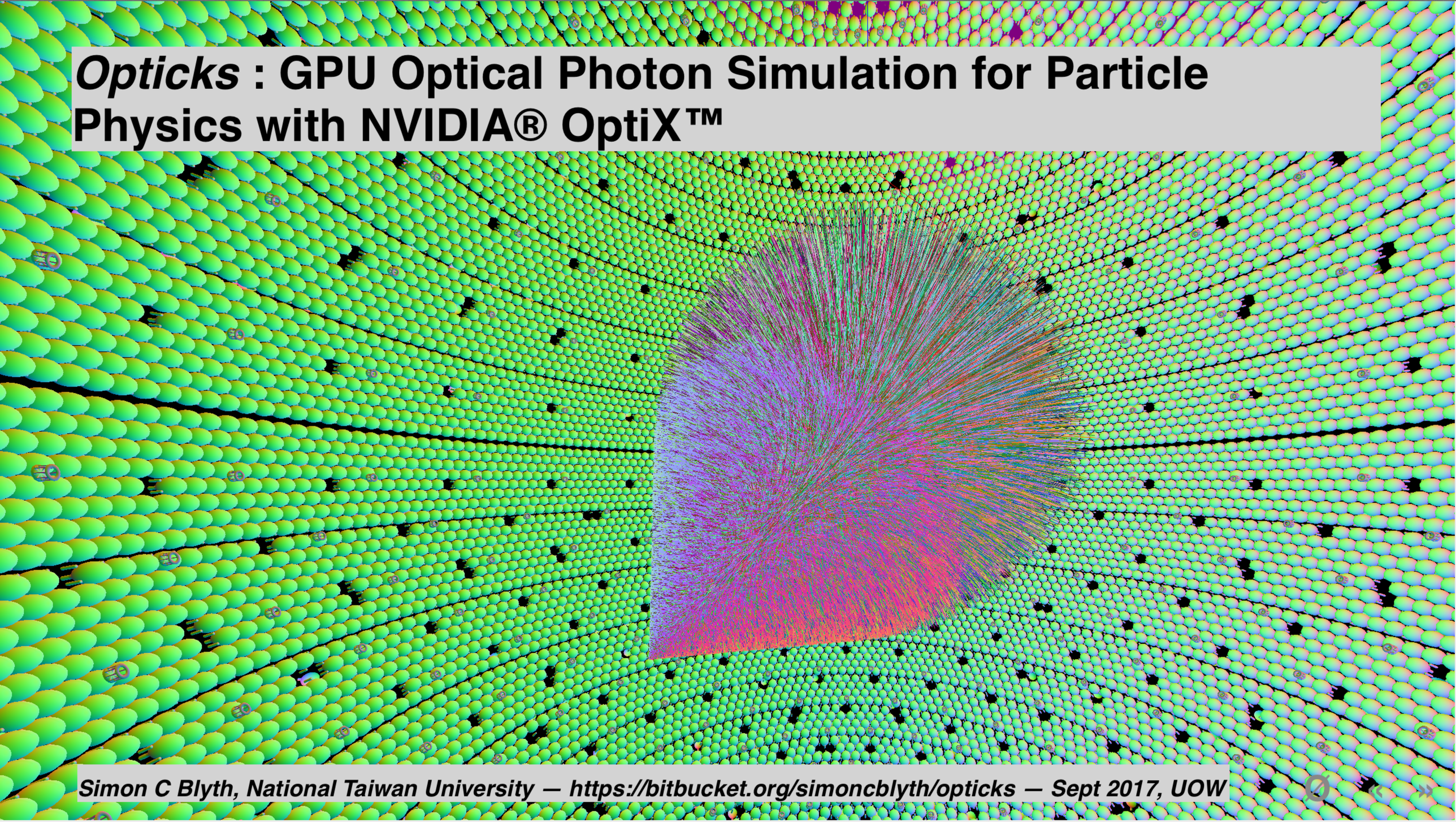
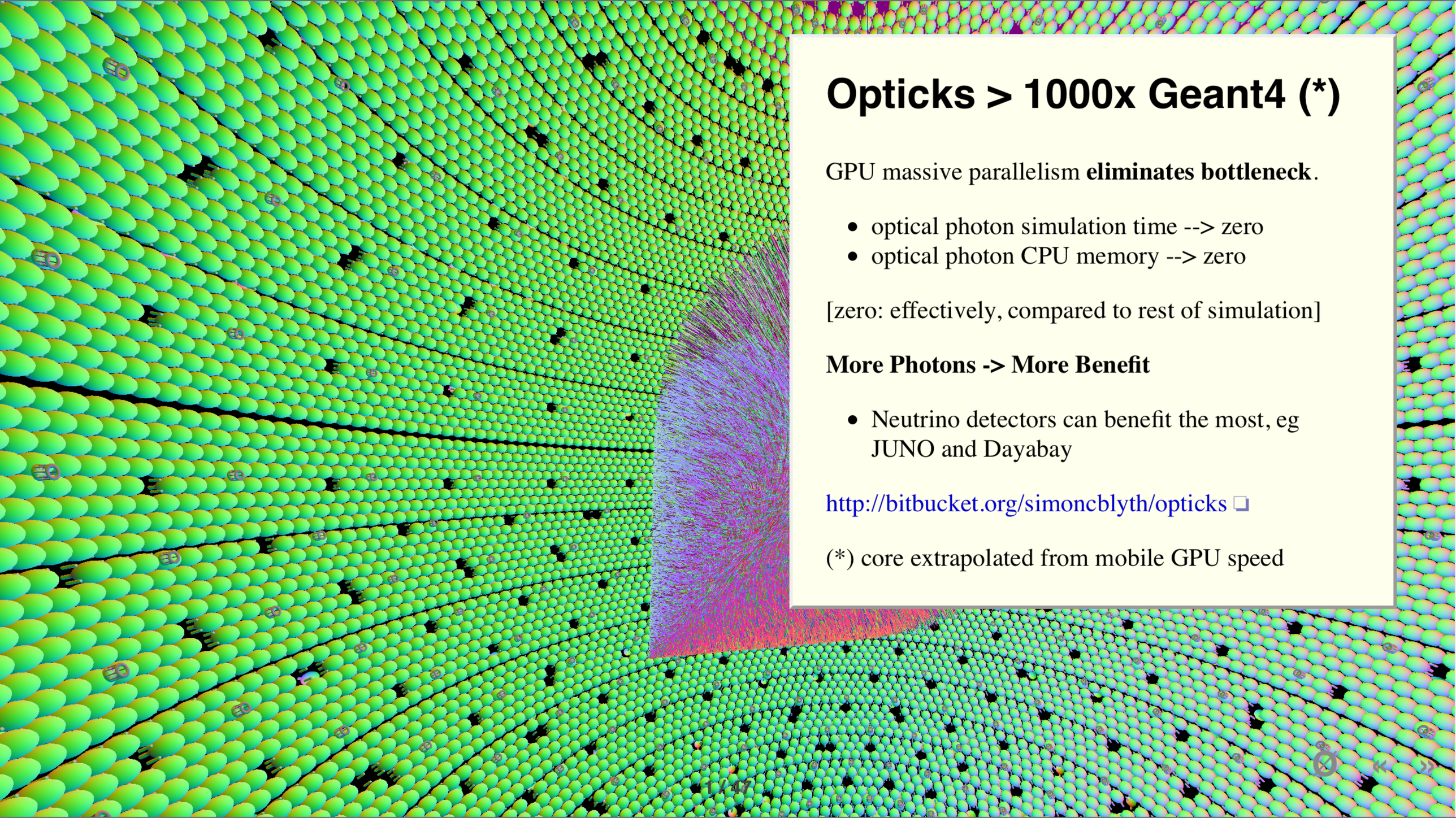
Opticks : GPU Optical Photon Simulation for Particle Physics with NVIDIA® OptiX™



Simon C Blyth, National Taiwan University — https://bitbucket.org/simoncblyth/opticks — Sept 2017, UOW



Opticks > 1000x Geant4 (*)

GPU massive parallelism **eliminates bottleneck**.

- optical photon simulation time --> zero
- optical photon CPU memory --> zero

[zero: effectively, compared to rest of simulation]

More Photons -> More Benefit

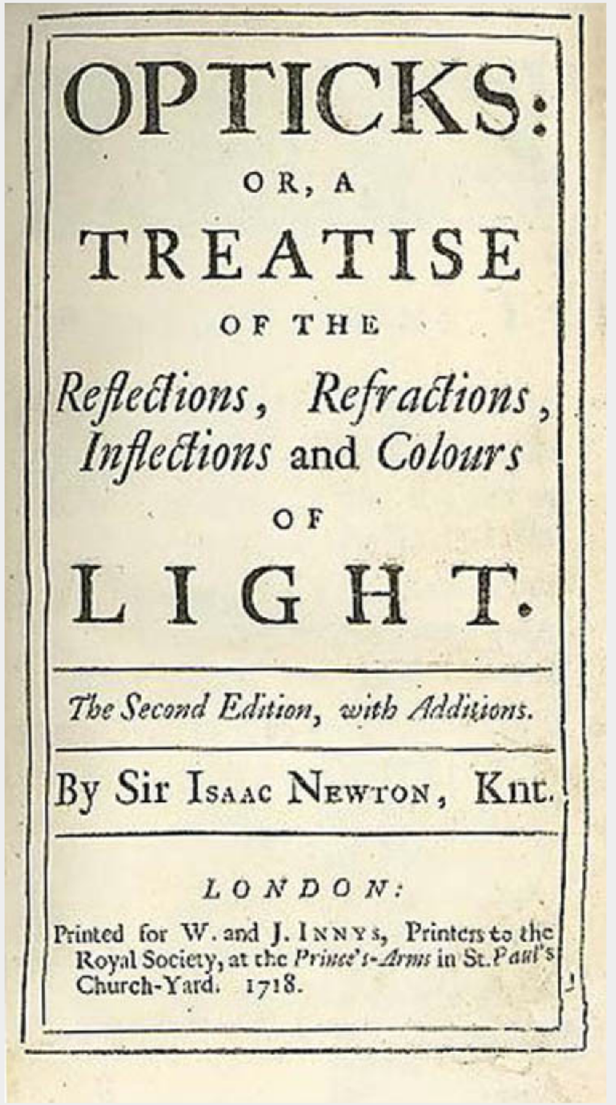
- Neutrino detectors can benefit the most, eg JUNO and Dayabay

http://bitbucket.org/simoncblyth/opticks ❏

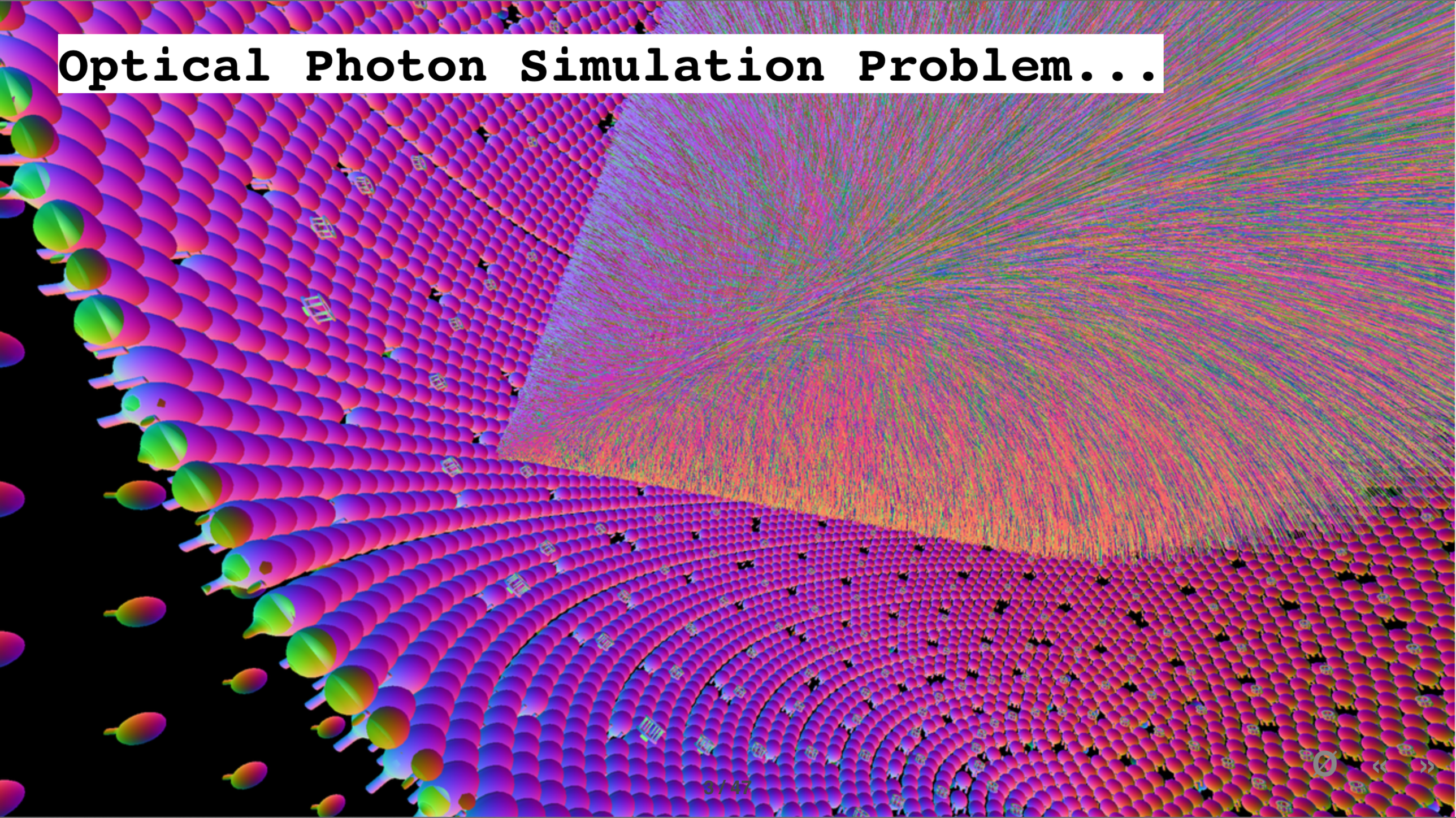
(*) core extrapolated from mobile GPU speed

