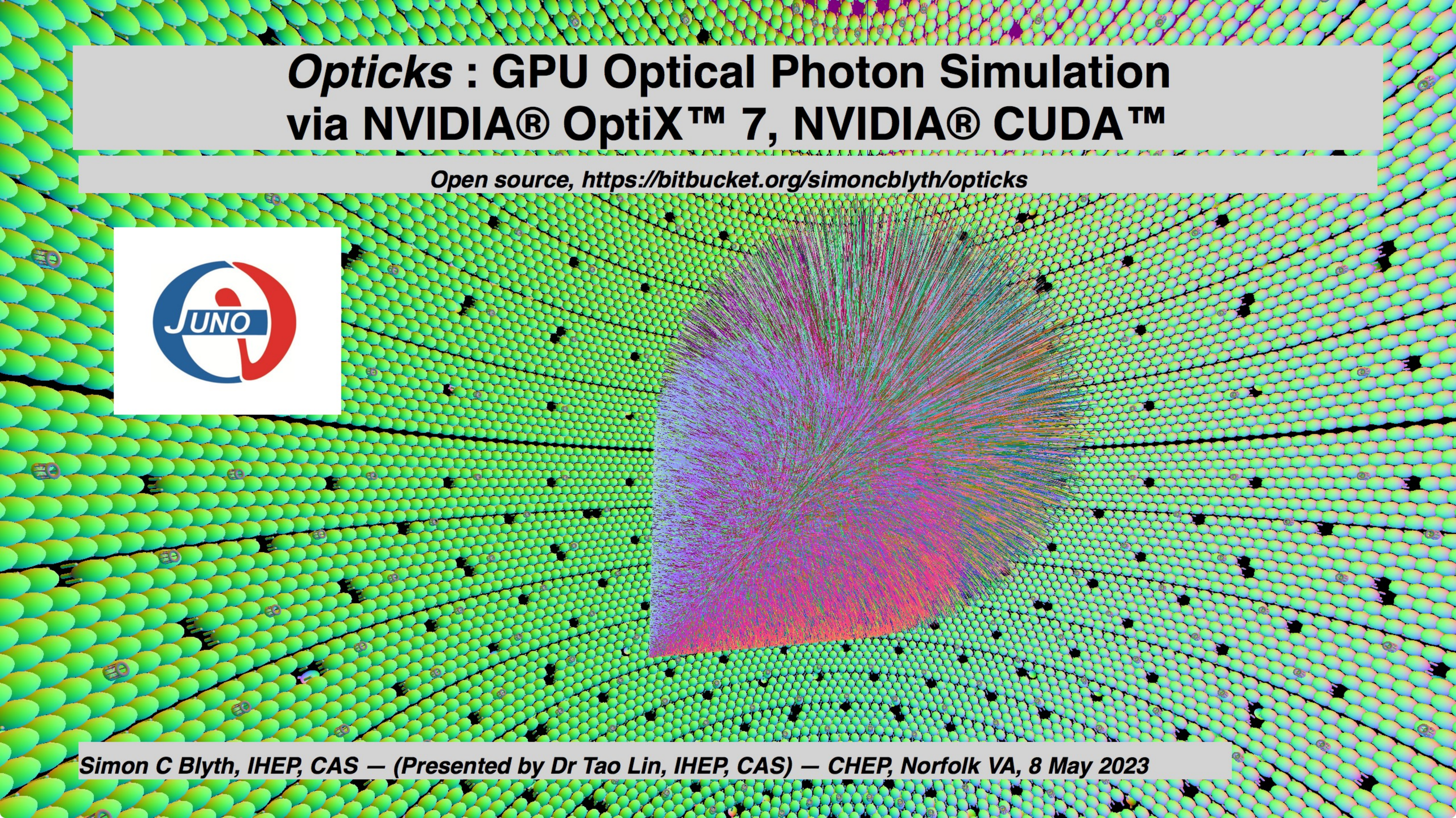
Opticks : GPU Optical Photon Simulation via NVIDIA® OptiX™ 7, NVIDIA® CUDA™

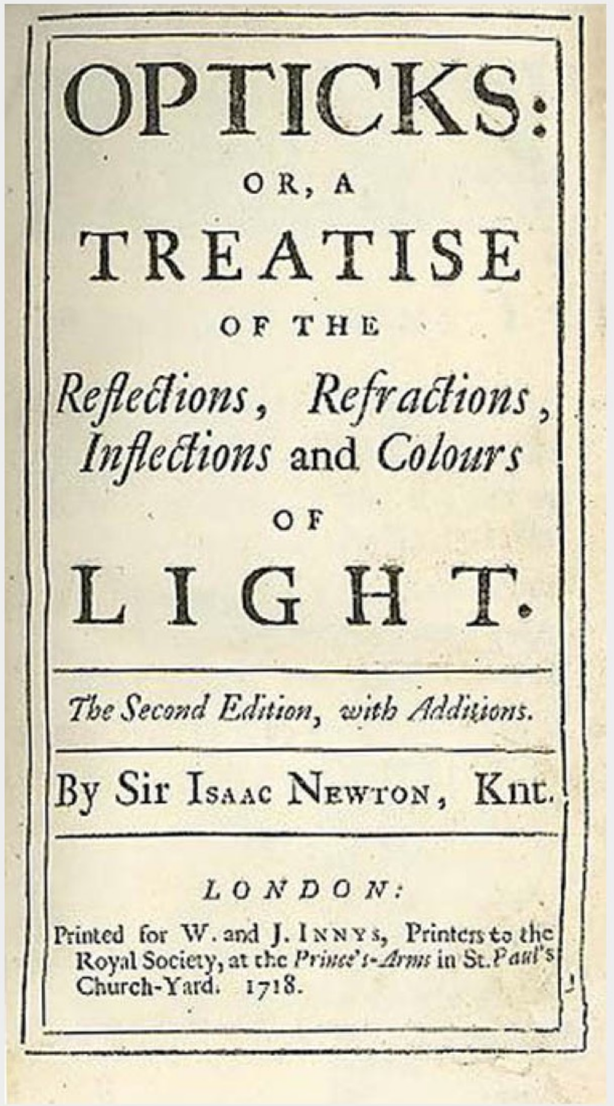
Open source, https://bitbucket.org/simoncblyth/opticks



Simon C Blyth, IHEP, CAS — (Presented by Dr Tao Lin, IHEP, CAS) — CHEP, Norfolk VA, 8 May 2023

Outline

- Optical Photon Simulation : Context and Problem
  - p2: Jiangmen Underground Neutrino Observatory (JUNO)
  - p3: JUNO Optical Photon Simulation Problem...
  - p4: Optical Photon Simulation ≈ Ray Traced Image Rendering
- NVIDIA Tools to create Solution
  - p5,6: NVIDIA Ada Lovelave : 3rd Generation RTX, RT Cores in Data-Center
  - p7: NVIDIA OptiX Ray Tracing Engine
  - p8: NVIDIA OptiX 7 : Entirely new thin API
- Opticks : Introduction + Full Re-implementation
  - p9,10: Geant4 + Opticks Hybrid Workflow : External Optical Photon Simulation
  - p11-12: Full re-implementation for NVIDIA OptiX 7 API
  - p13-14: *CSGFoundry* Geometry Model, Translation to GPU
  - p16: QUDARap : CUDA Optical Simulation Implementation
- Opticks : New Features
  - p17:n-Ary CSG "List-Nodes"
  - p18,19: Specialized n-ary CSG intersect algs : "contiguous" and "dis-contiguous"
  - p20: Multi-Layer Thin Film (A,R,T) Calc using TMM (Custom4 Package)
- p21: Summary + Links



OPTICKS:

OR, A

TREATISE

OF THE

Reflections, *Refractions*, *Inflections* and *Colours*

OF

LIGHT.

