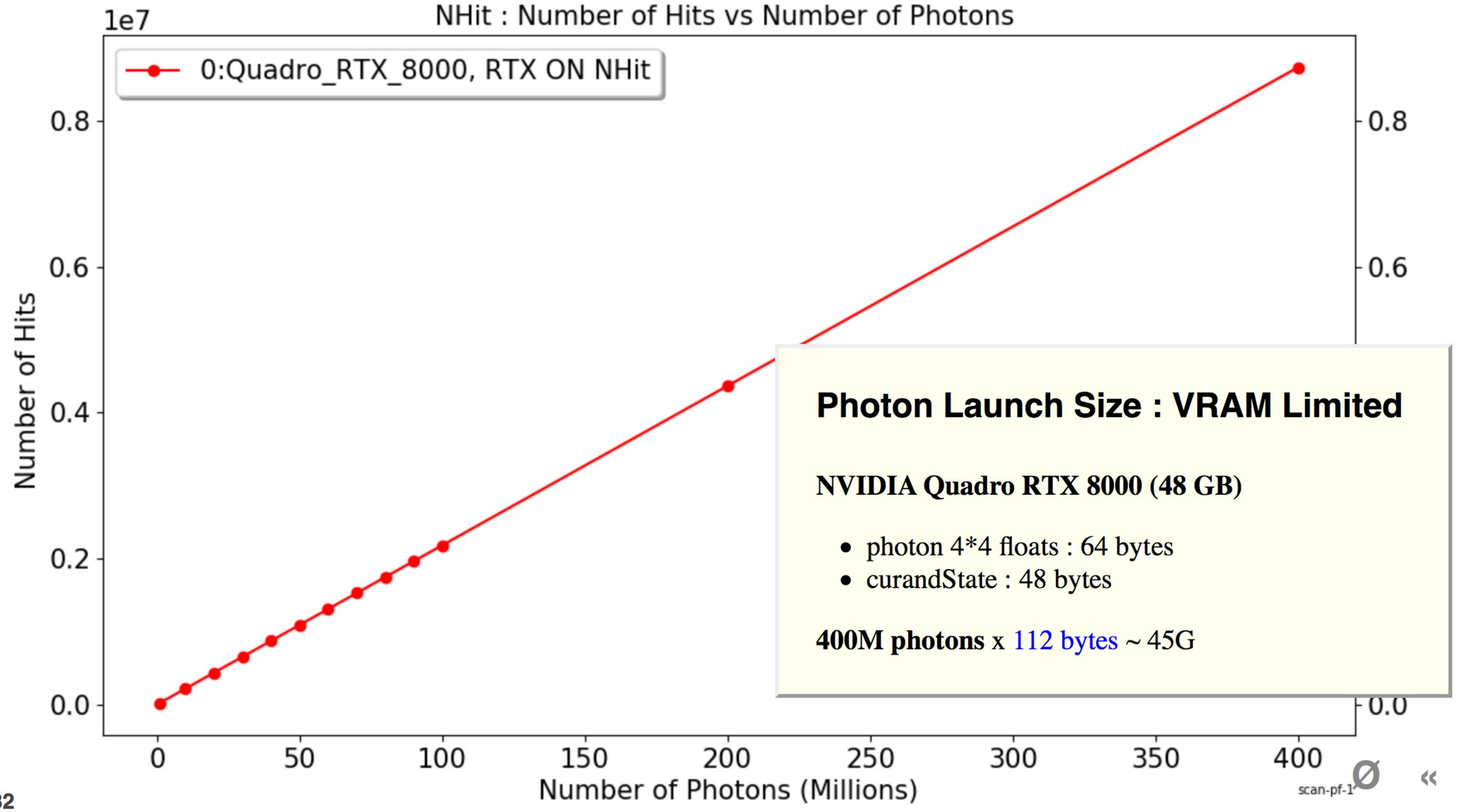
- Opticks 0.0.0 Alpha
- Geant4 10.4p2
- NVIDIA OptiX 6.5.0
- NVIDIA Driver 435.21
- CUDA 10.1