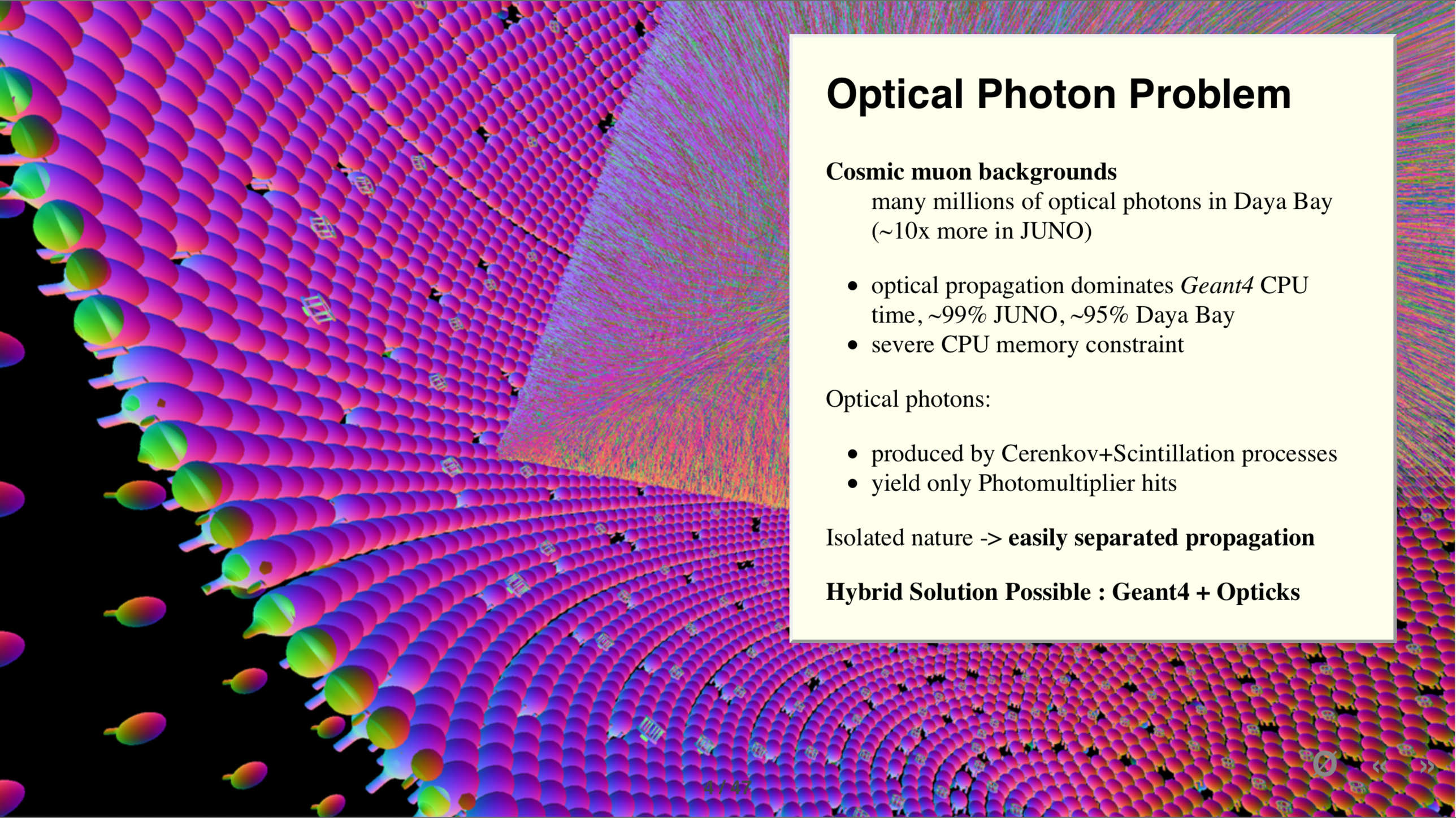
Outline

- Problem and approach to solving
  - Optical Photon Simulation problem
  - Hybrid solution : Geant4 + Opticks
  - NVIDIA OptiX Ray Tracing Engine
- *Opticks* : optical photon simulation with NVIDIA OptiX
  - geometry workflow : geocache, GPU textures
  - large geometry techniques, geometry modelling
  - Geant4/Opticks event workflow
- Validation, performance comparisons
- Summary


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...





Optical Photon Problem

Cosmic muon backgrounds

many millions of optical photons in Daya Bay (~10x more in JUNO)

- optical propagation dominates *Geant4* CPU time, ~99% JUNO, ~95% Daya Bay
- severe CPU memory constraint

Optical photons:

- produced by Cerenkov+Scintillation processes
- yield only Photomultiplier hits

Isolated nature -> **easily separated propagation**

Hybrid Solution Possible : Geant4 + Opticks

Ray Traced Image Synthesis ≈ Optical Photon Simulation

Geometry, light sources, optical physics ->

- pixel values at image plane
- photon parameters at detectors (eg PMTs)

Ray tracing has many applications :

- advertising, design, entertainment, games,...
- BUT : most ray tracers just render images

Ray-geometry intersection

- hw+sw continuously optimized over 30 years
- performance > 100M intersections per second per GPU

NVIDIA OptiX

- general geometry intersection API
- OptiX is to ray tracing what OpenGL is to rasterization

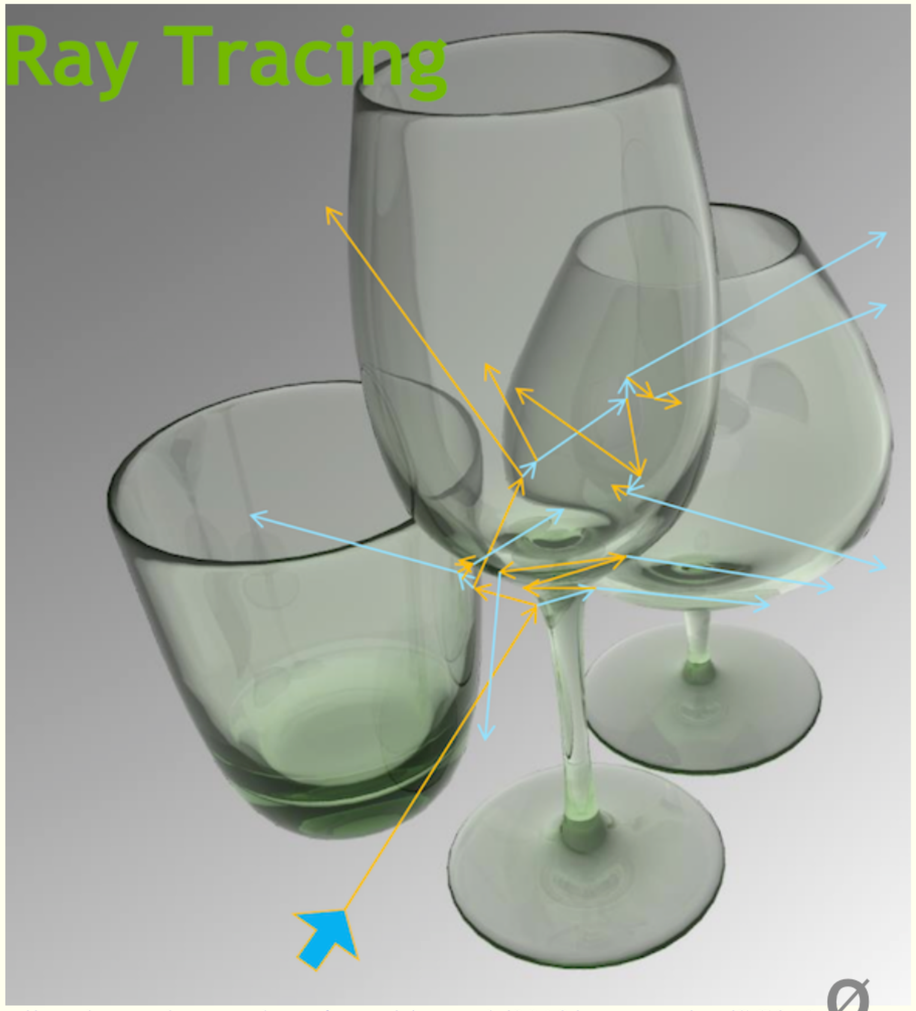
rasterization

project 3D primitives onto 2D image plane, combine fragments into pixel values

ray tracing

cast rays thru image pixels into scene, recursively reflect/refract with geometry intersected, combine returns into pixel values

OptiX Pixel Calculation



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

NVIDIA® OptiX™ Ray Tracing Engine

A software development kit for achieving high performance ray tracing on the GPU.

Image courtesy Tom Grammerstorf

NVIDIA® OptiX™ Ray Tracing Engine

A software development kit for achieving high performance ray tracir

https://research.nvidia.com/publication/optix-general-purpose-ray-tracing-engine

Image courtesy Tom Grammerstorf

NVIDIA® OptiX™

http://developer.nvidia.com/optix ❑

- no rendering assumptions
- **just accelerates ray-geometry intersections**
- compiler optimized for GPU ray tracing
- regular improvements, new GPU tuning
- NVIDIA expertise on GPU/multi-GPU usage
  - ~linear scaling with cores across GPUs
- free to acquire, use, distribute non-commercially
- simple : single-ray programming model

OptiX Programming Model : raytrace pipeline made from CUDA programs

OptiX provides a ray tracing pipeline analogous to OpenGL rasterization pipeline.

OptiX programs used by Opticks photon simulation

Ray Generation
: coordinate: generate, propagate, write hits

Geometry Intersection
: mesh triangle or analytic CSG tree intersection

Closest Hit
: pass intersect to generation (distance, normal, identity)

Handled by NVIDIA OptiX:

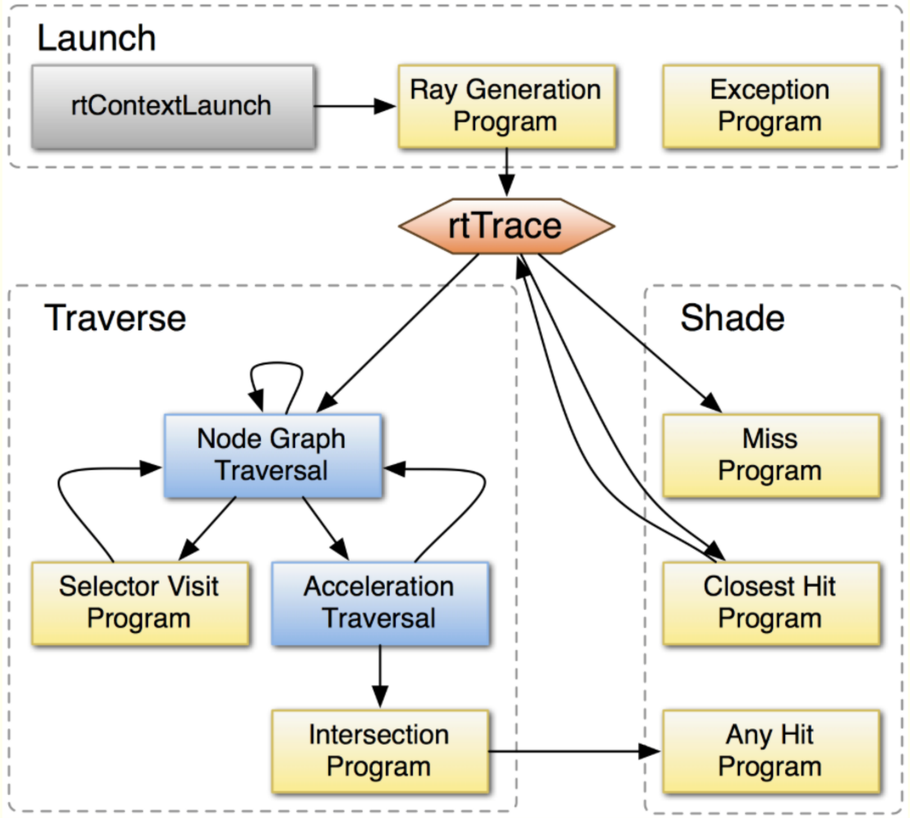
- acceleration structure creation + traversal
- instanced sharing of geometry + acceleration structures
- domain optimized combination of CUDA programs into kernel
- cross GPU work scheduling

Everything else : user supplied

https://research.nvidia.com/sites/default/files/publications/Parker10Optix_1.pdf

OptiX Control Flow

- Yellow: User supplied, Blue: OptiX internals



Opticks : GPU Optical photon simulation with NVIDIA OptiX. How ?

Optical Photon Generation

- translate G4Cerenkov, G4Scintillation optical photon generation loop into parallel CUDA form (only generation loop ported)
- "genstep" parameters collected, copied to GPU for propagation at end of event

Optical Photon Propagation + Scintillator Reemission

- CUDA port of G4OpAbsorption, G4OpRayleigh, G4OpBoundaryProcess (small subset for optical surfaces in use)
- scintillator reemission handled as fraction of bulk absorbed being "reborn" within same GPU thread

Collect PMT hits into standard G4 Hit collections

- just photomultiplier hits copied to CPU

G4 "Context" -> GPU

Straightforward : just translation

- translation of optical physics to CUDA
- interleave material/surface properties into GPU textures

Difficult : novel development was required

- develop general GPU CSG tree intersection
- develop auto conversion from GDML
- ~10 CUDA ray-primitive intersection programs: sphere, box, cone, torus, ...