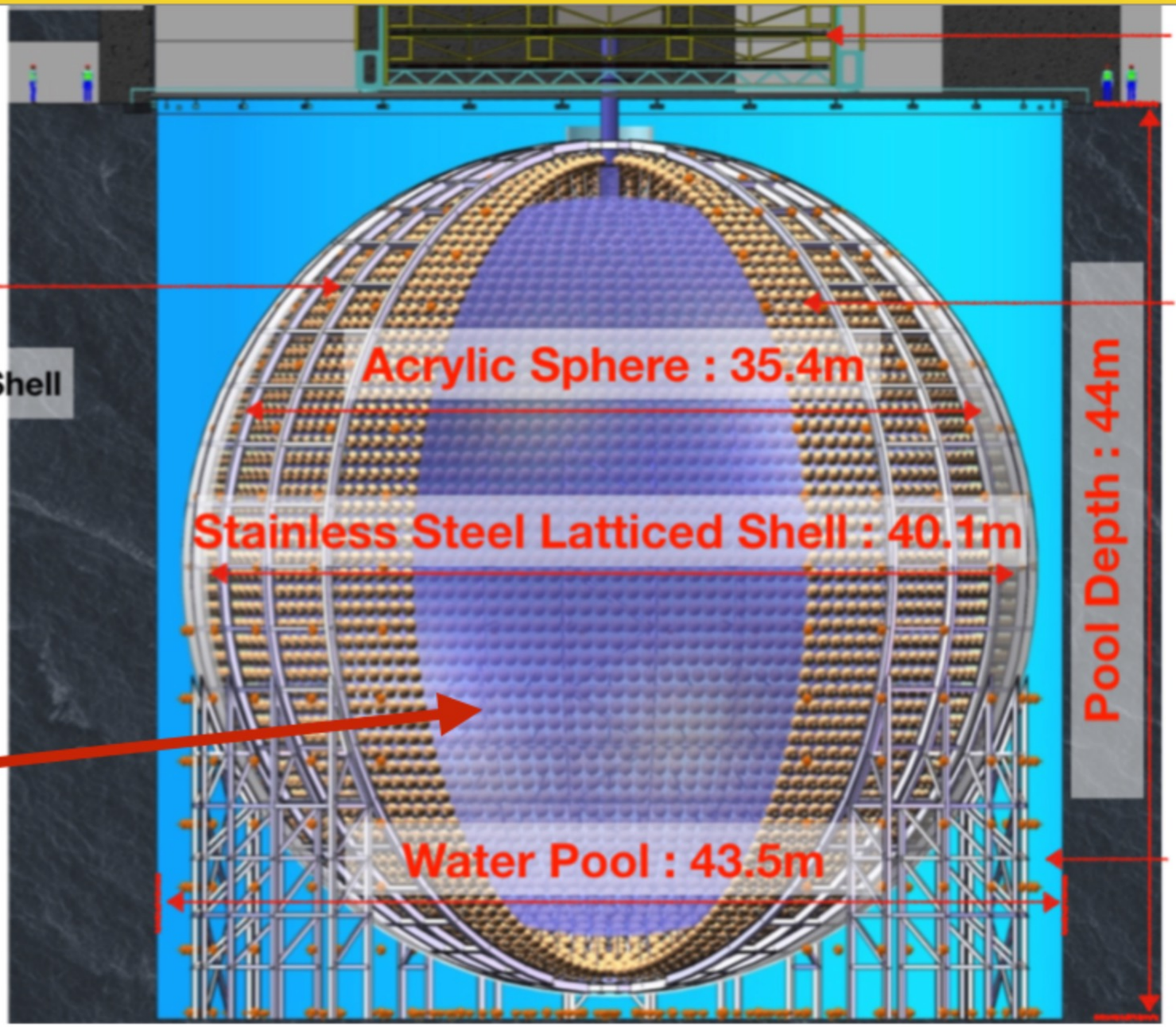
The Second Edition, with Additions.

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.

JUNO : Liquid Scintillator, 18k 20-inch PMTs, 25k 3-inch PMTs



Top Tracker

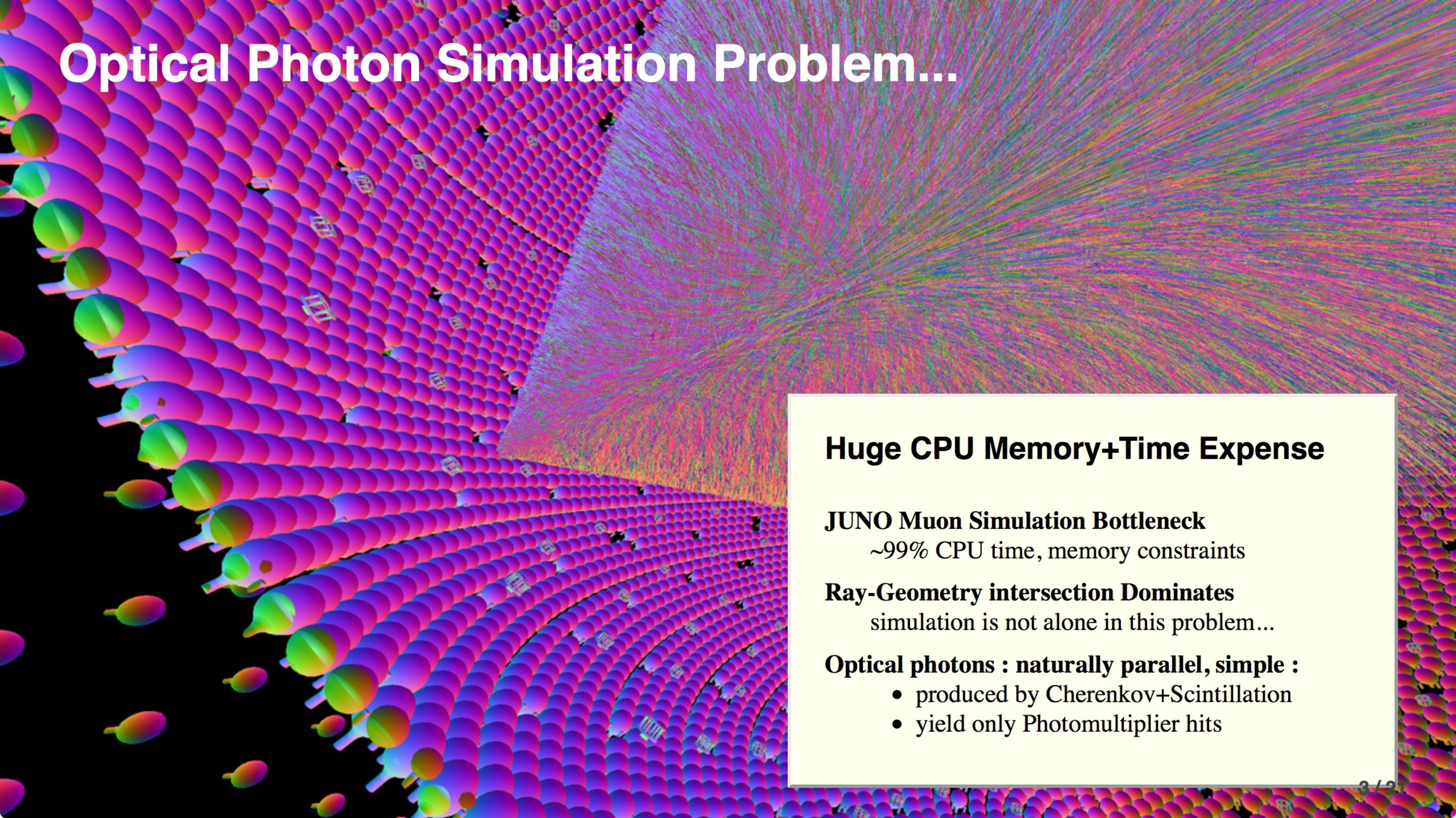
Central Detector
Acrylic Sphere
Stainless Steel Latticed Shell

PMT
18,000 20-inch PMTs
25,000 3-inch PMTs
78% Coverage

Liquid Scintillator
20 kton

Water Cherenkov
35 kton pure water
2,000 20-inch veto PMTs

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

simulation
photon parameters at sensors (PMTs)

rendering
pixel values at image plane

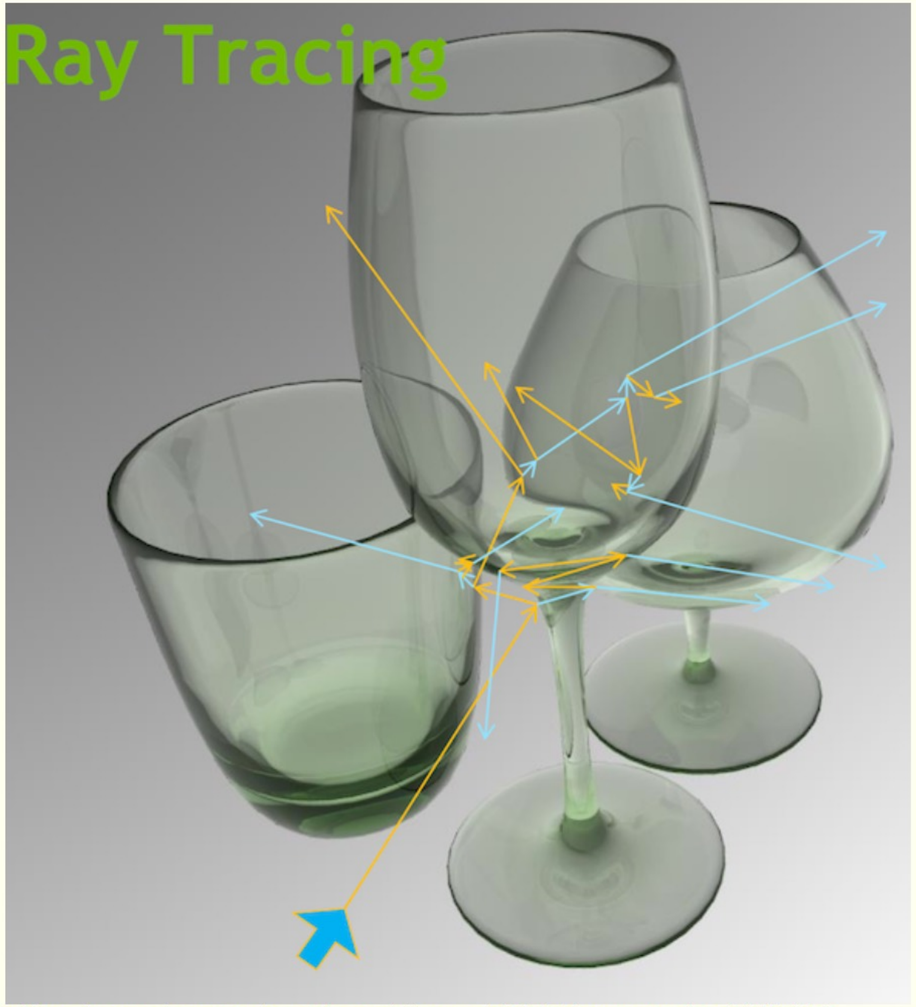
Much in common : geometry, light sources, optical physics