IHEP GPU Cluster

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

NVIDIA Quadro RTX 8000 (48G)

谢谢 NVIDIA China
for loaning the card

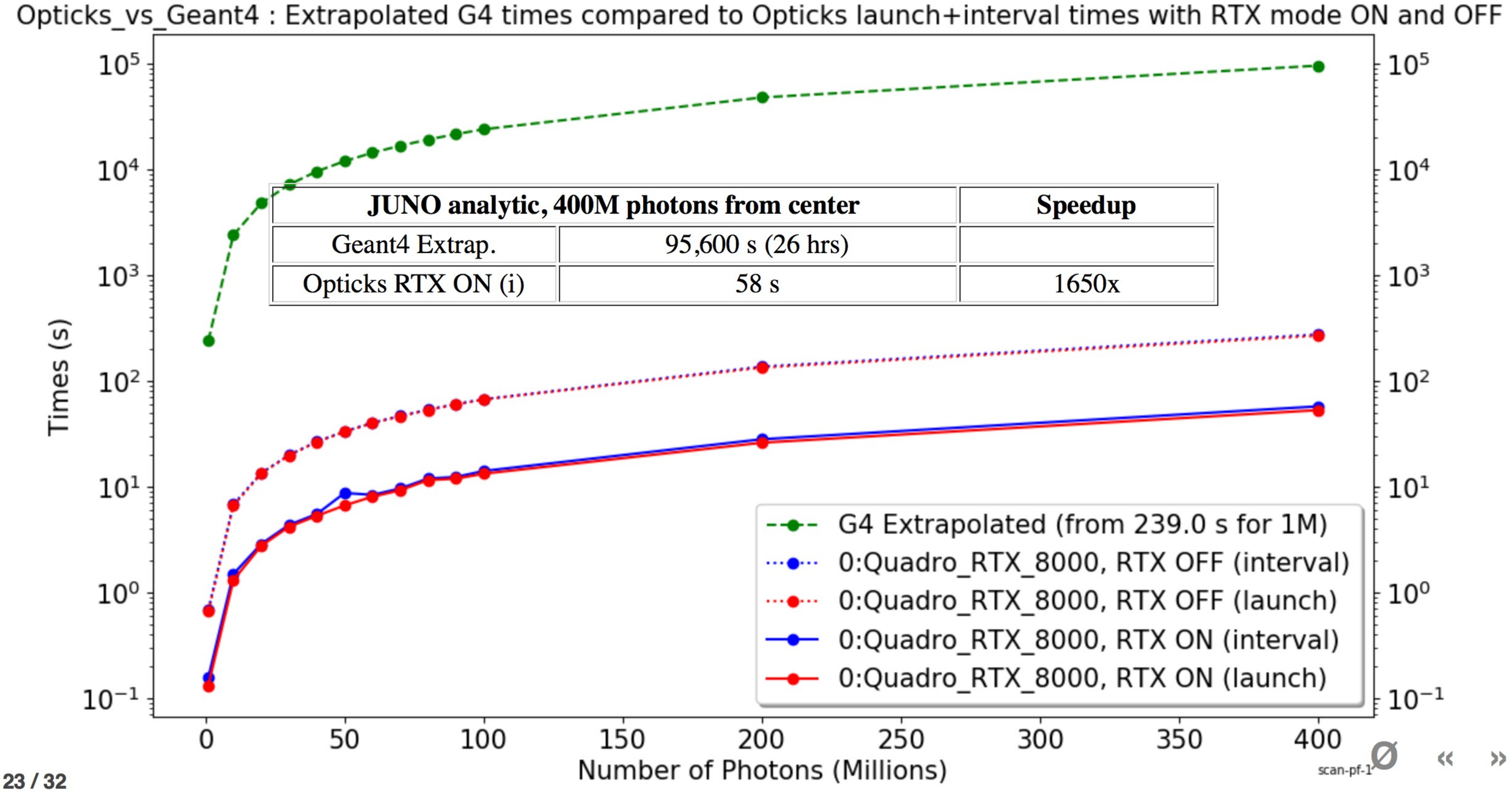


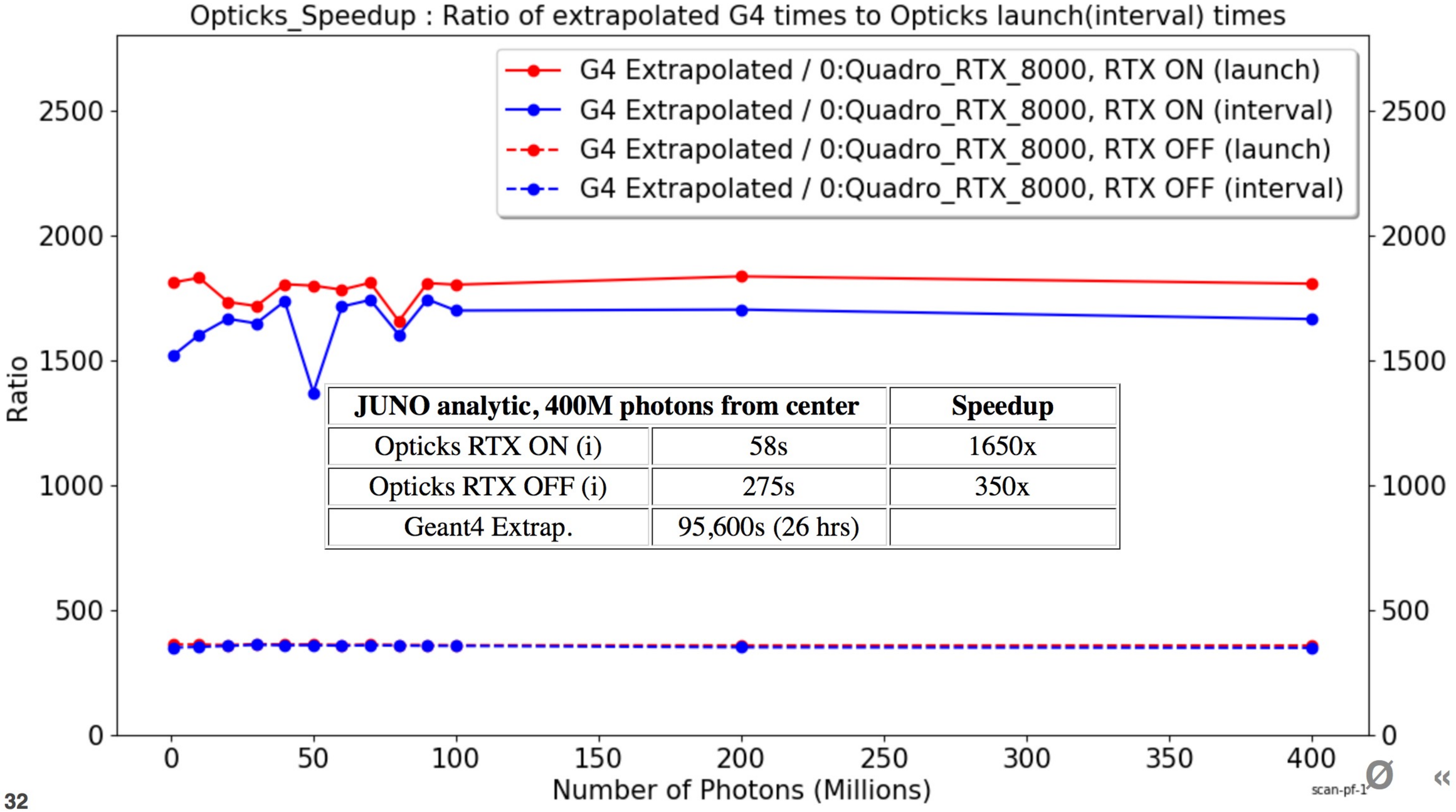
Photon Launch Size : VRAM Limited

NVIDIA Quadro RTX 8000 (48 GB)