Requires : Geometry converted into GPU appropriate form

- geometry serialized into buffers, uploaded to GPU at initialization
- CUDA intersect + bounding box programs using geometry buffers communicate geometry to OptiX
- general ray intersection with CSG trees implemented within OptiX primitive

Geometry : Geant4 -> Opticks Geocache -> OptiX GPU

Geocache -> few seconds startup, instead of few minutes

- **Benefit from constant nature of Geometry**

Once per geometry : Export, Construct Geocache

- G4 Export : Analytic (GDML) + Triangulated (G4DAE)
- parse G4DAE + GDML into volume trees
- find repeated geometry "instances" (progeny digests)
- construct material/surface property arrays
- full geometry written to NumPy `.npy` serialization files

Every run : upload geocache + make acceleration structure

- load geocache `.npy` files
- upload OptiX textures to GPU
- upload OptiX geometry buffers to GPU
- intersect + bounding box programs -> OptiX acceleration structures
- complex shapes modelled via CSG boolean combinations

https://bitbucket.org/simoncblyth/g4dae

Volumes -> Boundaries

Ray tracing favors Boundaries

Material/surface boundary representation (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

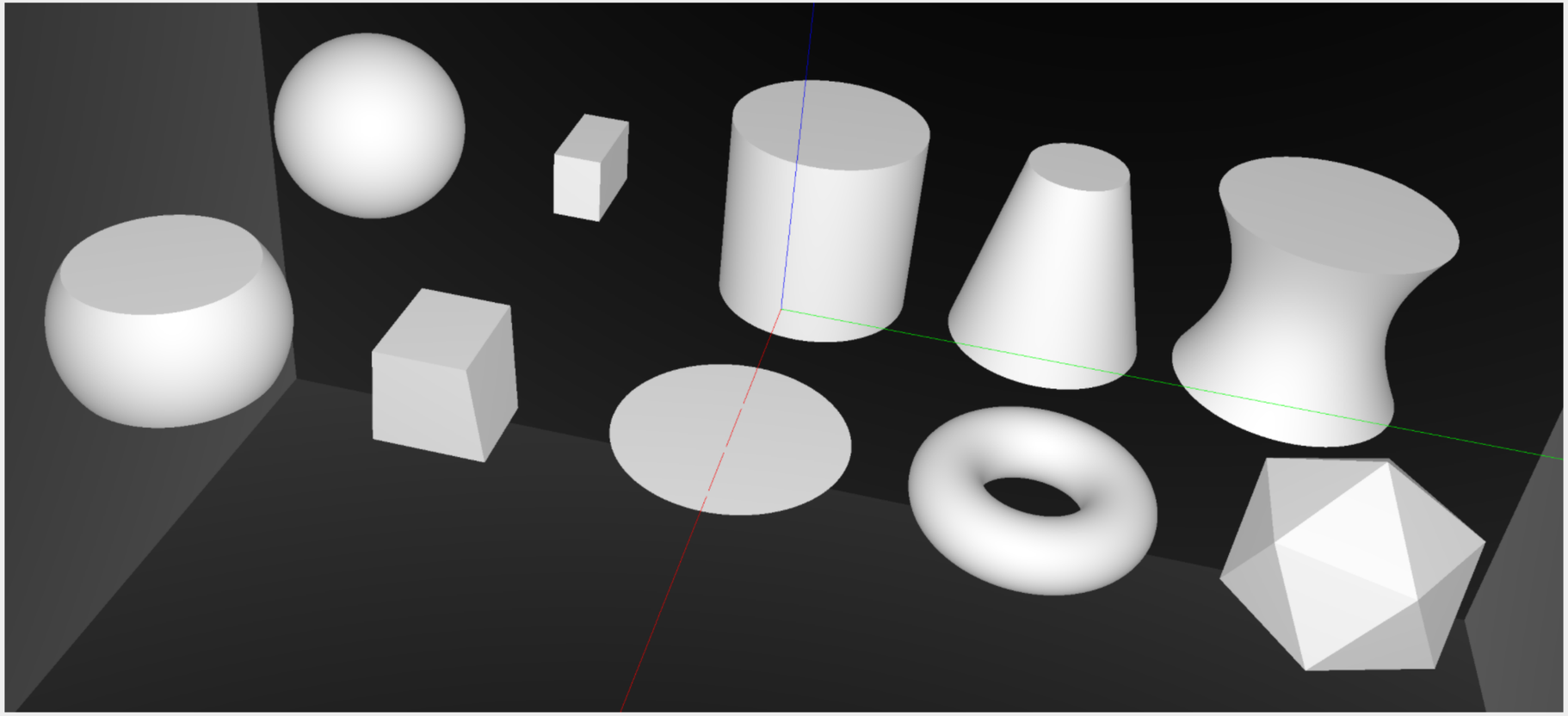
- ray primitive intersection -> boundary index
- texture lookup -> material/surface properties

GPU textures also used for:

- scintillator reemission wavelength generation
- standard illuminant Plankian wavelength gen

Opticks : GPU Geometry starts from ray-primitive intersection

- 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



Ray intersection with general CSG binary tree solids within OptiX

Performance on GPU requires

- massive parallelism -> more the merrier
- low register usage -> keep it simple
- small stack size -> **avoid recursion** : OptiX demands: no recursion in intersect

Approach (details in backup)

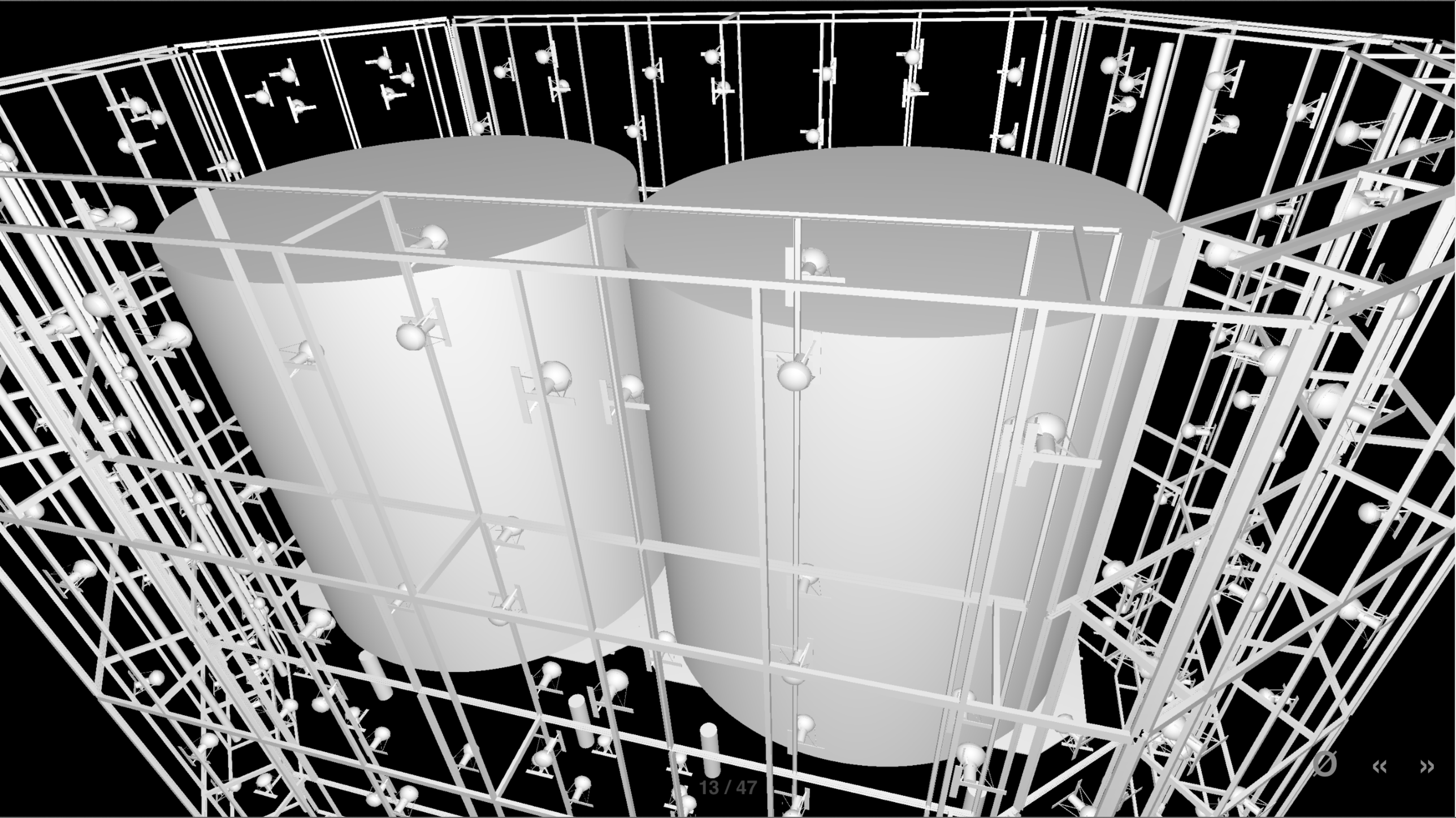
- Complete Binary Tree Serialization
  - postorder tree traversal, just by bit twiddling (avoids recursion)
- Ray Tracing CSG Objects Using Single Hit Intersections (A. Kensler)[1]
  - binary intersect classification, lookup table
  - avoids lots of state, BUT: sometimes advance **t_min** and re-intersect
  - converted recursive pseudocode into "Evaluative" implementation [2]
- Auto translation of GDML into GPU appropriate form
  - entire geometry translated, serialized, uploaded to GPU