- **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

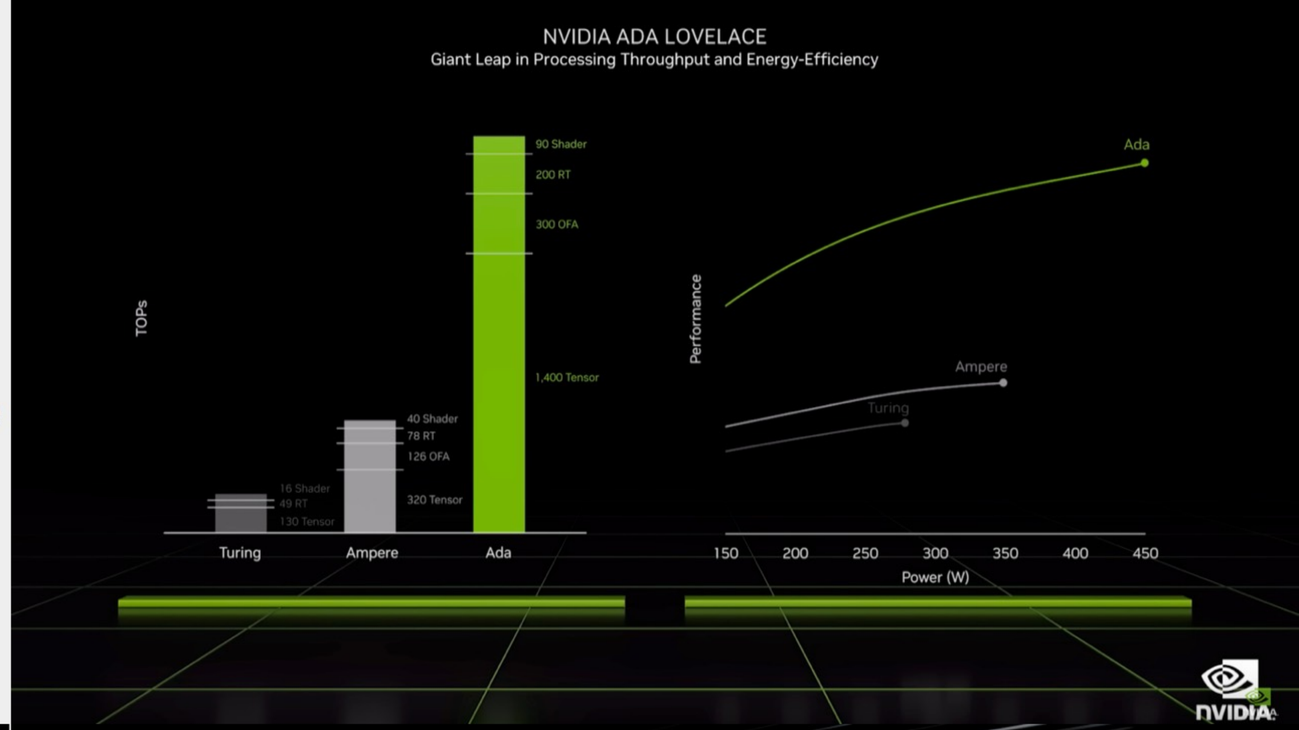
Not a Photo, a Calculation

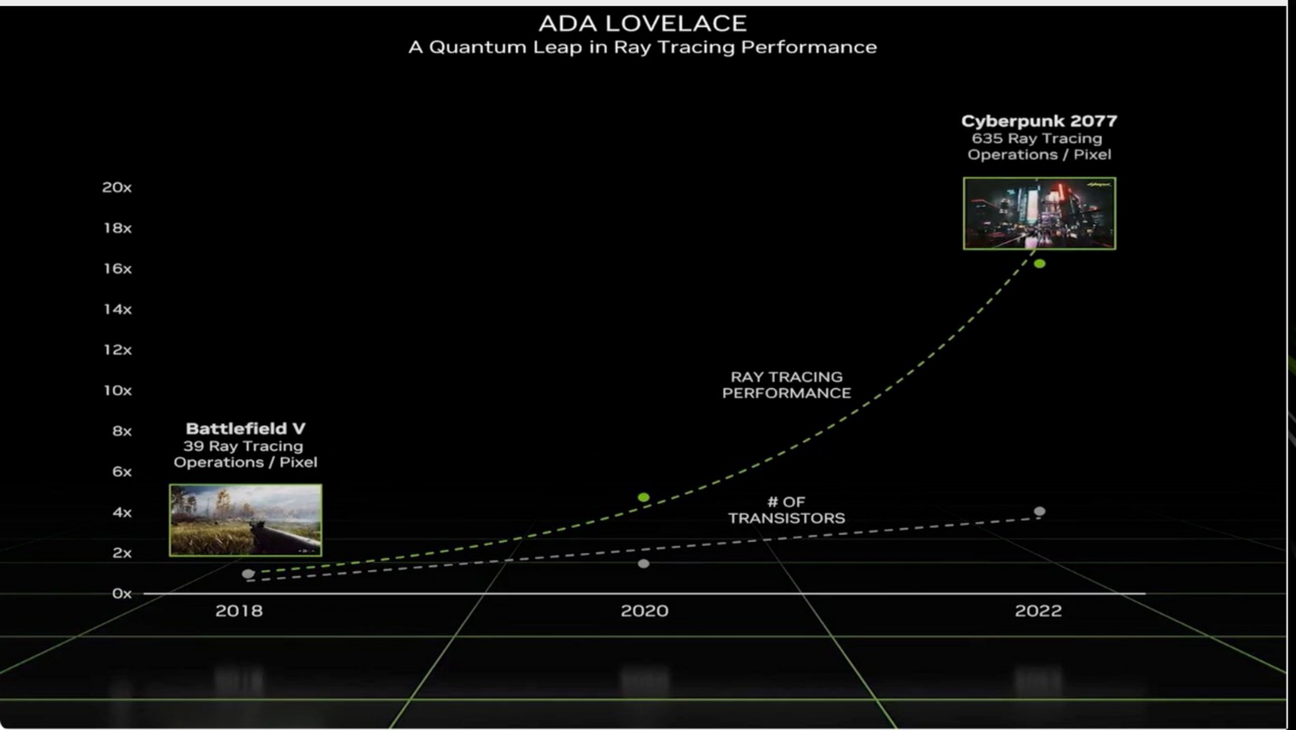


http://on-demand.gputechconf.com/siggraph/2013/presentation/SG3106-Building-Ray-Tracing-Applications-OptiX.pdf

NVIDIA Ada : 3rd Generation RTX

- **RT Core** : ray trace dedicated GPU hardware
- **NVIDIA GeForce RTX 4090 (2022)**
  - 16,384 CUDA Cores, 24GB VRAM, USD 1599
- **Continued large ray tracing improvements:**
  - **Ada** ~2x ray trace over **Ampere** (2020), 4x with DLSS 3
  - **Ampere** ~2x ray trace over **Turing** (2018)
- DLSS : Deep Learning Super Sampling
  - AI upsampling, not applicable to optical simulation