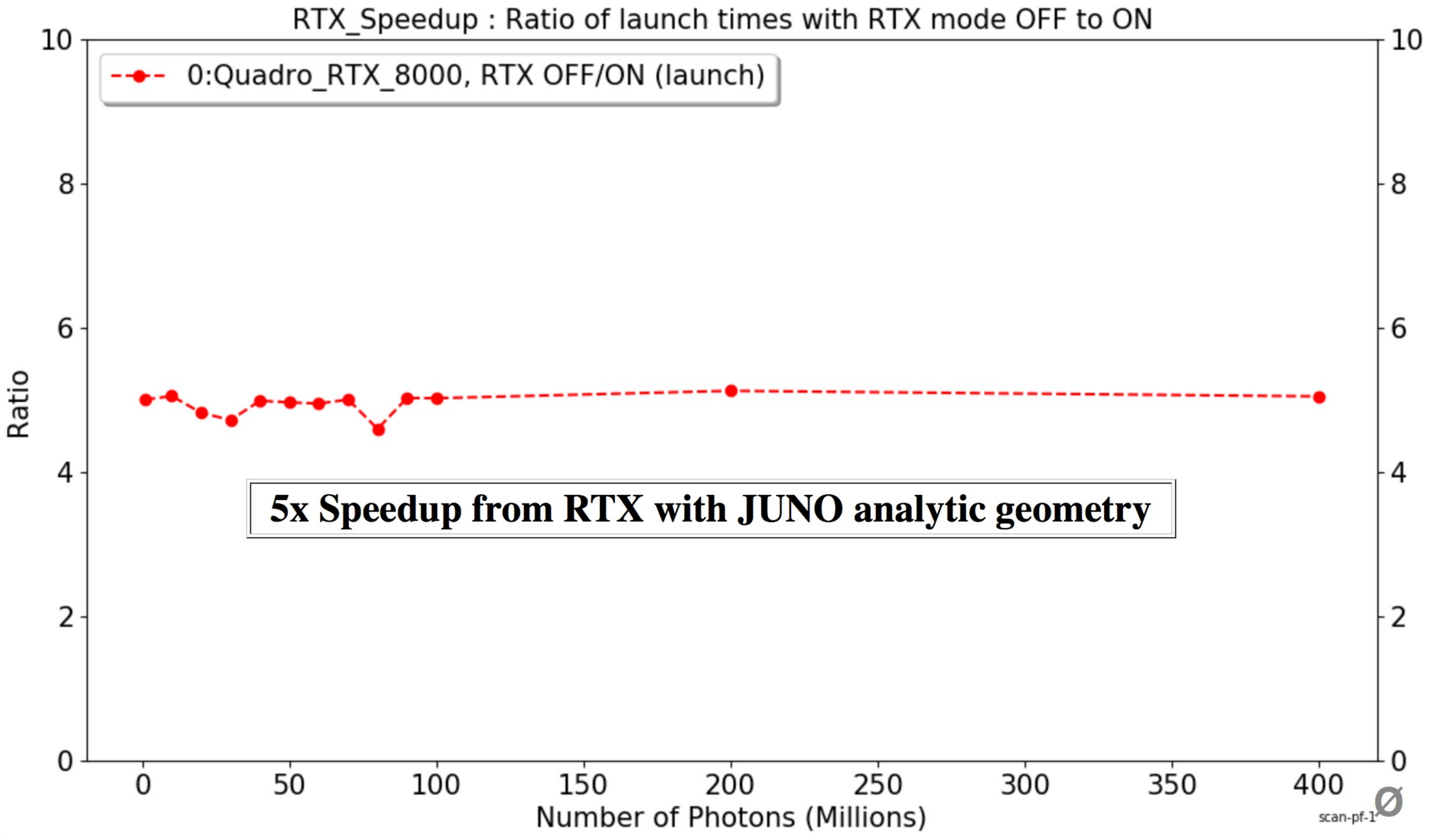
- photon 4*4 floats : 64 bytes
- curandState : 48 bytes