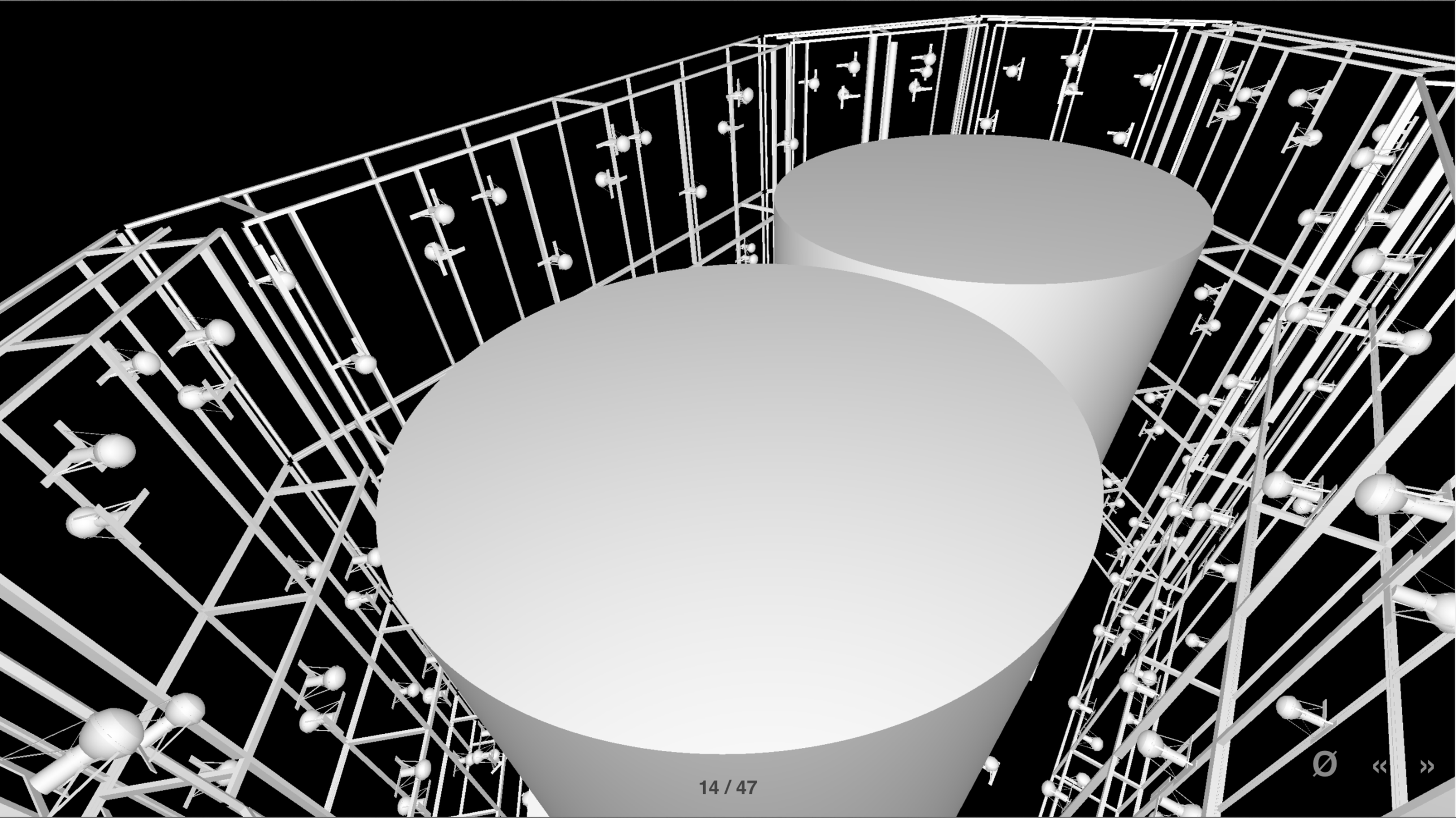
Very close GPU/CPU match is possible

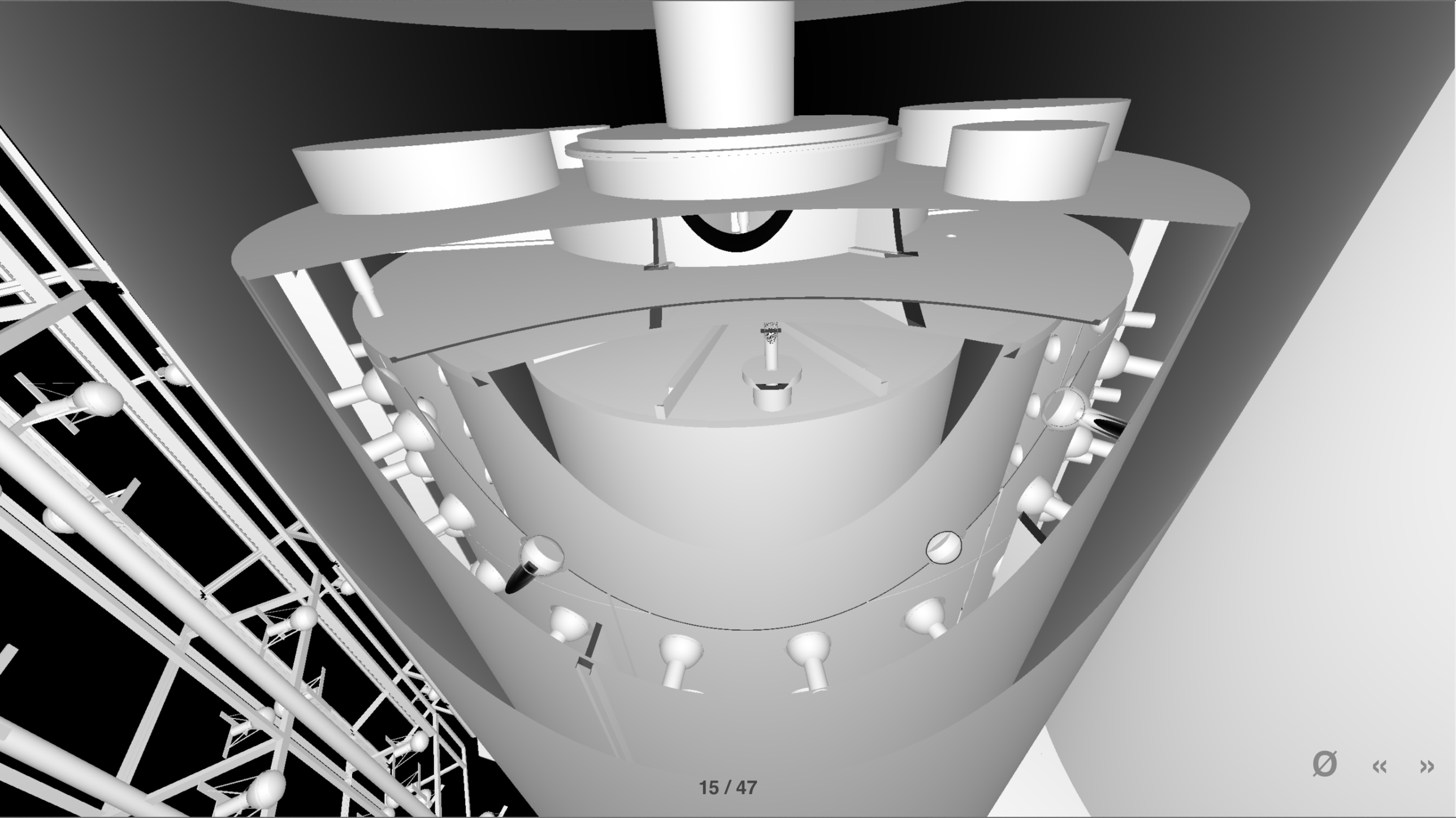
- same analytic geometry
- same optical physics

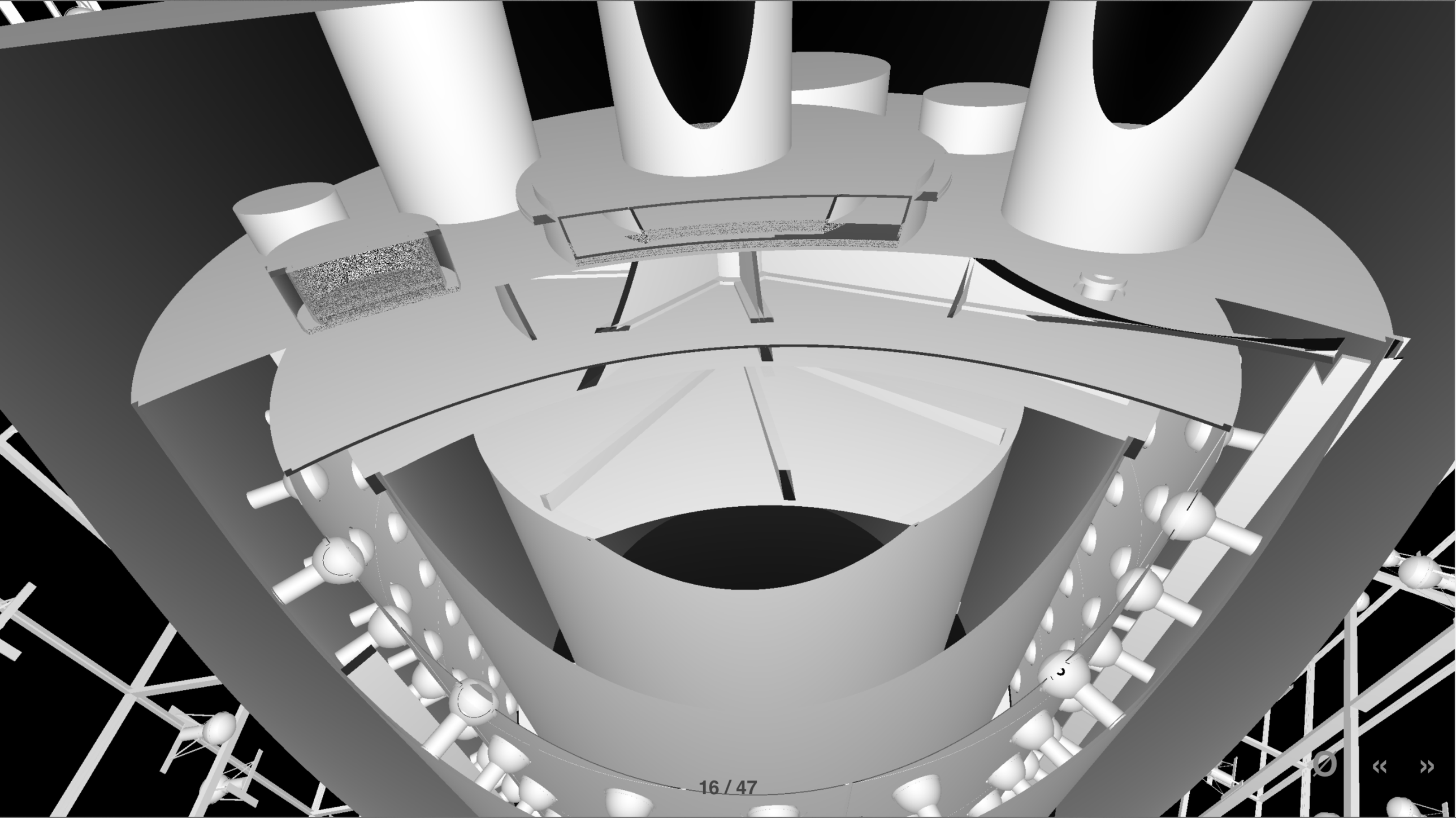
[1] Ray Tracing CSG Objects Using Single Hit Intersections, Andrew Kensler (2006)
with corrections by author of XRT Raytracer http://xrt.wikidot.com/doc:csg

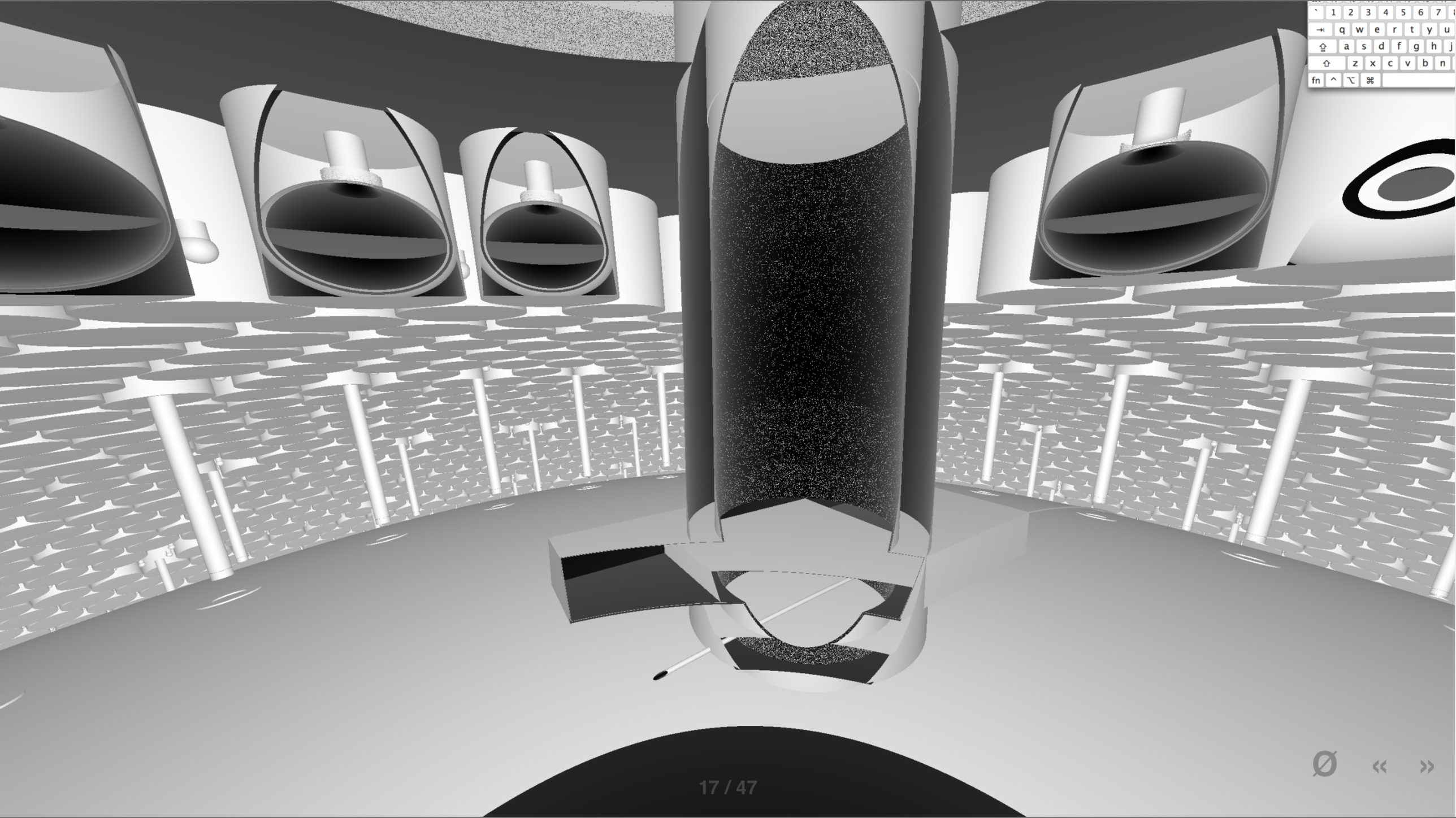
[2] Similar to binary expression tree evaluation using postorder traverse.

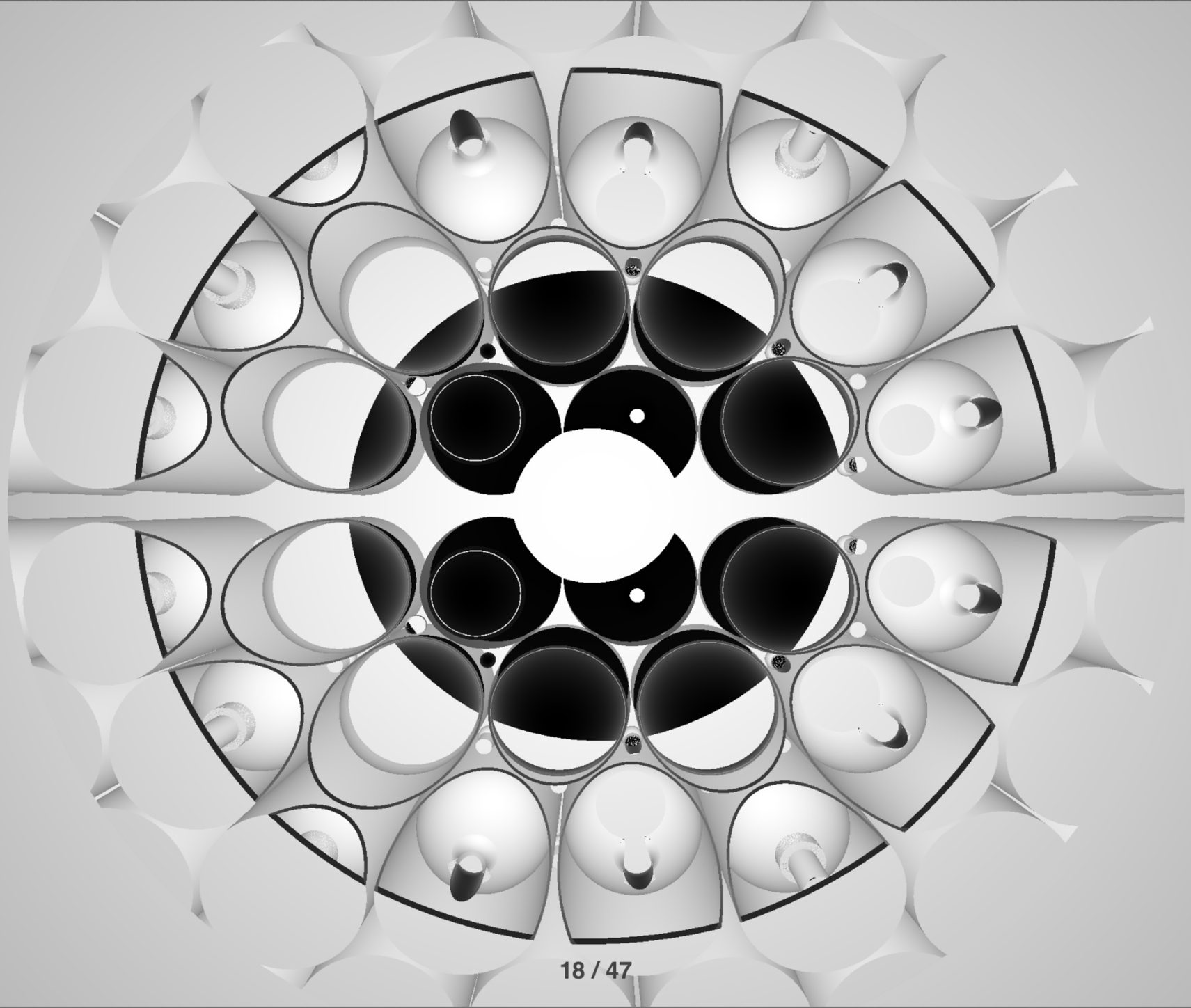












Large Geometry Techniques : Instancing Mandatory

Geometry analysed to find repeats

JUNO: 18k 20" PMTs, 36k 3" PMTs

Instances used by:

- OptiX geometry
- OptiX acceleration structures
- OpenGL visualization

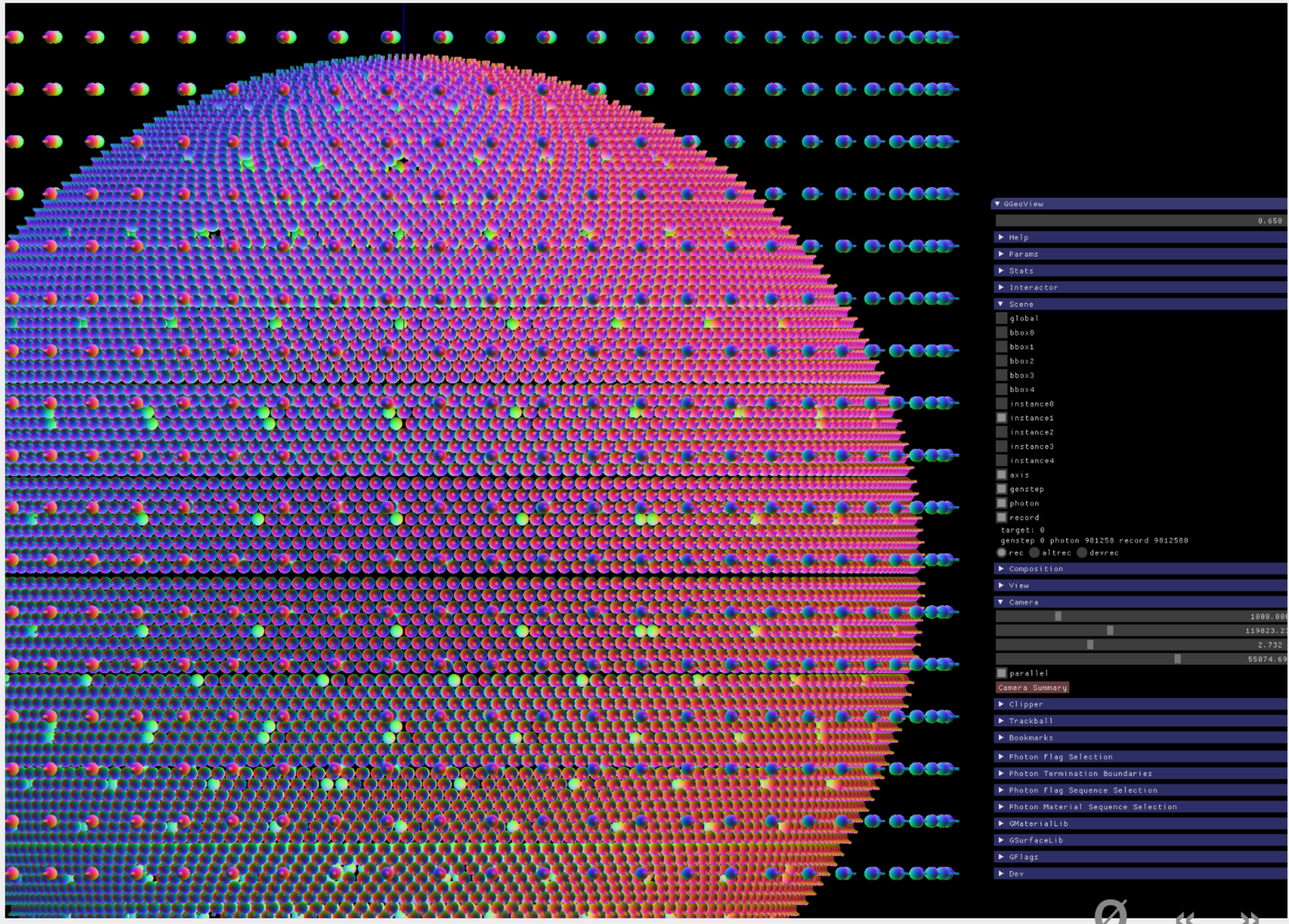
Advantages

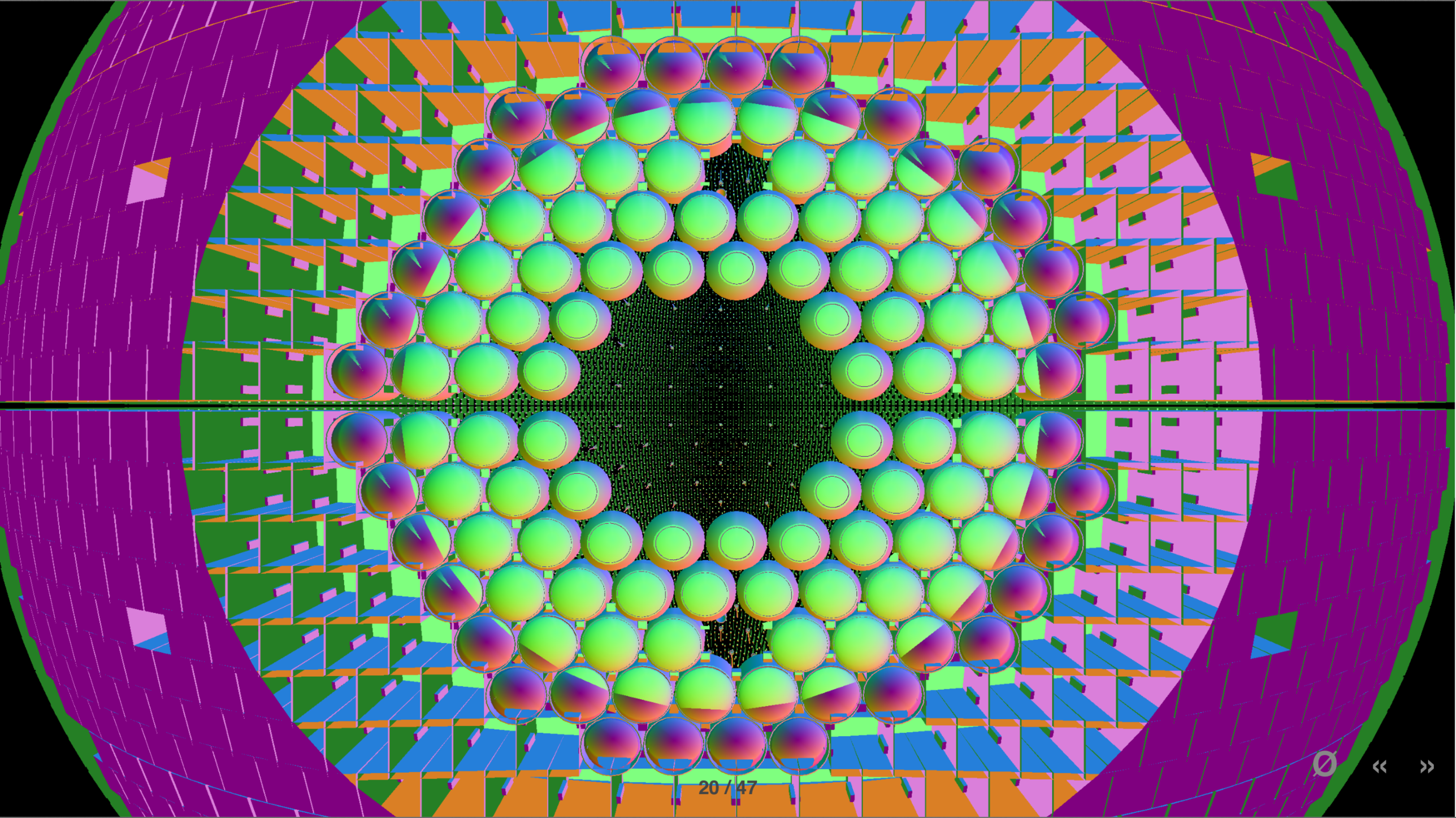
- drastic reduction in GPU memory
- one set of vertices for each PMT type
- 4x4 matrices position each PMT

Viz Optimizations (OpenGL 4+)

Use geometry shader transform feedback:

- cull non-visible instances
- level of detail (LOD) meshes
  - full/simplified/bbox
- switch mesh based on distance to PMT





Hybrid Geant4/Opticks Event Workflow

Geant4/Detector Simulation

- modified Scintillation, Cerenkov processes
  - collect *genstep*
  - skip optical photon generation

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)

Geant4/Detector Simulation

- populate standard hit collections
- subsequent electronics simulation proceeds unaltered

Multi-event handling

- reuse/resize OptiX buffers for each event

GPU Resident Photons

Seeded on GPU

associate photons -> gensteps (via seed buffer)

Generated on GPU, using gensteps:

~*Stacks* collected before photon generation:

- number of photons to generate
- start/end position of step
- other quantities needed for GPU generation

Propagated on GPU

Only photons hitting PMTs copied to CPU

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

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

Idealized geometry tests : photon generation, propagation, reemission

Idealized "tconcentric" scintillator detector avoids any geometry issues, tests optical physics in isolation

Single executable (cfg4 package):

- performs both pure G4 and hybrid G4+Opticks simulations
- writes two events recording up to 16 steps of each photon
- photons indexed on GPU by history and material sequences
- history category counts comparison, Opticks/G4 chi2/df ~ 1.0
- position, time, polarization, wavelength recorded at each step

point-by-point chi2-distance comparisons of 8 photon properties for top 100 history categories

- details in backup

NEXT STEPS

- JUNO integration + full JUNO geometry validation

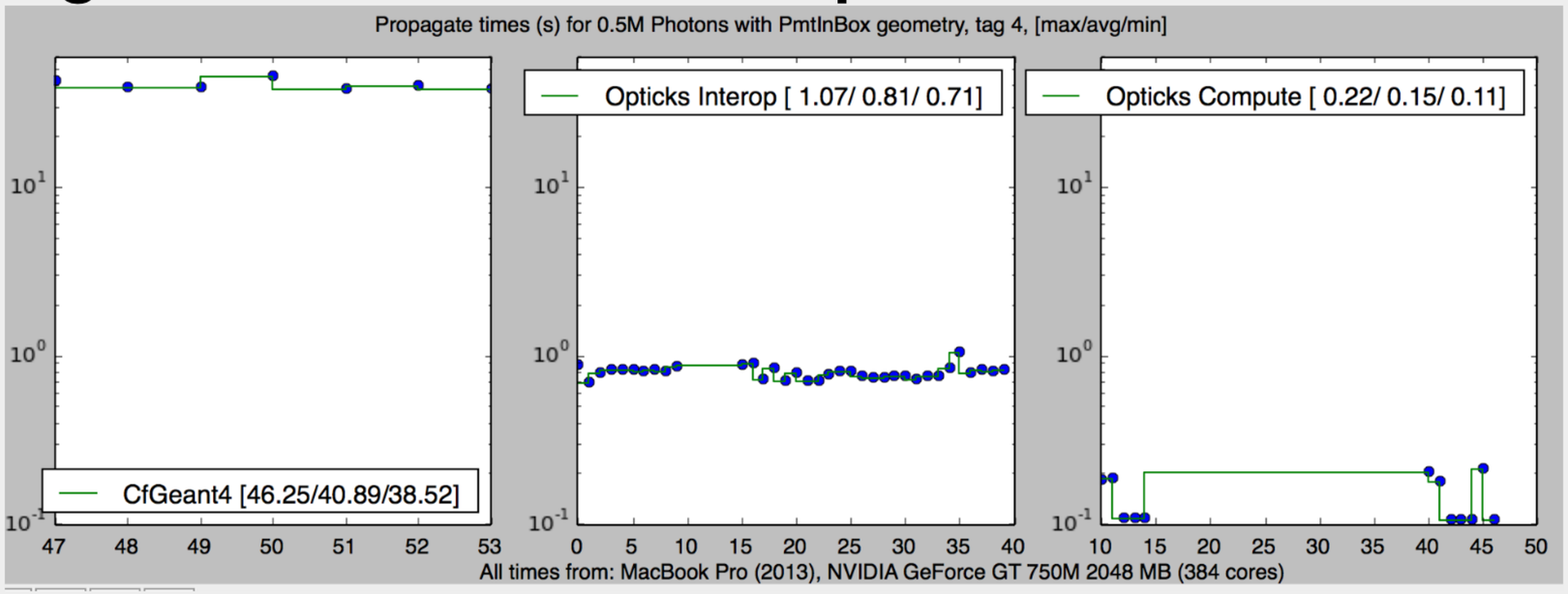
https://bugzilla-geant4.kek.jp/show_bug.cgi?id=1275

Match achieved after many fixes

- scattering, was comparing different implementations
- diffuse reflection, ported G4 approach
- reemission "rejoining"
- missing optical surfaces
- truncation recording discrepancy
- group velocity of wrong material after refraction (G4 issue 1275)
- interpolation mismatches

Work guided by the next largest chi2 contributor

Photon Propagation Times Geant4 cf Opticks



Test	Geant4 10.2	Opticks Interop	Opticks Compute
Rainbow 1M(S)	56 s	1.62 s	0.28 s
Rainbow 1M(P)	58 s	1.71 s	0.25 s
PmtInBox 0.5M	**41 s**	0.81 s	**0.15 s**

- **Opticks > 200X Geant4** with only 384 core mobile GPU[1] (multi-GPU workstation up to 20x more cores)

- **Interop** uses OpenGL buffers allowing visualization, **Compute** uses OptiX buffers
- **Interop/Compute** : perfectly identical results, monitored by digest

[1] MacBook Pro (2013), NVIDIA GeForce GT 750M, 2048 MB, 384 cores

Summary

Opticks enables *Geant4* based simulations to benefit from optical photon simulation **taking effectively zero time and zero CPU memory**, due to massive parallelism made accessible by NVIDIA OptiX. GDML detector geometry is auto translated into a GPU optimized analytic form, equivalent to the source geometry.