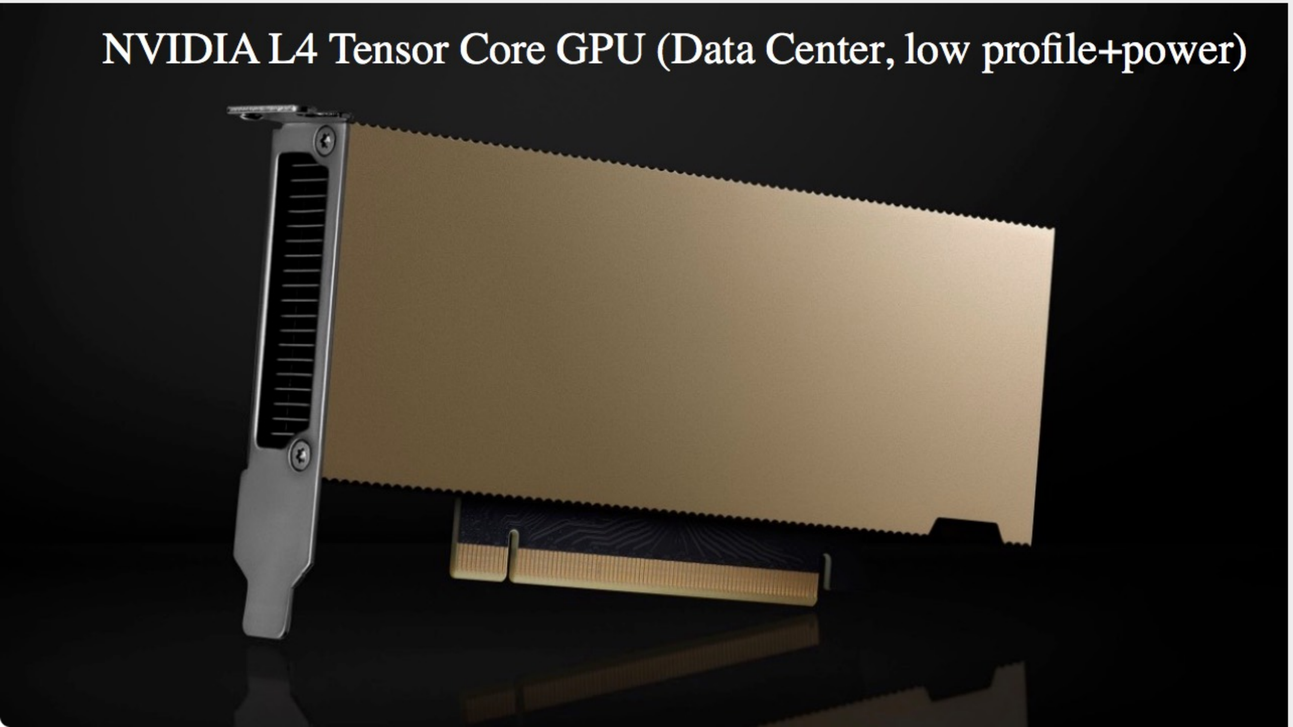


Hardware accelerated Ray tracing (RT Cores) in the Data Center

NVIDIA L4 Tensor Core GPU (Released 2023/03)

- Ada Lovelace GPU architecture
- **universal accelerator for graphics and AI workloads**
- **small form-factor, easy to integrate, power efficient**
- PCIe Gen4 x16 slot without extra power
- Google Cloud adopted for G2 VMs, successor to **NVIDIA T4**
- **NVIDIA L4 likely to become a very popular GPU**

NVIDIA L4 Tensor Core GPU (Data Center, low profile+power)



Data Center GPUs with RT Cores

- **An Established part of NVIDIA Lineup**

	NVIDIA L40	NVIDIA L4
Release	2022/10	2023/03
GPU Arch	Ada Lovelace	Ada Lovelace
VRAM	48 GB gddr6	24 GB gddr6
TDP	300W	**72W**
Form factor	dual slot	**1-slot**
CUDA Core	18,176	7,680
RT Core	**142 (3rd gen)**	**60 (3rd gen)**
Tensor Core	568 (4th gen)	240 (4th gen)
FP32	90.5 TFLOPS	30 TFLOPS
FP16(*)	181 TFLOPS	121 TFLOPS
Predecessor	A40	T4

(*) x2 with Sparsity

NVIDIA® OptiX™ Ray Tracing Engine -- Accessible GPU Ray Tracing

OptiX makes GPU ray tracing accessible

- **Programmable GPU-accelerated Ray-Tracing Pipeline**
- Single-ray shader programming model using CUDA
- ray tracing acceleration using RT Cores (RTX GPUs)
- "...free to use within any application..."

OptiX features

- acceleration structure creation + traversal (eg BVH)
- instanced sharing of geometry + acceleration structures
- compiler optimized for GPU ray tracing

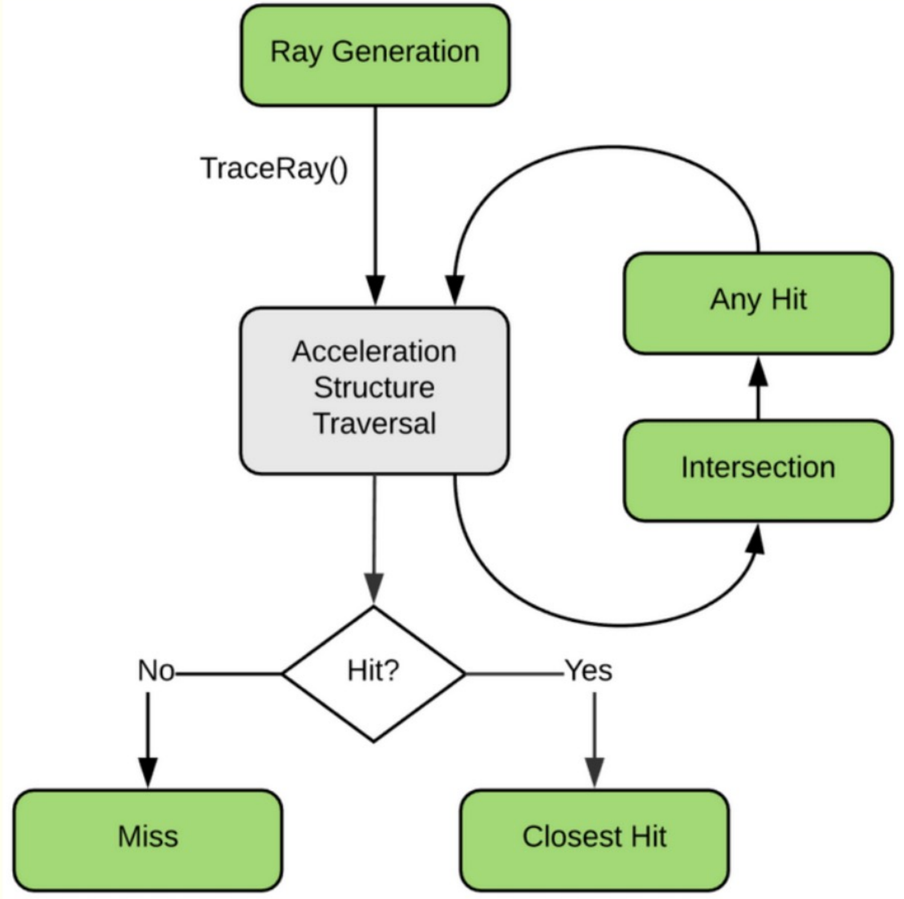
`https://developer.nvidia.com/rtx/ray-tracing/optix`

User provides (Green):

- ray generation
- geometry bounding boxes
- intersect functions
- instance transforms

Flexible Ray Tracing Pipeline

Green: User Programs, **Grey: Fixed function/HW**



Analogous to OpenGL rasterization pipeline

NVIDIA OptiX 7 : Entirely new thin API => Full Opticks Re-implementation

NVIDIA OptiX 6->7 : **drastically slimmed down**

- low-level CUDA-centric thin API (Vulkan-ized)
- headers only (no library, impl in Driver)
- Minimal host state, All host functions are thread-safe
- GPU launches : explicit, asynchronous (CUDA streams)
- ~~near perfect scaling to 4 GPUs, for free~~
- ~~Shared CPU/GPU geometry context~~
- ~~GPU memory management~~
- ~~Multi-GPU support~~

Advantages of 6->7 transition

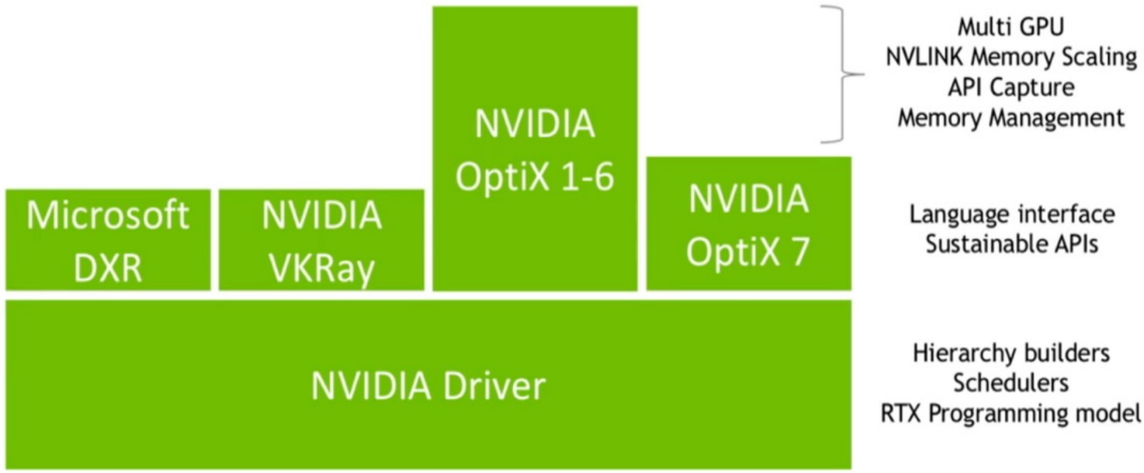
- More control/flexibility over everything
- **Keep pace with state-of-the-art GPU ray tracing**
- Fully benefit from current + future GPUs : RT cores, RTX