400M photons x 112 bytes ~ 45G


Opticks_vs_Geant4 : Extrapolated G4 times compared to Opticks launch+interval times with RTX mode ON and OFF
JUNO analytic, 400M photons from center
Speedup
Geant4 Extrap.
95,600 s (26 hrs)
Opticks RTX ON (i)
58 s
1650x
Times (s)
Number of Photons (Millions)
G4 Extrapolated (from 239.0 s for 1M)
0:Quadro_RTX_8000, RTX OFF (interval)
0:Quadro_RTX_8000, RTX OFF (launch)
0:Quadro_RTX_8000, RTX ON (interval)
0:Quadro_RTX_8000, RTX ON (launch)

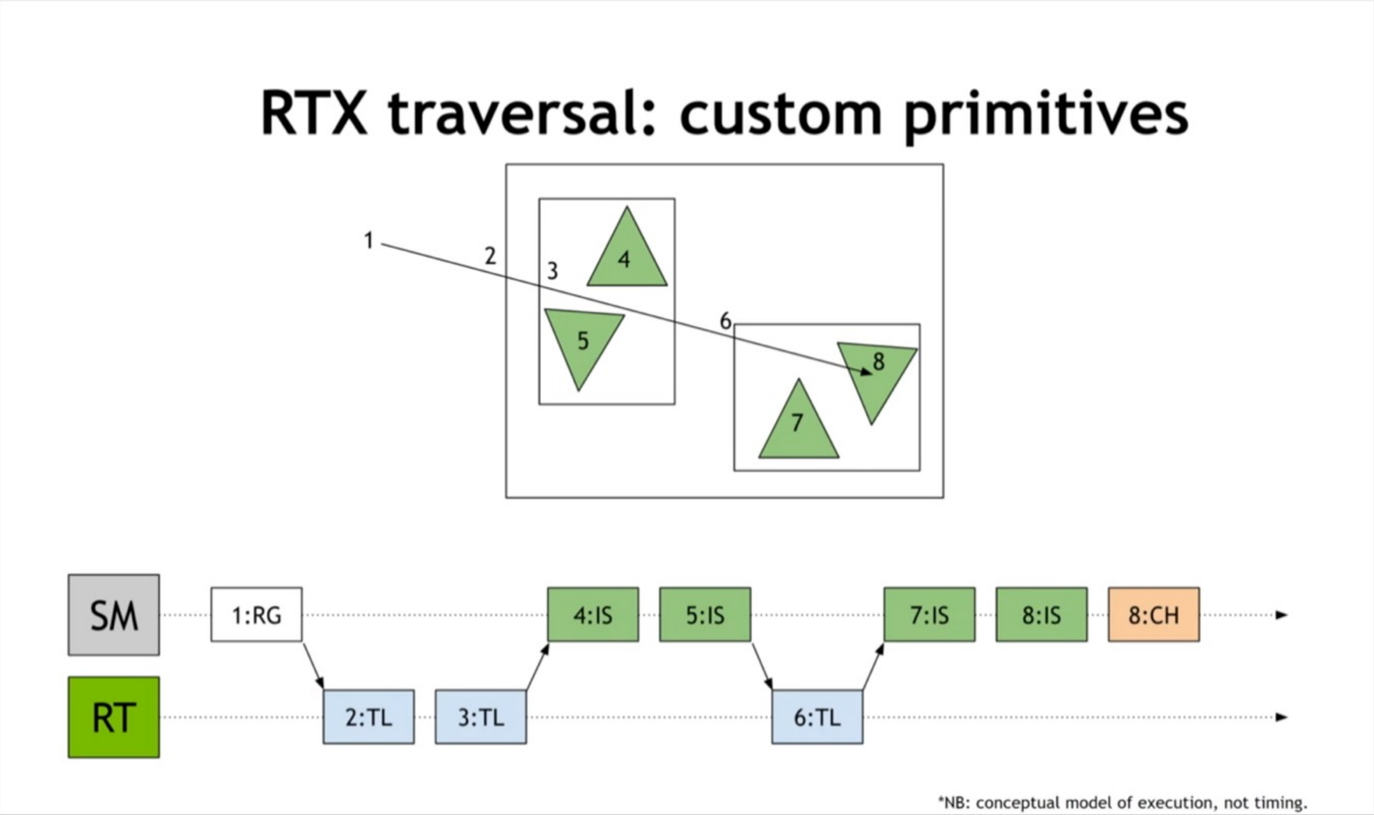


JUNO analytic, 400M photons from center		Speedup
Opticks RTX ON (i)	58s	1650x
Opticks RTX OFF (i)	275s	350x
Geant4 Extrap.	95,600s (26 hrs)	