Overview

- **Opticks 200x Geant4** with mobile GPU
- Expect: **Opticks > 1000x Geant4** (with workstation GPUs)
- photon propagation time --> zero
- automated full fidelity translation of GDML

- The more photons the bigger the overall speedup (99% -> 100x)
- Drastic speedup -> better detector understanding -> greater precision
- Simulations limited by optical photons can benefit greatly from Opticks+Geant4

https://bitbucket.org/simoncblyth/opticks

List of "backup" slides

CSG

- Constructive Solid Geometry (CSG) : Shapes defined "by construction"
- CSG : Which primitive intersect to pick
- Ray Tracing CSG Objects Using Single Hit Intersections (A. Kensler)
- CSG Complete Binary Tree Serialization -> simplifies GPU side
- Evaluative CSG intersection Pseudocode : recursion emulated
- Opticks CSG Primitives : Closed Solids, Consistent Normals
- Opticks CSG Primitives : What is included
- Opticks CSG : Balancing Deep Trees Drastically Improves Performance
- Dayabay ESR reflector : Deep CSG tree : disc with 9 holes
- Opticks CSG Serialized into OpticksCSG format (numpy buffers, json)

Validation

- tconcentric : spherical GdLS/LS/MineralOil
- tconcentric : Opticks/Geant4 chi2 comparison
- tconcentric : Opticks/Geant4 distrib chi2/df ~ 1.0
- PMT Opticks/Geant4 step distribution comparison TO BT [SD]
- PMT Opticks/Geant4 step distribution comparison : chi2/ndf
- Opticks/Geant4 Rainbow Step Sequence Comparison
- 1M Rainbow S-Polarized, Comparison Opticks/Geant4
- Compare Opticks/Geant4 Simulations with Simple Lights/Geometries

Misc

- OptiX Performance Scaling with GPU cores
- Torus : much more difficult/expensive than other primitives
- Geometry Modelling : Tesselated vs Analytic Photomultiplier Tubes
- Hybrid Geant4/Opticks Event Workflow
- Open Source Opticks

Constructive Solid Geometry (CSG) : Shapes defined "by construction"

Simple *by construction* definition, implicit geometry.

- **A, B** implicit primitive **solids**
- **A + B** : union (OR)
- **A * B** : intersection (AND)
- **A - B** : difference (AND NOT)
- **!B** : complement (NOT) (inside <-> outside)

CSG expressions

- non-unique: **A - B == A * !B**
- represented by binary tree, primitives at leaves

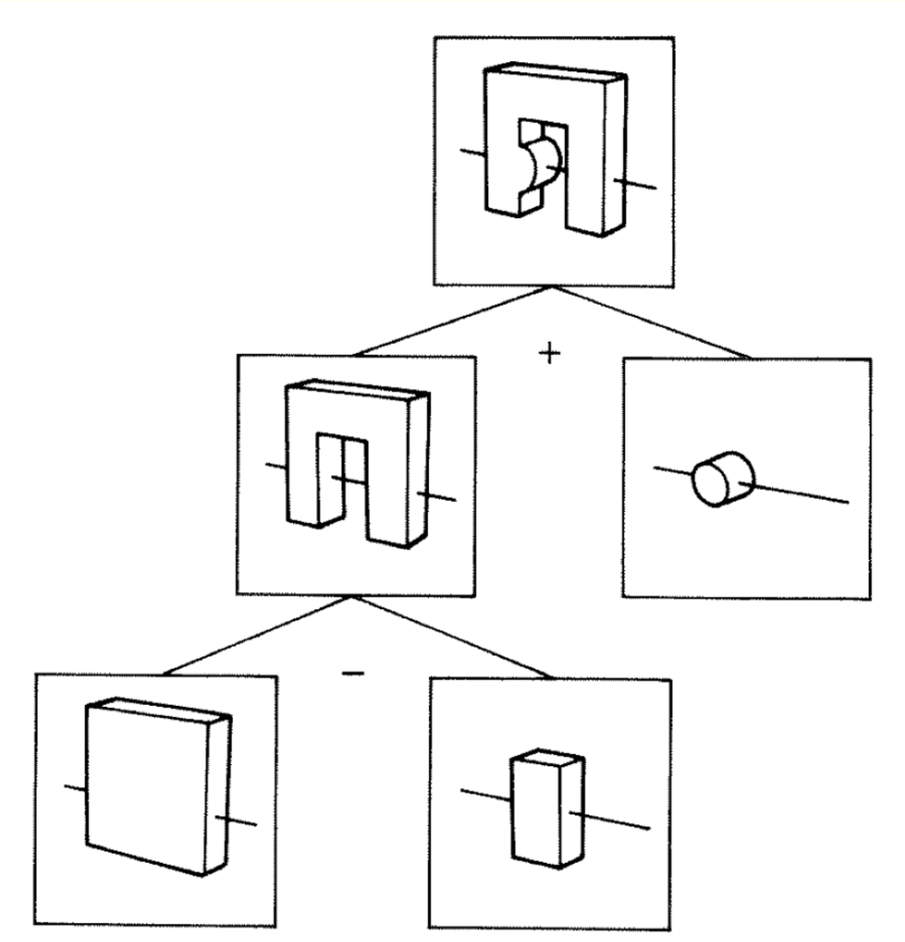
3D Parametric Ray : **ray(t) = r0 + t rDir**

Ray Geometry Intersection

- primitive : find *t* roots of implicit eqn
- composite : pick primitive intersect, depending on CSG tree

How to pick exactly ?

CSG Binary Tree



Primitives combined via binary operators

CSG : Which primitive intersect to pick ?

Classical Roth diagram approach

- find all ray/primitive intersects
- recursively combine inside intervals using CSG operator
- works from leaves upwards

Computational requirements:

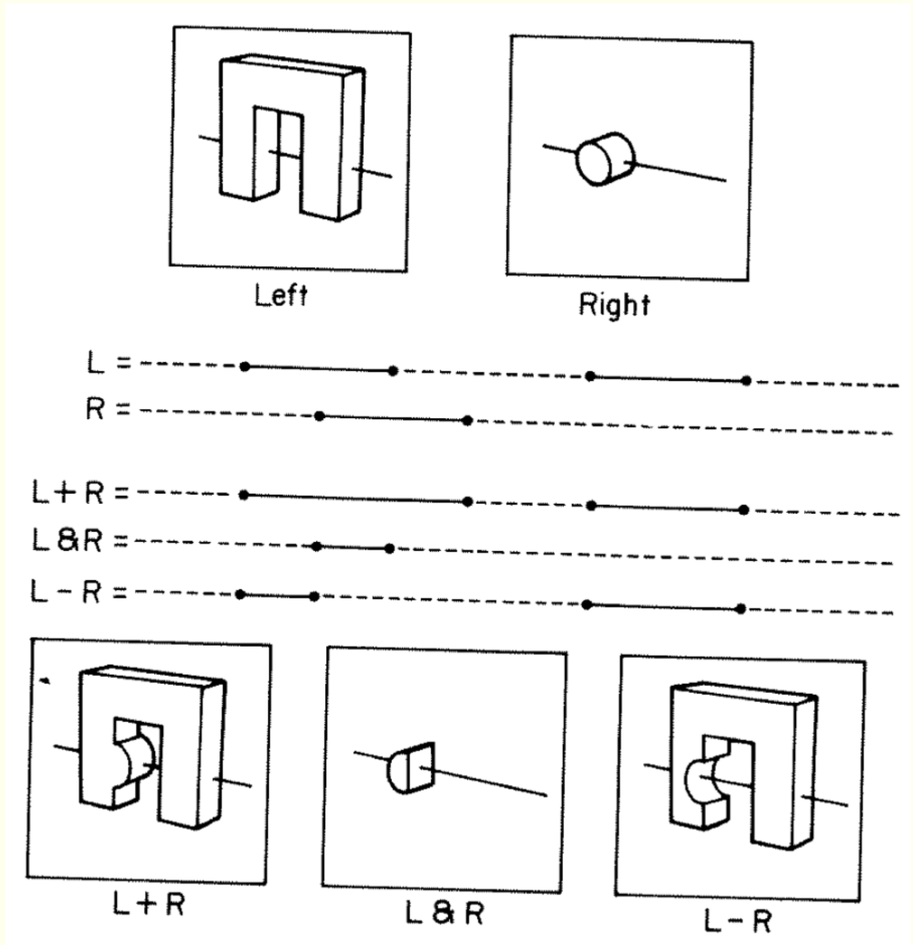
- find all intersects, store them, order them
- recursive traverse

BUT : **High performance on GPU requires**:

- massive parallelism -> more the merrier
- low register usage -> keep it simple
- small stack size -> **avoid recursion**

Classical approach not appropriate on GPU

In/On/Out transitions



Ray Tracing CSG Objects Using Single Hit Intersections (A. Kensler) [*]

- Classify A,B intersects, Enter/Exit/Miss
- state(A,B) -> action
- LoopA : tMinA->tA, re-intersectA, re-classifyA (ditto LoopB)

Union, tA < tB	Enter B	Exit B	Miss B
Enter A	ReturnA	LoopA	ReturnA
Exit A	ReturnA	ReturnB	ReturnA
Miss A	ReturnB	ReturnB	ReturnMiss

Union, tB < tA	Enter B	Exit B	Miss B
Enter A	ReturnB	ReturnB	ReturnA
Exit A	LoopB	ReturnA	ReturnA
Miss A	ReturnB	ReturnB	ReturnMiss

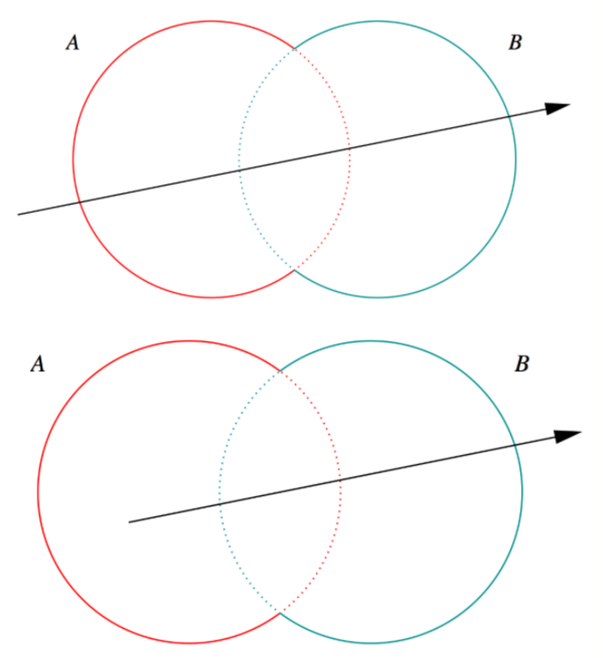
Recursive CSG tree python prototype of Kensler pseudocode worked after state table corrections/extensions

- BUT GPU/OptiX demands: no recursion in intersect program

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

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

CSG Tree, leaf node primitives, internal node operators, 4x4 transforms on any node, serialized as complete binary tree:

- bit twiddling navigation avoids recursion
- no need to deserialize
- no child/parent pointers
- BUT: very inefficient when unbalanced

Bit Twiddling Navigation

- parent(i) = i/2 = i >> 1
- leftchild(i) = 2*i = i << 1
- rightchild(i) = 2*i + 1 = (i << 1) + 1
- leftmost(height) = 1 << height

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

Postorder tree traverse visits all nodes, starting from leftmost, such that children are visited prior to their parents.