BUT: demanded full re-implementation of Opticks

GPU Ray Tracing APIs Converged

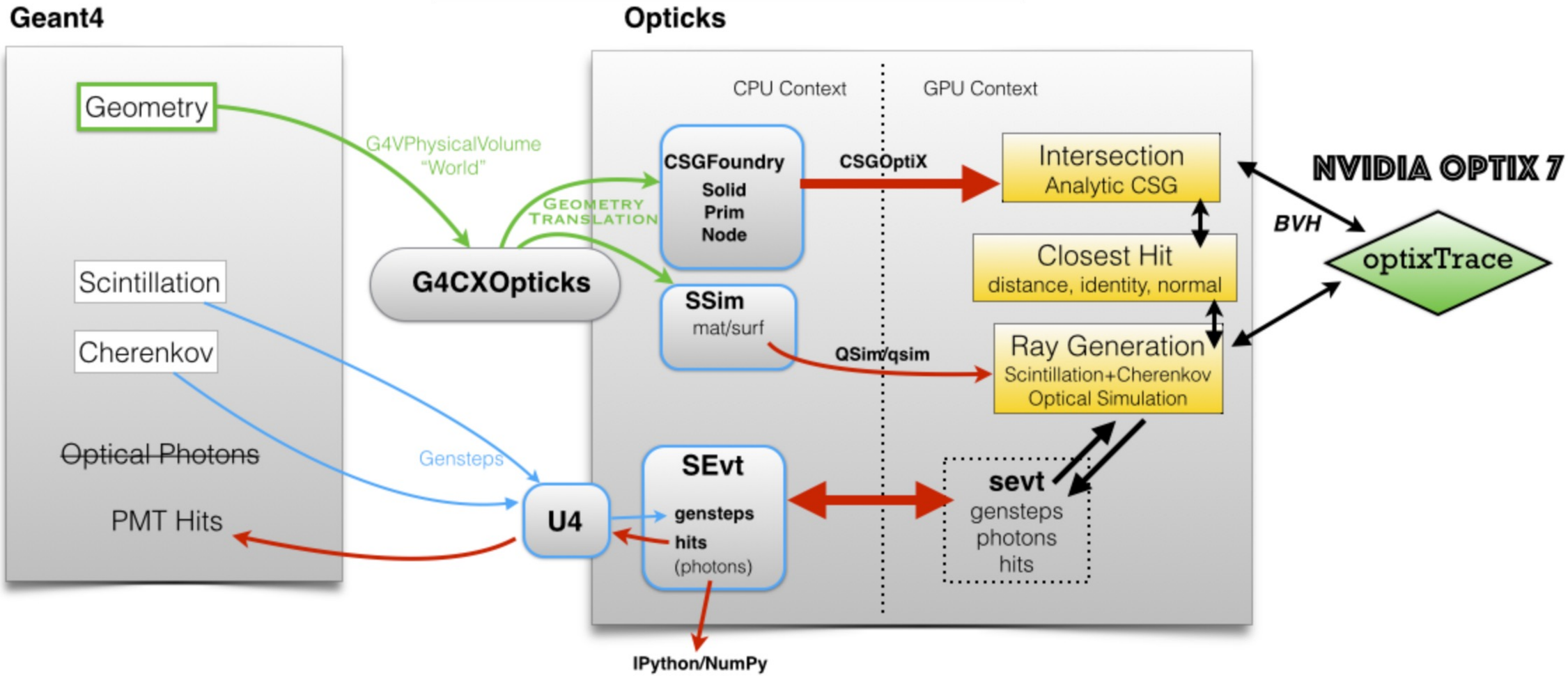
- 3 APIs (DXR,VKRay,OptiX7) over RTX
- Driver updates **independent of application**
- Support new GPUs, performance improvements

Introducing OptiX 7



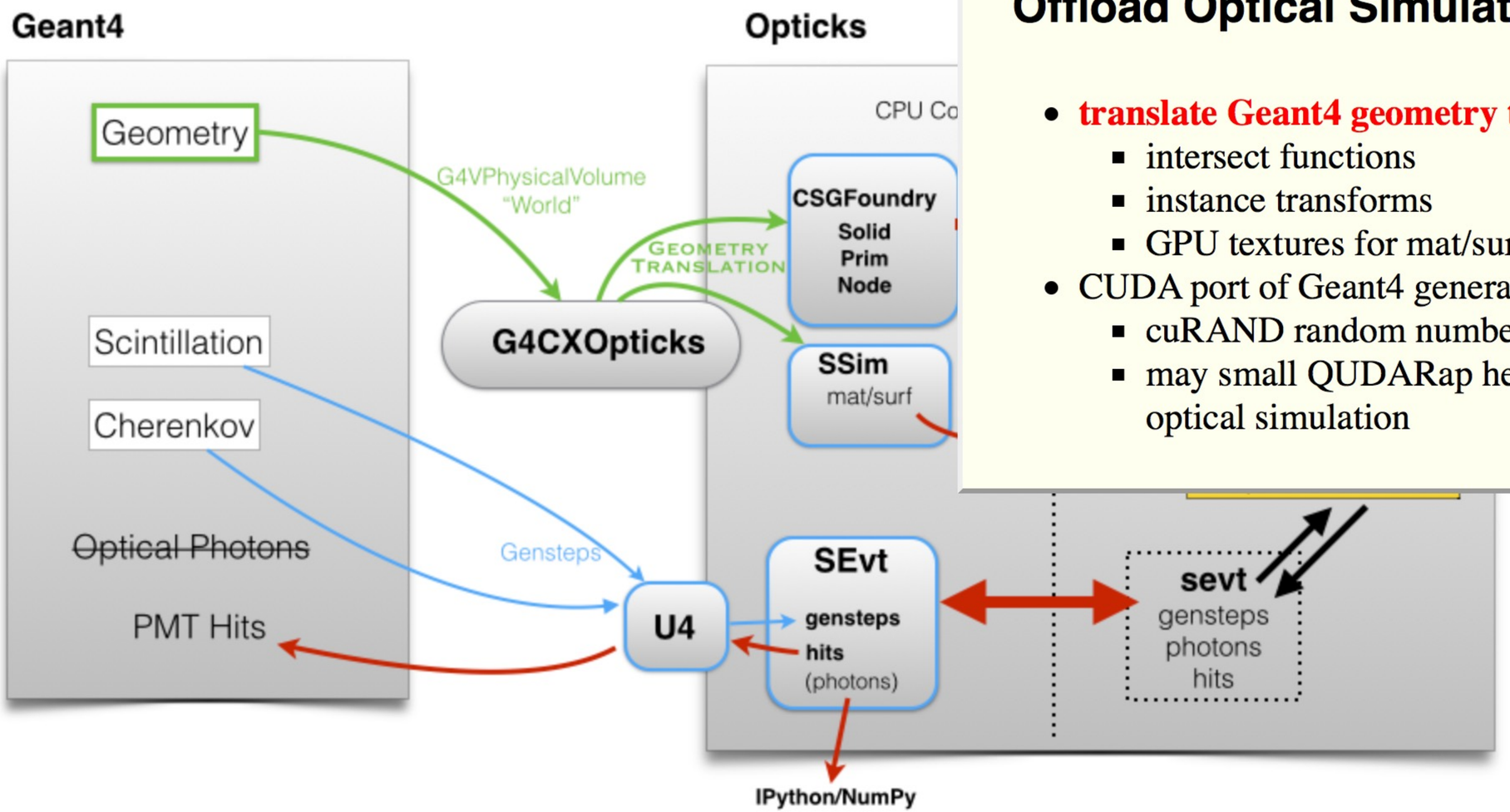
Geant4 + Opticks + NVIDIA OptiX 7 : Hybrid Workflow

https://bitbucket.org/simoncblyth/opticks



Opticks API : split according to dependency -- Optical photons are GPU "resident", only hits need to be copied to CPU memory

Geant4 + Opticks + NVIDIA OptiX 7 : Hybrid Workflow 2



Offload Optical Simulation to GPU

- **translate Geant4 geometry to OptiX GPU**
  - intersect functions
  - instance transforms
  - GPU textures for mat/surf properties
- CUDA port of Geant4 generation+propagation
  - cuRAND random number generation
  - may small QUDARap headers implement optical simulation

Primary Packages and Structs Of Re-Implemented Opticks

SysRap : *many small CPU/GPU headers*

- *stree.h,snode.h* : geometry base types
- *sctx.h sphoton.h* : event base types
- *NP.hh* : serialization into NumPy .npy format files

QUDARap

- *QSim* : optical photon simulation steering
- *QScint,QCerenkov,QProp,...* : modular CUDA implementation

U4

- *U4Tree* : convert geometry into *stree.h*
- *U4* : collect gensteps, return hits

CSG

- *CSGFoundry/CSGSolid/CSGPrim/CSGNode* geometry model
- *csg_intersect_tree.h csg_intersect_node.h csg_intersect_leaf.h* : CPU/GPU intersection functions