RTX_Speedup : Ratio of launch times with RTX mode OFF to ON
0:Quadro_RTX_8000, RTX OFF/ON (launch)
Ratio
Number of Photons (Millions)
0
2
4
6
8
10
0
50
100
150
200
250
300
350
400
5x Speedup from RTX with JUNO analytic geometry
scan-pf-1

Useful Speedup > 1000x : But Why Not Giga Rays/s ? (1 Photon ~10 Rays)

- NVIDIA claim : 10 Giga Rays/s with RT Core
- **-> 1 Billion photons per second**
- **RT cores : built-in triangle intersect + 1-level of instancing**
- flatten scene model to avoid SM<->RT roundtrips ?



OptiX Performance Tools and Tricks, David Hart, NVIDIA
https://developer.nvidia.com/siggraph/2019/video/sig915-vid ❏

100M photon RTX times, avg of 10

Launch times for various geometries			
Geometry	**Launch (s)**	**Giga Rays/s**	**Relative to ana**
JUNO ana	13.2	0.07	
JUNO tri.sw	6.9	0.14	1.9x
JUNO tri.hw	2.2	0.45	6.0x
Boxtest ana	0.59	1.7	
Boxtest tri.sw	0.62	1.6	
Boxtest tri.hw	0.30	3.3	1.9x

- ana : Opticks analytic CSG (SM)
- tri.sw : software triangle intersect (SM)
- tri.hw : hardware triangle intersect (RT)

JUNO 15k triangles, 132M without instancing

Simple Boxtest geometry gets into ballpark ∅

Where Next for Opticks ?

JUNO+Opticks into Production

- optimize geometry modelling for RTX
- full JUNO geometry validation iteration
- JUNO offline integration
- optimize GPU cluster throughput:
  - split/join events to fit VRAM
  - job/node/multi-GPU strategy
- support OptiX 7, find multi-GPU load balancing approach

Geant4+Opticks Integration : Work with Geant4 Collaboration

- finalize *Geant4+Opticks* extended example
  - aiming for *Geant4* distrib
- prototype *Genstep* interface inside *Geant4*
  - avoid customizing *G4Cerenkov G4Scintillation*

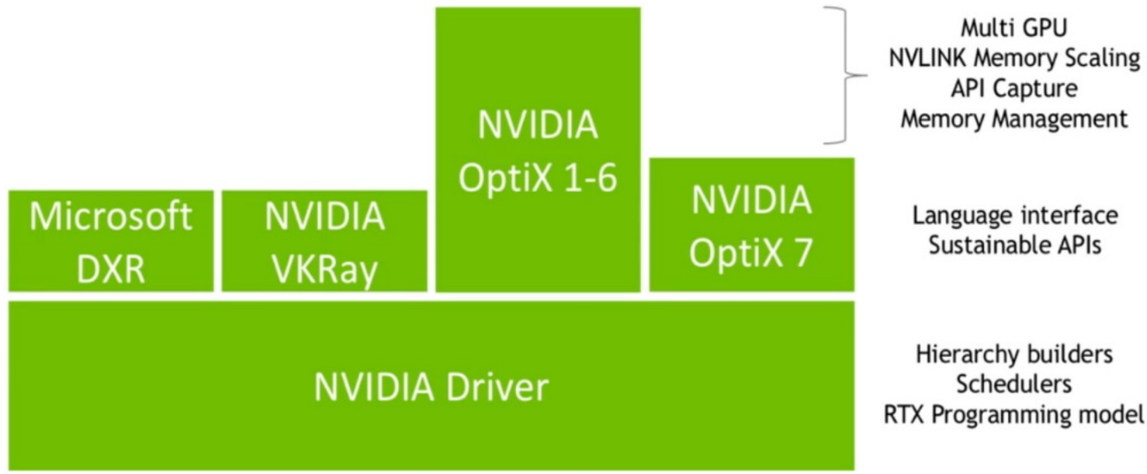
Alpha Development ------>----------> Robust Tool