Evaluative CSG intersection Pseudocode : recursion emulated

```
fullTree = PACK( 1 << height, 1 >> 1 )  // leftmost, parent_of_root(=0)
tranche.push(fullTree, ray.tmin)

while (!tranche.empty)         // stack of begin/end indices
{
    begin, end, tmin <- tranche.pop  ; node <- begin ;
    while( node != end )                    // over tranche of postorder traversal
    {
        elevation = height - TREE_DEPTH(node) ;
        if(is_primitive(node)){ isect <- intersect_primitive(node, tmin) ;  csg.push(isect) }
        else{
            i_left, i_right = csg.pop, csg.pop            // csg stack of intersect normals, t
            l_state = CLASSIFY(i_left, ray.direction, tmin)
            r_state = CLASSIFY(i_right, ray.direction, tmin)
            action = LUT(operator(node), leftIsCloser)(l_state, r_state)

            if(      action is ReturnLeft/Right)     csg.push(i_left or i_right)
            else if( action is LoopLeft/Right)
            {
                left = 2*node ; right = 2*node + 1 ;
                endTranche = PACK( node,  end );
                leftTranche = PACK(  left << (elevation-1), right << (elevation-1) )
                rightTranche = PACK(  right << (elevation-1),  node  )
                loopTranche = action ? leftTranche : rightTranche

                tranche.push(endTranche, tmin)
                tranche.push(loopTranche, tminAdvanced )  // subtree re-traversal with changed tmin
                break ; // to next tranche
            }
        }
        node <- postorder_next(node, elevation)         // bit twiddling postorder
    }
}
isect = csg.pop();          // winning intersect
```

https://bitbucket.org/simoncblyth/opticks/src/tip/optixrap/cu/csg_intersect_boolean.h

Opticks CSG Primitives : Closed Solids, Consistent Normals

Closed Solid as: implementation requires otherside intersect, **Rigidly attached normals**

Type code	Python name	C++ nnode sub-struct
CSG_BOX3,CSG_BOX	box3,box	nbox
CSG_SPHERE,CSG_ZSPHERE	sphere,zsphere	nsphere,nzsphere
CSG_CYLINDER,CSG_DISC	cylinder,disc	ncylinder,ndisc
CSG_CONE	cone	ncone
CSG_CONVEXPOLYHEDRON	convexpolyhedron	nconvexpolyhedron
CSG_TRAPEZOID,CSG_SEGMENT	trapezoid,segment	nconvexpolyhedron
CSG_TORUS	torus	ntorus
CSG_HYPERBOLOID	hyperboloid	nhyperboloid

- **zsphere, cone, cylinder, disc** : truncated shapes **closed** by endcaps <-- NOT OPTIONAL
- **disc** : avoids endcap degeneracy with very thin cylinders
- **convexpolyhedron** : defined by a set of planes, used for trapezoid and segment
- **segment** : prism shape used for deltaphi intersection
- **!complemented** (inside<->outside) solids handled by special casing classification (cannot miss otherside).

Non-primitives, high level CSG definition avoids loadsa code

- **ellipsoid** : non-uniform scaling of sphere, **polycone** : union of cylinders and cones
- **inner-radii** : via subtraction, **deltaphi-segment** : via intersect with **segment**

Opticks CSG Primitives : What is included

OptiX/CUDA functions providing:

- axis aligned bounding box (AABB)
- intersect ray position (parametric t), surface normal

C++/nnode sub-struct methods

- signed distance function (SDF)
- parametric surface generation

4x4 Transforms on any node (translation/rotation/scaling)

Intersect inverse-transformed ray with un-transformed primitive

- parametric-t same in both frames
- inverse transform transposed brings normal back to world frame

OptiX Geometry

OptiX provides acceleration of geometrical intersection, not the intersection itself.

- parametric ray : **p(t) = p0 + t rayDir**
- implicit surface : **f(p) = 0**

Intersect finding next closest root:

- smallest **t**, with **t > tmin**
- surface normal at **t**

Supporting non-uniform scaling requires **rayDir** not be be normalized (or assumed to be normalized) by primitives.

Opticks CSG : Balancing Deep Trees Drastically Improves Performance

Intended for solids, not scenes (tree height <8, <256 nodes[*])

- unbalanced trees inefficiently handled as complete binary trees
- CSG trees non-unique, many expressions of same shape

```
Dayabay TopESRCutHols lvidx:57  (height:9 totnodes:1023)
di(di(di(di(di(di(di(di(di(cy,cy),cy),cy),cy),cy),cy),cy),cy),cy)

                                                                    di
                                                             di          cy
                                                     di          cy
                                             di          cy
                                     di          cy
                             di          cy
                     di          cy
             di          cy
     di          cy
 cy      cy
```

```
Balanced Tree, height:4 totnodes:31
in(in(in(in(cy,!cy),in(!cy,!cy)),in(in(!cy,!cy),in(!cy,!cy))),!cy)

                                                             in
                                                                !cy
                             in
             in                              in
     in              in              in              in
 cy     !cy     !cy     !cy     !cy     !cy     !cy     !cy
```

[*] Algorithm has no inherent height limit, but use of complete binary tree imposes practical performance limitation

Positive form CSG Trees

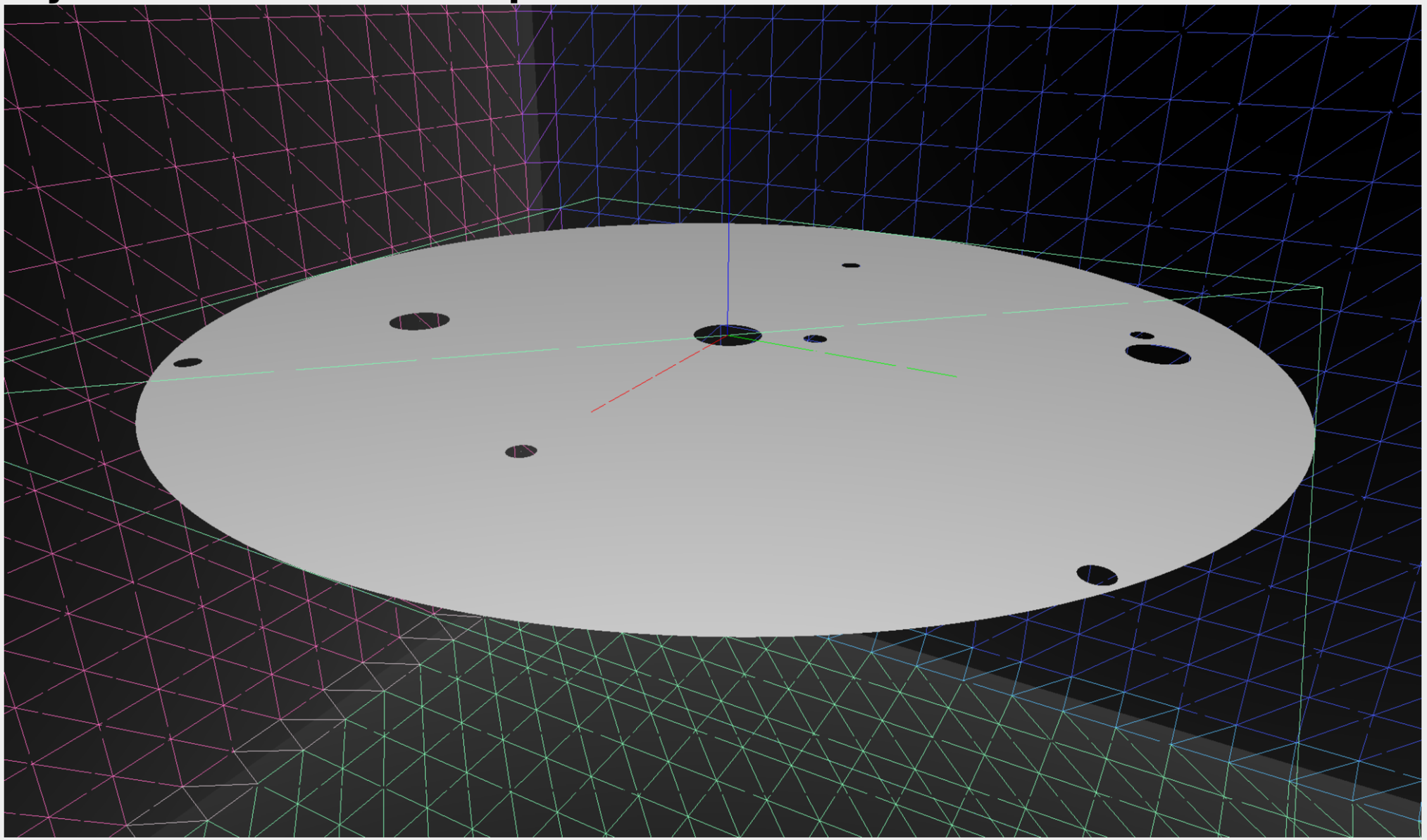
Apply deMorgan pushing negations down tree

- **A - B -> A * !B**
- **!(A*B) -> !A + !B**
- **!(A+B) -> !A * !B**
- **!(A - B) -> !(A*!B) -> !A + B**

End with only UNION, INTERSECT operators, and some complemented leaves.

COMMUTATIVE -> easily rearranged

Dayabay ESR reflector : Deep CSG tree : disc with 9 holes

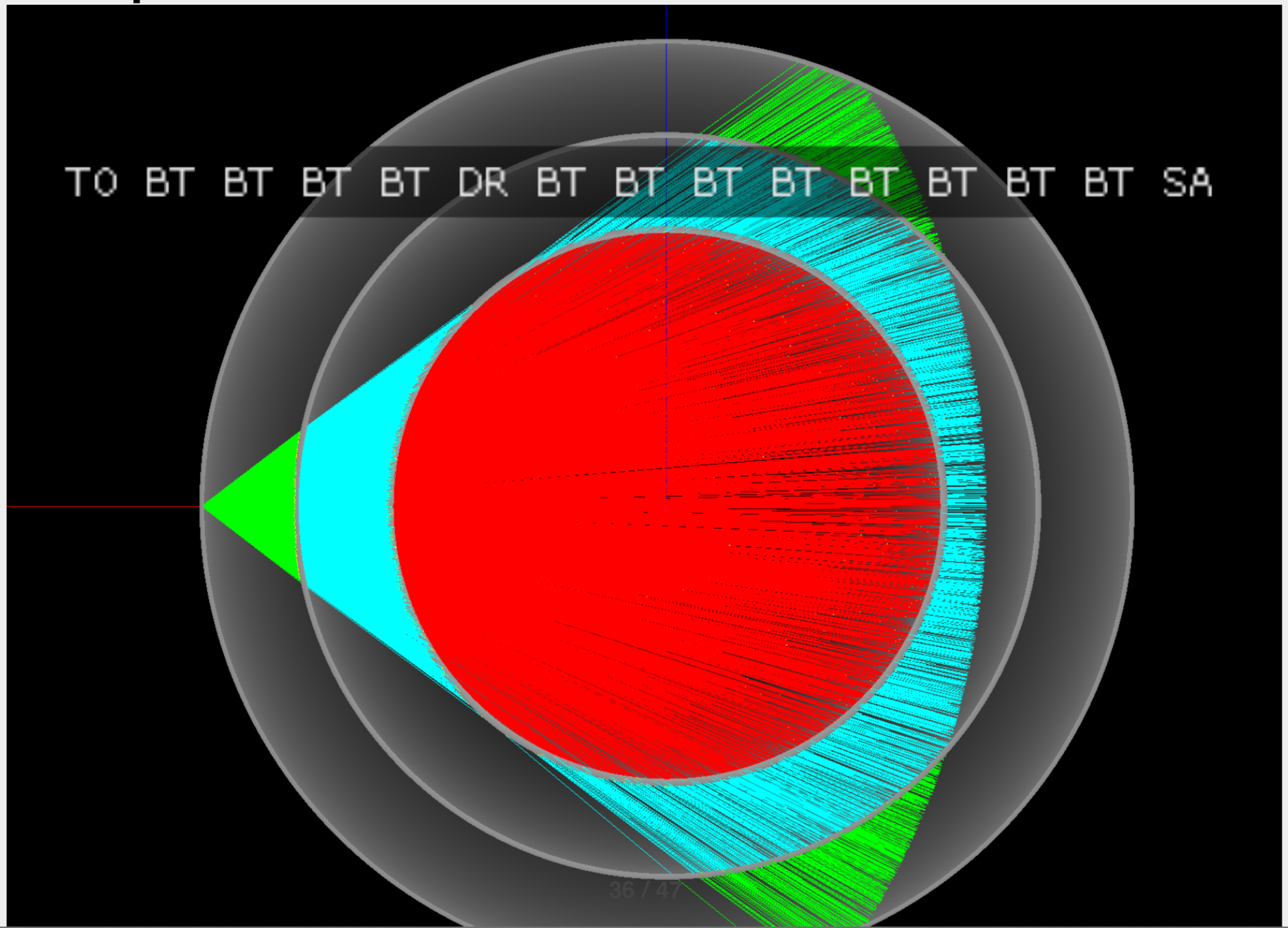


Opticks CSG Serialized into OpticksCSG format (numpy buffers, json)

```
// tboolean-parade

from opticks.ana.base import opticks_main
from opticks.analytic.csg import CSG

args = opticks_main(csgpath="$TMP/$FUNCNAME")

container = CSG("box", param=[0,0,0,1200], boundary=args.container, poly="MC", nx="20" )

a = CSG("sphere", param=[0,0,0,100])
b = CSG("zsphere", param=[0,0,0,100], param1=[-50,60,0,0])
c = CSG("box3",param=[100,50,70,0])
d = CSG.MakeTrapezoid(z=100, x1=80, y1=100, x2=100, y2=80)
e = CSG("cylinder",param=[0,0,0,100], param1=[-100,100,0,0])
f = CSG("disc",param=[0,0,0,100], param1=[-1,1,0,0])
g = CSG("cone", param=[100,-100,50,100])
h = CSG.MakeTorus(R=70, r=30)
i = CSG.MakeHyperboloid(r0=80, zf=100, z1=-100, z2=100)
j = CSG.MakeIcosahedron(scale=100.)

prims = [a,b,c,d,e,f,g,h,i,j]

...  // setting translations

CSG.Serialize([container] + prims, args.csgpath )     <-- write trees to file
```