CSGOptiX

- *CSGOptiX.h* : manage geometry convert from *CSG* to OptiX 7 *IAS GAS*, pipeline creation
- *CSGOptiX7.cu* : compiled into ptx that becomes OptiX 7 pipeline
  - includes QUDARap headers for simulation
  - includes *csg_intersect_tree.h,..* headers for CSG intersection

G4CX

- *G4CXOpticks* : Top level Geant4 geometry interface

Flexible Multi-Package Organization

Code Organized by Dependency Only

- **maximizes: utility, re-use, ease of testing**
- => "GPU" code usable+tested on CPU

Many small header-only implementations

- common CPU/GPU headers

Full re-implementation of Opticks for NVIDIA OptiX 7 API

- **Huge change unavoidable from new OptiX API --> So profit from rethink of simulation code --> 2nd impl advantage**

Old simulation (OptiXRap)	New simulation (QUDARap/qsim.h + CSGOptiX, CSG)
• implemented on top of old OptiX API	• pure CUDA implementation • **OptiX use kept separate, just for intersection**
• monolithic .cu • GPU only implementation • deep stack of support code	• many small headers • many GPU+CPU headers • shallow stack : QUDARap depends only on SysRap
• most code in GPU only context, even when not needing OptiX or CUDA	• strict code segregation ▪ **code not needing GPU in SysRap not QUDARap**
• testing : GPU only, coarse	• **testing : CPU+GPU , fine-grained** • curand mocking on CPU
• limited CPU/GPU code sharing	• maximal sharing : SEvt.hh, sphoton.h, ...
• timeconsuming manual random alignment conducted via debugger	• **new systematic approach to random alignment**

Goals of re-implementation : flexible, modular GPU simulation, easily testable, less code

- code reduction, sharing as much as possible between CPU and GPU
- fine grained testing on both CPU and GPU, with GPU curand mocking
- profit from several years of CUDA experience, eg QSim.hh/qsim.h host/device counterpart pattern:
  - hostside initializes and uploads device side counterpart --> **device side hits ground running**

Two-Level Hierarchy : Instance transforms (TLAS) over Geometry (BLAS)

OptiX supports multiple instance levels : IAS->IAS->GAS BUT: **Simple two-level is faster** : works in hardware RT Cores

AS
Acceleration Structure

TLAS (aka **IAS**)
4x4 transforms, refs to BLAS

BLAS (aka **GAS**)
triangles : vertices, indices
custom primitives : AABB

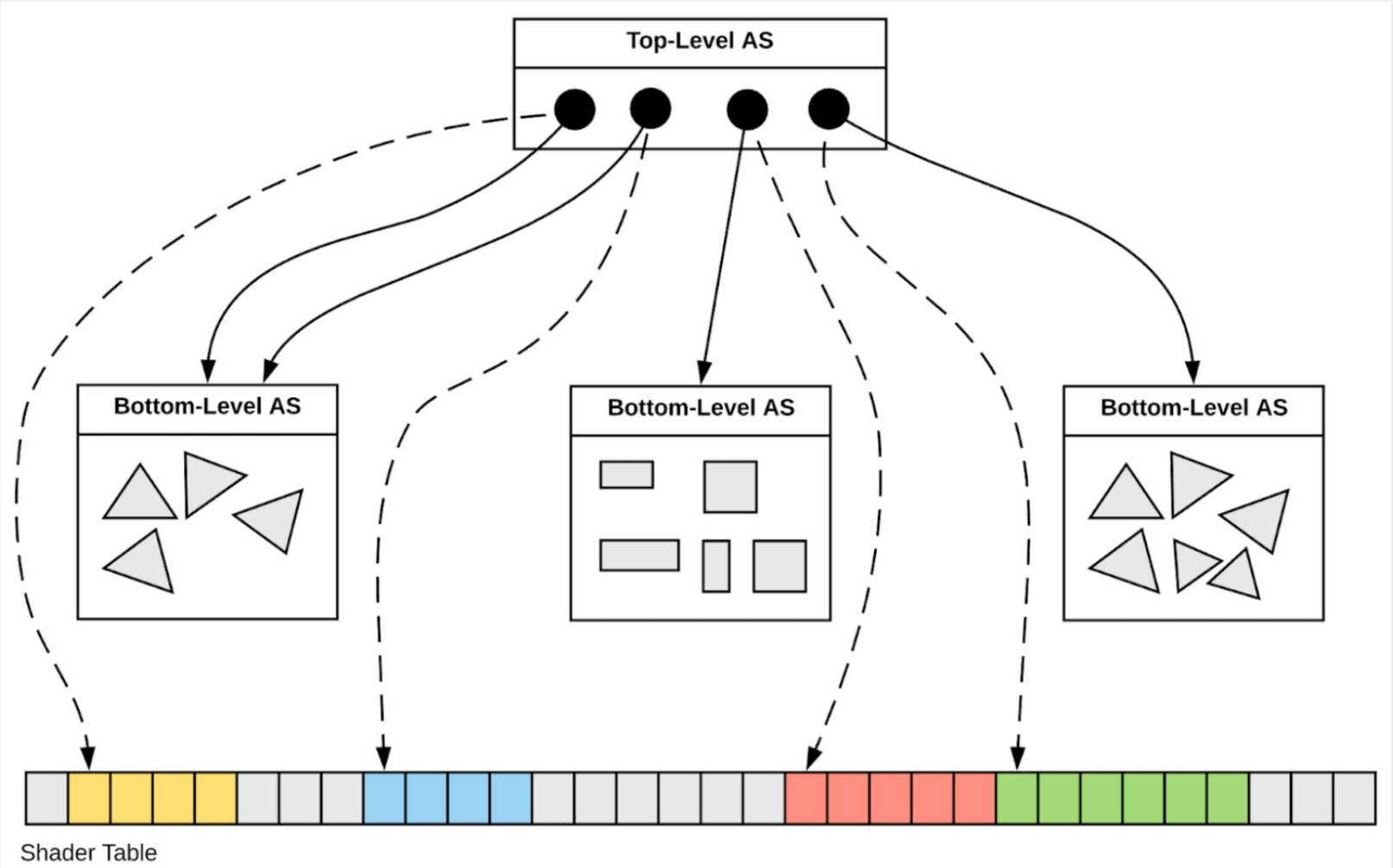
AABB
axis-aligned bounding box

SBT : Shader Binding Table

Flexibly binds together:

1. geometry objects
2. shader programs
3. data for shader programs

Hidden in OptiX 1-6 APIs



https://developer.nvidia.com/blog/introduction-nvidia-rtx-directx-ray-tracing/ ❑

Geometry Model Translation : Geant4 => CSGFoundry => NVIDIA OptiX 7

Geant4 Geometry Model (JUNO: 300k PV, deep hierarchy)

PV	*G4VPhysicalVolume*	placed, refs LV
LV	*G4LogicalVolume*	unplaced, refs SO
SO	*G4VSolid,G4BooleanSolid*	binary tree of SO "nodes"

CSGFoundry Model

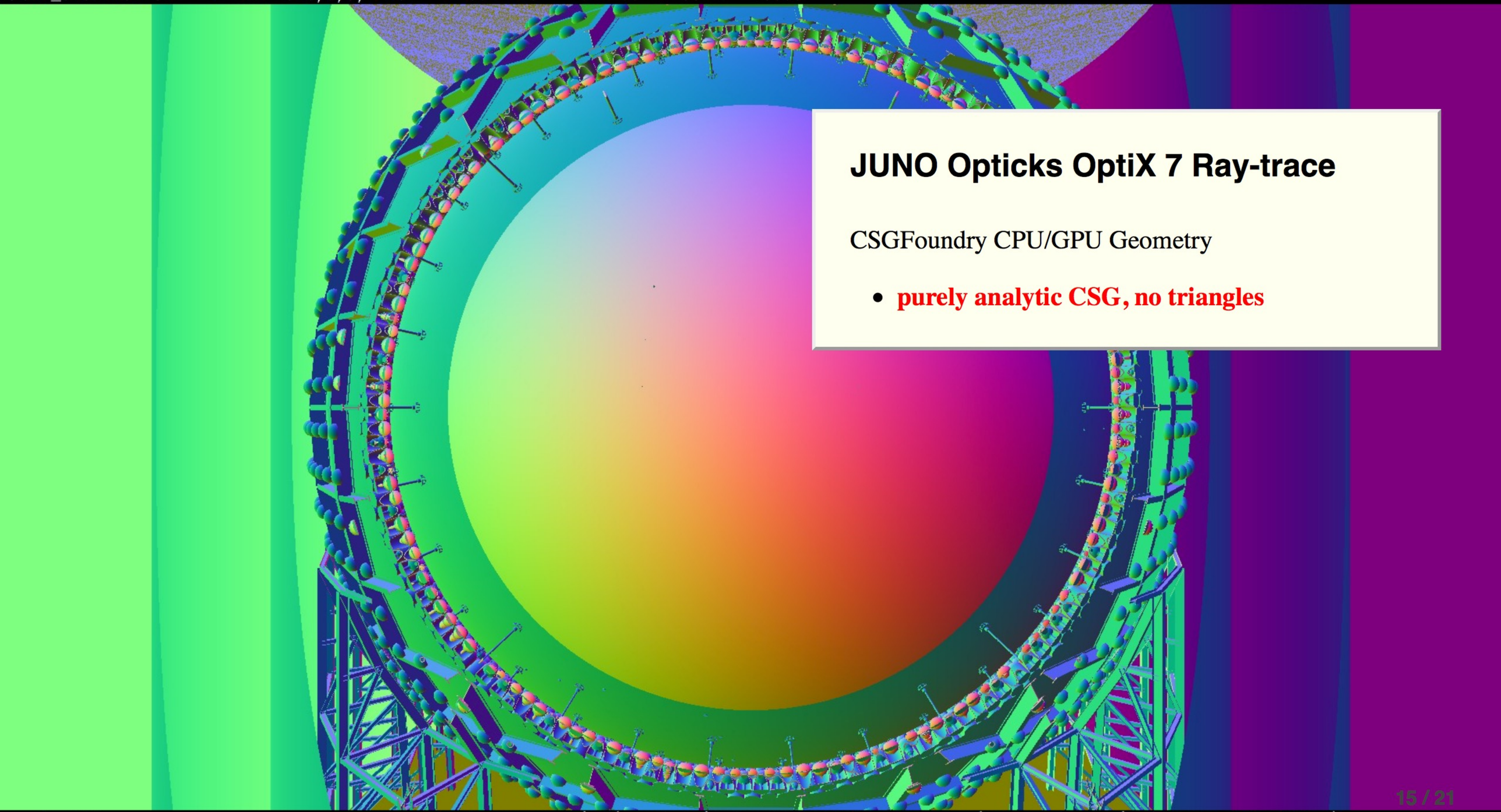
- array-based -> simple serialization + upload
- entire geometry in 4 GPU allocations
- factorized using subtree digests