- many more users+developers required (current ~10+1)
- if you have an optical photon simulation problem ...
  - start by joining : https://groups.io/g/opticks ❏

NVIDIA OptiX 7 : Entirely new API

- introduced August 2019
- low-level CUDA-centric thin API
- ~~near perfect scaling to 4 GPUs, for free~~

Introducing OptiX 7



Drastically Improved Optical Photon Simulation Performance...

Three revolutions reinforcing each other:

- games -> graphics revolution -> GPU -> cheap TFLOPS
- internet scale big datasets -> ML revolution
- computer vision revolution for autonomous vehicles

Deep rivers of development, ripe for re-purposing

- analogous problems -> solutions
- experience across fields essential to find+act on analogies

Example : DL denoising for faster ray trace convergence

- analogous to hit aggregation
- skip the hits, jump straight to DL smoothed probabilities
  - blurs the line between simulation and reconstruction

Re-evaluate long held practices in light of new realities:

- large ROOT format (C++ object) MC samples repeatedly converted+uploaded to GPU for DL training ... OR:
- small Genstep NumPy arrays uploaded, dynamically simulated into GPU hit arrays in fractions of a second

How is >1000x possible ?

Progress over 30 yrs, Billions of Dollars

- industry funded : game, film, design, ...
- re-purposed by translating geometry to GPU
  - tree of C++ objects -> arrays -> BVH

Photon Simulation ideally suited to GPU

- millions of photons -> abundantly parallel
- simple phys. -> small stack -> many in flight
- decoupled -> no synchronization

Dynamically generated simulation feasible ?

- current reconstruction -> custom simulation
- no more : limited MC stats in edge cases

NEST + Opticks Approaches to Integration ?

How difficult to port NEST photon generation to GPU ?

A quick look at NEST code:

```
NESTProc::AtRestDoIt
NESTProc::MakePhoton
NESTcalc::PhotonTime
```

- suggests doable
  - as pure calulation and random throws
  - no need even for GPU texture lookups
  - just need to collect "stack" into Gensteps
- only generation loop needed on GPU
- "Genstep" parameter setup stays on CPU

Example : JUNO Scintillator Reemission

- subset of absorption photons reborn in same CUDA thread
- bake inverse CDF into a GPU texture
- *cuRAND* random lookup into texture yields wavelength
- wavelength distribution matches CPU implementation
  - just a fast texture lookup on GPU

NESTProc::PostStepDoIt Options

GPU Photon Generation

- offload optical photon memory costs to VRAM
- no photon copying overheads
- requires porting generation loop to CUDA
  - collect "genstep" inputs

-> splits implementation CPU/GPU

CPU Input Photons

- simple : nothing to port
  - just collect photons, and not *AddSecondary*
- pay twice for photon memory + copy overhead
- (used by Opticks for convenient CPU/GPU comparisons)
- easily added to *G4Opticks* interface

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.

Highlights 2019

- Benefit from hardware accelerated ray tracing
- **Opticks > 1000x Geant4** (one Turing GPU)

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

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

Opticks : Needs and Challenges

- perfect *Geant4* match not feasible (*float* vs *double*)

- NVIDIA OptiX : closed source proprietry
  - "black box" BVH : details secret
    - flying blind -> expts -> best BVH
- NVIDIA OptiX engineers helpful, BUT:
  - "performance minimal reproducers" tough to create
- No influence on direction of NVIDIA OptiX
  - film/game/design studios call the shots
  - must accept what : *NVIDIA giveth and taketh away*
    - eg OptiX 7 : ~~near perfect linear scaling up to 4 GPUs~~

- BYOB : bring your own BVH (like Chroma from Stanley Seibert)
  - difficult to justify development time when no access to RT Cores

Opticks Needs YOU

- to use it -> improves it
- currently ~10 users + 1 developer
- many more to become a robust tool

Lots of interest, little contribution