Opticks CSGFoundry Geometry Model (index references)

struct	Notes	Geant4 Equivalent
CSGFoundry	vectors of the below, easily serialized + uploaded	None
qat4	4x4 transform refs *CSGSolid* using "spare" 4th column	
CSGSolid	refs sequence of *CSGPrim*	Groups of nearby PV, LV + Remainder
	eg JUNO *CSGSolid* numPrim [3089, 5, 11, 14, 6, 1, 1, 1, 1, 130]	
CSGPrim	bbox, refs sequence of *CSGNode*, root of CSG Tree of nodes	root *G4VSolid*
CSGNode	CSG node parameters (JUNO: ~23k *CSGNode*)	node *G4VSolid*

NVIDIA OptiX 7 Geometry Acceleration Structures (JUNO: 1 IAS + 10 GAS, 2-level hierarchy)

IAS	Instance Acceleration Structures	JUNO: 1 IAS created from vector of ~50k *qat4* (JUNO)
GAS	Geometry Acceleration Structures	JUNO: 10 GAS created from 10 *CSGSolid* (which refs *CSGPrim,CSGNode*)



JUNO Opticks OptiX 7 Ray-trace

CSGFoundry CPU/GPU Geometry

- **purely analytic CSG, no triangles**

QUDARap : CUDA Optical Simulation Implementation

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

CPU Pre-Init of GPU Counterpart

hh

instanciate device .h **on host**, upload constituents (eg texture buffers), set constituent device pointers into .h instance, upload .h instance to GPU

h

simple device header, testable on CPU

--> **device side hits ground running**

Aims of counterpart code organization:

- **facilitate fine-grained modular simulation testing**
- bulk of GPU code in simple to test headers
  - many can be tested on CPU
- *QUDARap* does not depend on OptiX -> more flexible -> simpler testing

n-ary CSG Compound "List-Nodes" => Much Smaller CSG trees

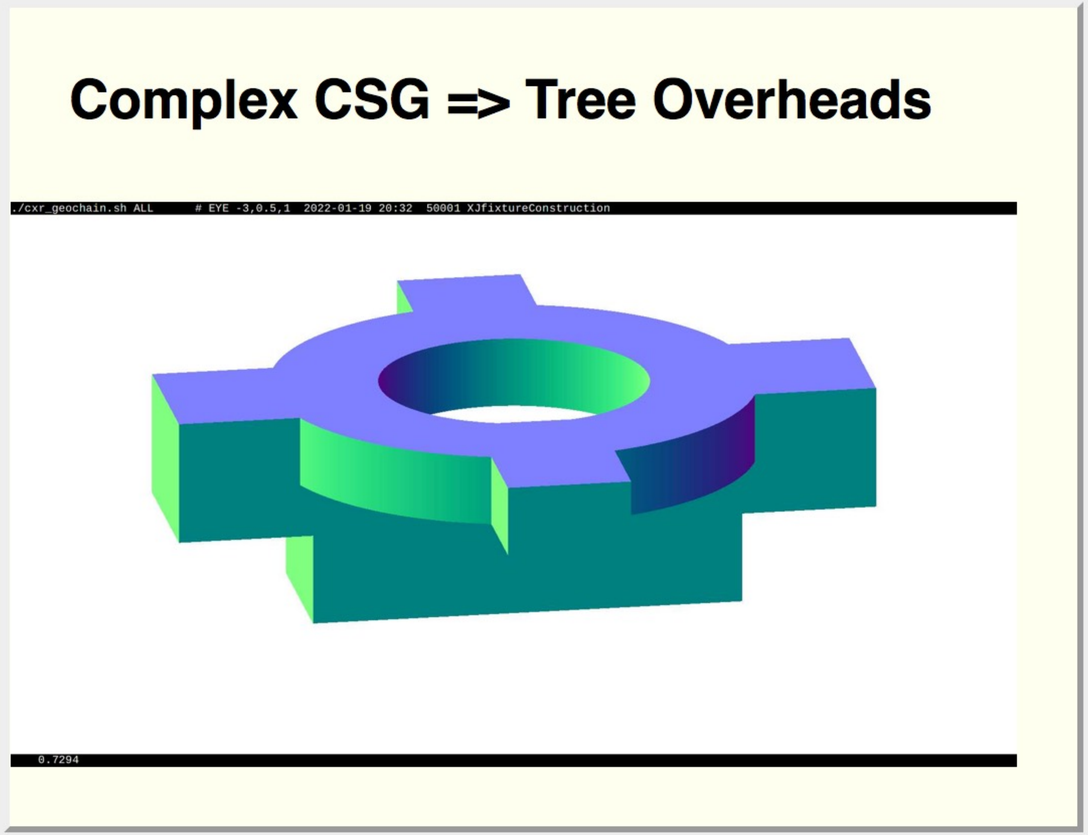
- list-node references sub-nodes by **subNum subOffset**

CSG_CONTIGUOUS Union
  user guarantees contiguous
    - like G4MultiUnion of prim only

CSG_DISCONTIGUOUS Union
  user guarantees no overlaps
    - => simple, low resource intersect
    - eg "union of holes" to be CSG subtracted

CSG_OVERLAP Intersection
  user guarantees overlap
    - eg general G4Sphere: inner radius, thetacut, phicut

Communicate shape more precisely
  => better suited intersect alg => less resources => faster



Generalized Opticks CSG into three levels : tree < node < leaf (avoids recursion in intersect)

- opticks/src/master/CSG/csg_intersect_tree.h ❑ csg_intersect_node.h ❑ csg_intersect_leaf.h ❑

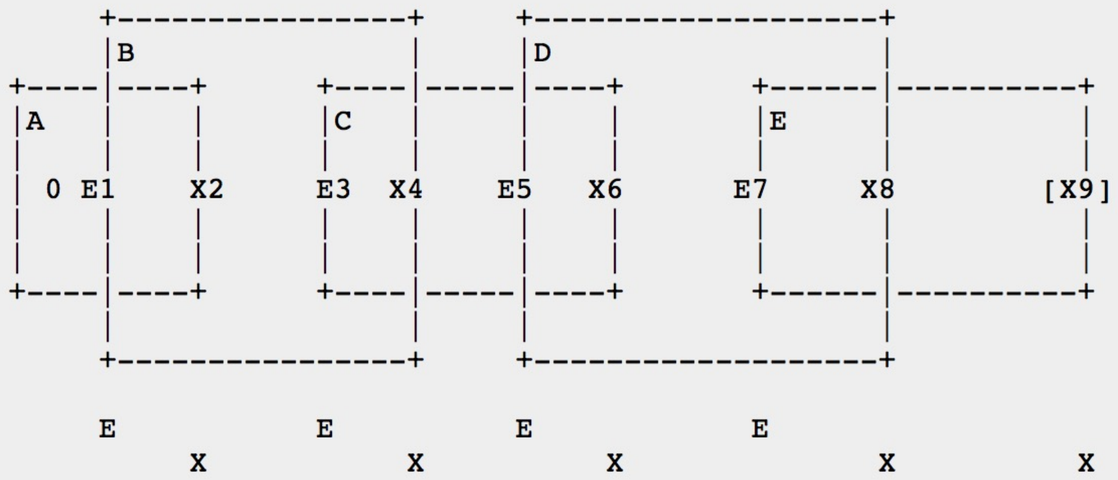
CSG_CONTIGUOUS Union : n-ary (not bin-ary) CSG intersection

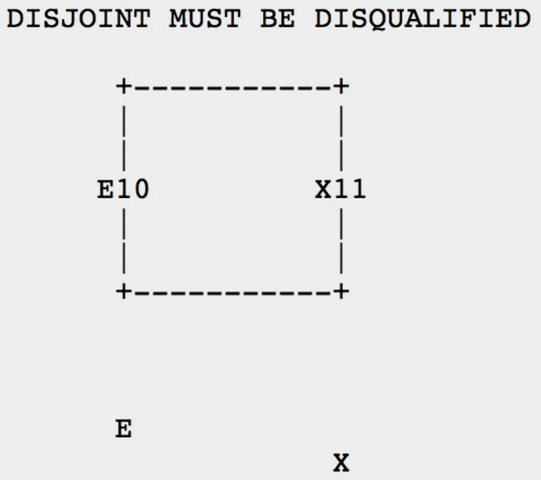
- https://bitbucket.org/simoncblyth/opticks/src/master/CSG/csg_intersect_node.h ❑ **intersect_node_contiguous**

1. *zeroth pass* : find **nearest_enter** and count first exits
2. if zero exits => outside compound => return **nearest_enter**
3. *first pass* : collect enter distances, **farthest_exit**
4. order enter indices making **enter** distances ascend
   - **n-ary : store, sort enters (cf bin-ary : compare two)**
   - **no tree overheads, but must store+sort distances**
5. *2nd pass* : loop over enters in distance order
   - contiguous requirement : **enter < farthest_exit** so far
   - find Exits for Enters that qualify as contiguous, update farthest_exit
6. return **farthest_exit** that qualifies as contiguous

Alg works : but many TODOs

- integer templating : suit resources to shape
- try sort networks, bitonic sort, ...
- compare performance with unbalanced trees
- iterate implementation whilst measuring perf.

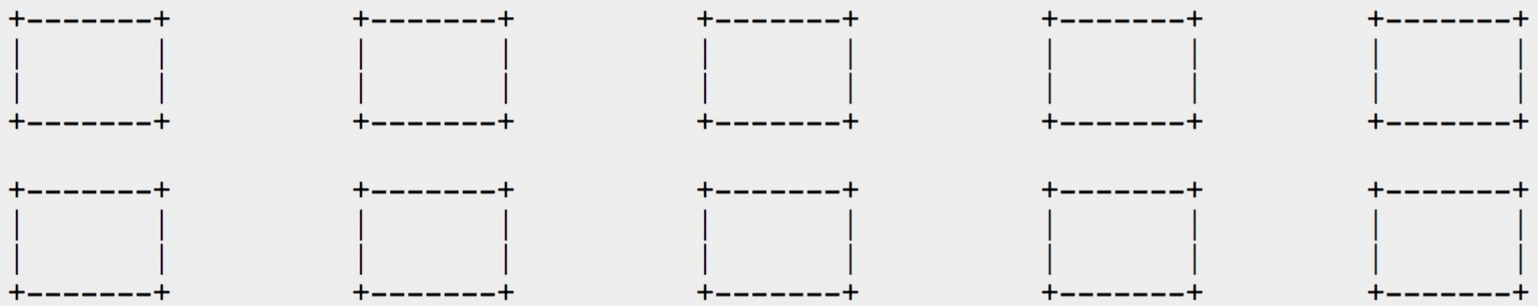




CSG_DISCONTIGUOUS Union : CSG intersection

- https://bitbucket.org/simoncblyth/opticks/src/master/CSG/csg_intersect_node.h ❏ **intersect_node_discontiguous**

User guarantees : absolutely no overlapping between constituents



- => very simple low resource intersection : **closest Enter or Exit**
- **More closely suiting algorithm to geometry => better performance**
- this can help with "holes" subtracted from another solid : the "holes" usually do not overlap

Multi-Layer Thin Film (A,R,T) Calc using TMM Calc (Custom4 Package)

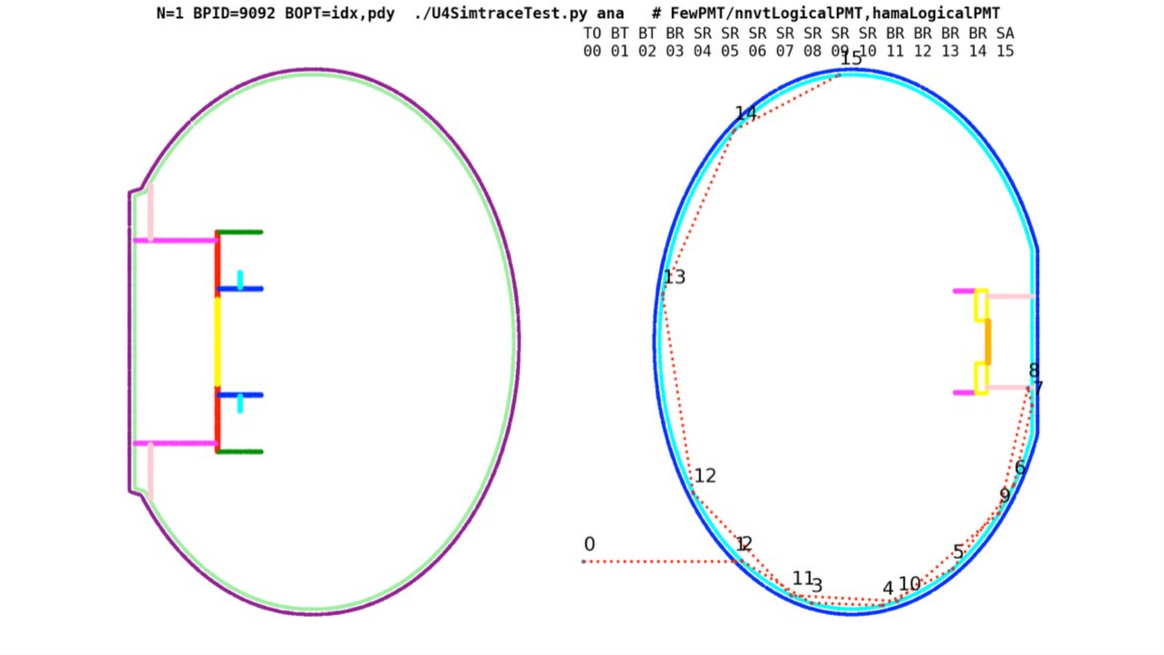
C4OpBoundaryProcess.hh
G4OpBoundaryProcess with C4CustomART.h

C4CustomART.h
integrate custom boundary process and TMM calculation

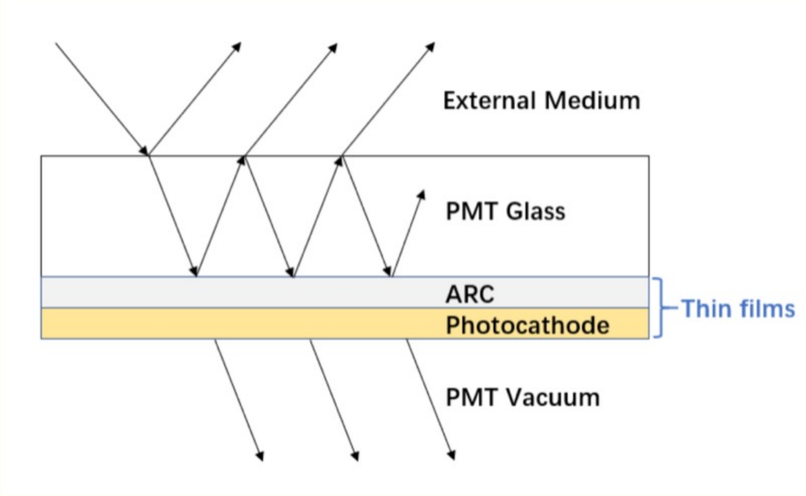
C4MultiLayrStack.h : ***CPU/GPU TMM calculation of (A,R,T)***
based on complex refractive indices and layer thicknesses

- GPU: **using thrust::complex** CPU:**using std::complex**

Custom4: Simplifies JUNO PMT Optical Model + Geometry



TMM : Transfer Matrix Method



multi-layer thin films, coherent calc:

- complex refractives indices, thicknesses
- => (A,R,T) (Absorb, Reflect, Transmit)
- Used from **C4OpBoundaryProcess**

header-only GPU/CPU : **C4MultiLayrStack.h**

https://github.com/simoncblyth/customgeant4/ ❑

Summary and Links

Opticks : state-of-the-art GPU ray traced optical simulation integrated with *Geant4*. Full re-implementation of Opticks geometry and simulation for NVIDIA OptiX 7 completed.

Hidden Benefits of Adopting Opticks

- detailed photon instrumentation, validation
- comparisons find issues with both simulations:
  - complex geometry, overlaps, bugs...

=> using Opticks improves CPU simulation too !!

- NVIDIA Ray Trace Performance continues rapid progress (2x each generation)
- Increasing availability of HW accelerated ray tracing
- NEXT: Opticks production testing, optimization

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
`simon.c.blyth@gmail.com`	any questions