- imported into C++ **nnode** tree by **NCSG**

tconcentric : spherical GdLS/LS/MineralOil



tconcentric : Opticks/Geant4 chi2 comparison

```
.      seqhis_ana  1:concentric   -1:concentric     c2
.                        1000000       1000000       373.13/356 =  1.05  (pval:0.256 prob:0.744)
0000           8ccccd     669843        670001             0.02  [6 ] TO BT BT BT BT SA
0001              4d       83950         84149             0.24  [2 ] TO AB
0002         8cccc6d       45490         44770             5.74  [7 ] TO SC BT BT BT BT SA
0003          4ccccd       28955         28718             0.97  [6 ] TO BT BT BT BT AB
0004            4ccd       23187         23170             0.01  [4 ] TO BT BT AB
0005         8cccc5d       20238         20140             0.24  [7 ] TO RE BT BT BT BT SA
0006         8cc6ccd       10214         10357             0.99  [7 ] TO BT BT SC BT BT SA
0007         86ccccd       10176         10318             0.98  [7 ] TO BT BT BT BT SC SA
0008         89ccccd        7540          7710             1.90  [7 ] TO BT BT BT BT DR SA
0009        8cccc55d        5976          5934             0.15  [8 ] TO RE RE BT BT BT BT SA
0010            45d         5779          5766             0.01  [3 ] TO RE AB
0011  8ccccccccc9ccccd        5339          5269             0.46  [15] TO BT BT BT BT DR BT BT BT BT BT BT BT
0012         8cc5ccd         5111          4940             2.91  [7 ] TO BT BT RE BT BT SA
0013            46d         4797          4886             0.82  [3 ] TO SC AB
0014      8cccc9ccccd         4494          4469             0.07  [11] TO BT BT BT BT DR BT BT BT BT SA
0015      8cccccc6ccd         3317          3302             0.03  [11] TO BT BT SC BT BT BT BT BT BT SA
0016        8cccc66d          2670          2675             0.00  [8 ] TO SC SC BT BT BT BT SA
0017         49ccccd          2432          2383             0.50  [7 ] TO BT BT BT BT DR AB
0018         4cccc6d          2043          1991             0.67  [7 ] TO SC BT BT BT BT AB
0019           4cc6d          1755          1826             1.41  [5 ] TO SC BT BT AB
```

Top 20 chart above, (category 100 down to ~100 photons for propagation of 1M photons)

tconcentric : Opticks/Geant4 distrib chi2/df ~ 1.0

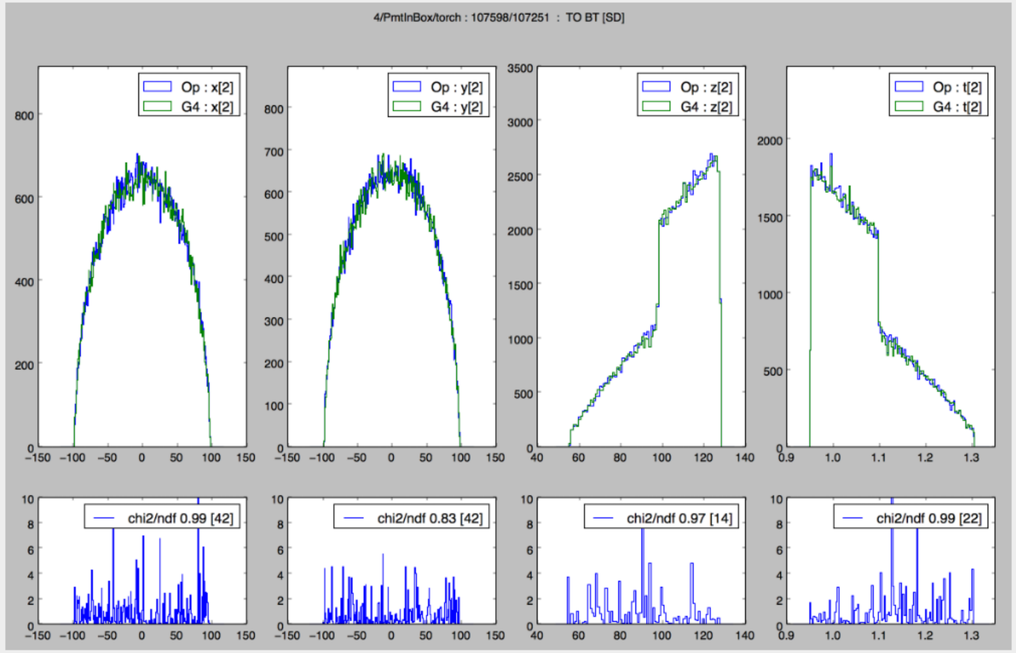
- Top 100 history categories correspond to ~900 propagation points
- 8 quantities at each point : ~7200 histograms pairs to chi2 compare
- selecting discrepant points : distchi2 > 1.1 (yields 41 out of 900 points)

XYZT:position/time ABCW:polarization/wavelength

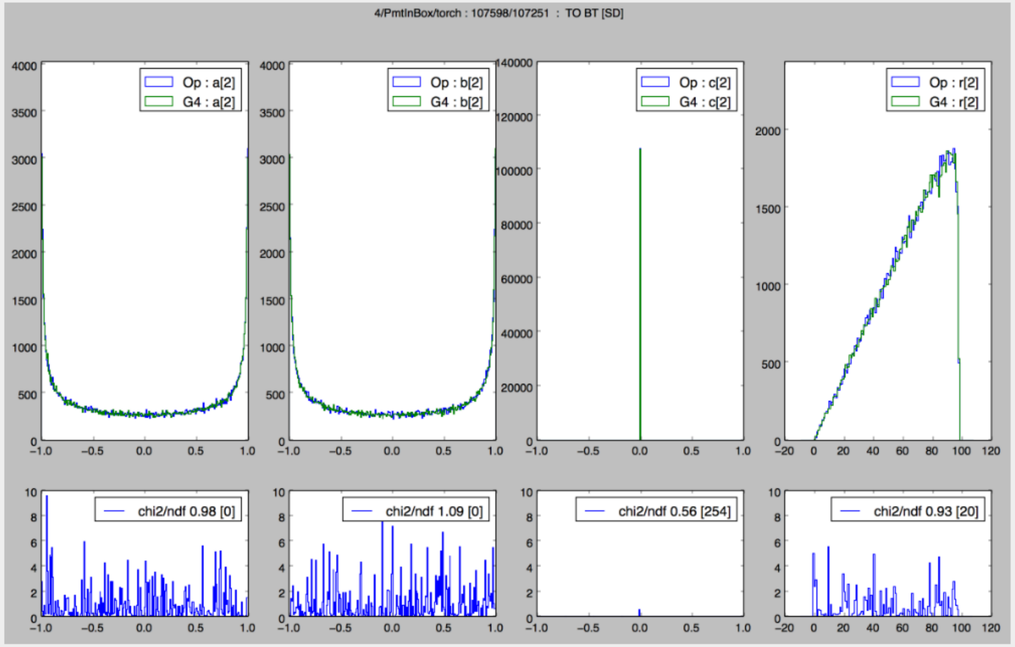
iv	is	na	nb	reclab	X	Y	Z	T	A	B	C	W	seqc2	distc2
26	5	20238	20140	TO [RE] BT BT BT BT SA	0.85	0.00	0.00	1.31	1.12	1.37	1.10	0.78	0.24	1.10
27	5	20238	20140	TO RE [BT] BT BT BT SA	2.14	2.26	0.80	1.08	1.15	0.82	0.76	0.78	0.24	1.18
28	5	20238	20140	TO RE BT [BT] BT BT SA	2.01	2.23	0.79	0.83	1.17	0.83	0.83	0.78	0.24	1.17
29	5	20238	20140	TO RE BT BT [BT] BT SA	2.66	4.37	1.13	0.49	1.20	0.81	0.79	0.78	0.24	1.68
30	5	20238	20140	TO RE BT BT BT [BT] SA	2.56	4.48	1.19	1.04	1.12	0.97	0.91	0.78	0.24	1.75
31	5	20238	20140	TO RE BT BT BT BT [SA]	3.18	5.17	1.23	0.48	1.12	0.97	0.91	0.78	0.24	2.06
38	6	10214	10357	TO BT BT SC BT BT [SA]	0.79	1.37	1.43	0.55	1.00	1.33	0.97	0.00	0.99	1.16
52	8	7540	7710	TO BT BT BT BT DR [SA]	1.70	1.32	1.48	1.49	1.12	1.03	1.37	0.00	1.90	1.28
56	9	5976	5934	TO RE RE [BT] BT BT BT SA	1.26	1.51	1.21	2.36	0.99	1.40	1.10	1.65	0.15	1.24
57	9	5976	5934	TO RE RE BT [BT] BT BT SA	1.23	1.39	1.25	2.31	0.98	1.45	0.98	1.65	0.15	1.21
58	9	5976	5934	TO RE RE BT BT [BT] BT SA	1.24	0.98	1.18	1.88	0.97	1.39	1.01	1.65	0.15	1.14
59	9	5976	5934	TO RE RE BT BT BT [BT] SA	1.24	0.90	1.04	1.83	0.93	1.55	0.92	1.65	0.15	1.11
60	9	5976	5934	TO RE RE BT BT BT BT [SA]	0.95	1.03	1.50	3.12	0.93	1.55	0.92	1.65	0.15	1.18
69	11	5339	5269	TO BT BT BT BT [DR] BT BT BT BT BT BT BT BT SA	0.00	0.00	0.00	0.00	1.29	1.69	2.42	0.00	0.46	1.31
74	11	5339	5269	TO BT BT BT BT DR BT BT BT BT [BT] BT BT BT SA	1.10	1.45	1.02	0.67	1.42	0.83	1.38	0.00	0.46	1.12
75	11	5339	5269	TO BT BT BT BT DR BT BT BT BT BT [BT] BT BT SA	0.98	1.42	1.16	0.52	1.58	0.82	1.46	0.00	0.46	1.15
76	11	5339	5269	TO BT BT BT BT DR BT BT BT BT BT BT [BT] BT SA	1.46	1.66	0.79	0.65	1.69	0.89	1.46	0.00	0.46	1.21
77	11	5339	5269	TO BT BT BT BT DR BT BT BT BT BT BT BT [BT] SA	1.04	1.64	0.81	0.51	2.20	0.91	1.35	0.00	0.46	1.19
78	11	5339	5269	TO BT BT BT BT DR BT BT BT BT BT BT BT BT [SA]	1.10	1.56	0.73	0.21	2.20	0.91	1.35	0.00	0.46	1.17
85	12	5111	4940	TO BT BT RE BT BT [SA]	1.26	2.13	0.79	2.07	1.03	0.93	0.72	0.68	2.91	1.11
94	14	4494	4469	TO BT BT BT BT [DR] BT BT BT BT SA	0.00	0.00	0.00	0.00	1.90	3.74	1.95	0.00	0.07	2.01

PMT Opticks/Geant4 step distribution comparison TO BT [SD]

Good agreement reached, after several fixes: geometry, total internal reflection, group velocity



position(xyz), time(t)



polarization(abc), radius(r)

PMT Opticks/Geant4 step distribution comparison : chi2/ndf

4/PMT In Box/torch :	X	Y	Z	T	A	B	C	R
340271/340273 : [TO] BT SA	1.15	1.00	0.00	0.00	1.06	1.03	0.00	1.21
340271/340273 : TO [BT] SA	1.15	1.00	1.06	0.91	1.06	1.03	0.00	1.21
340271/340273 : TO BT [SA]	0.97	1.02	1.05	0.99	1.06	1.03	0.00	1.29
107598/107251 : [TO] BT SD	0.91	0.73	0.56	0.56	0.98	1.09	0.56	0.94
107598/107251 : TO [BT] SD	0.91	0.73	0.81	0.93	0.98	1.09	0.56	0.94
107598/107251 : TO BT [SD]	0.99	0.83	0.97	0.99	0.98	1.09	0.56	0.93
23217/23260 : [TO] BT BT SA	0.94	0.82	0.04	0.04	0.97	0.89	0.04	0.57
23217/23260 : TO [BT] BT SA	0.94	0.82	0.70	0.50	0.97	0.89	0.04	0.57
23217/23260 : TO BT [BT] SA	0.91	0.94	0.43	0.60	0.97	0.89	0.04	0.05
23217/23260 : TO BT BT [SA]	0.94	0.88	0.04	0.35	0.97	0.89	0.04	0.72
18866/19048 : [TO] AB	0.99	1.10	0.87	0.87	0.85	0.84	0.87	1.00
18866/19048 : TO [AB]	0.99	1.10	0.93	0.92	0.85	0.84	0.87	1.00
3179/3133 : [TO] SC SA	1.07	0.83	0.34	0.34	0.86	0.96	0.34	0.73
3179/3133 : TO [SC] SA	1.07	0.83	0.98	1.05	0.98	1.06	0.98	0.73
3179/3133 : TO SC [SA]	0.96	1.04	0.93	0.97	0.98	1.06	0.98	1.10
2204/2249 : [TO] BT AB	0.85	1.04	0.45	0.45	0.99	0.92	0.45	1.06
2204/2249 : TO [BT] AB	0.85	1.04	0.95	0.88	0.99	0.92	0.45	1.06
2204/2249 : TO BT [AB]	0.98	0.94	1.01	1.00	0.99	0.92	0.45	0.90
1696/1732 : [TO] BT BT AB	1.05	0.85	0.38	0.38	0.86	1.09	0.38	0.26
1696/1732 : TO [BT] BT AB	1.05	0.85	1.48	1.28	0.86	1.09	0.38	0.26
1696/1732 : TO BT [BT] AB	0.99	0.86	1.17	1.40	0.86	1.09	0.38	0.86
1696/1732 : TO BT BT [AB]	1.15	0.88	1.08	1.06	0.86	1.09	0.38	0.79
1446/1455 : [TO] BR SA	1.21	0.94	0.03	0.03	0.90	0.87	0.03	1.09
1446/1455 : TO [BR] SA	1.21	0.94	1.02	1.01	0.90	0.87	0.03	1.09
1446/1455 : TO BR [SA]	1.00	0.93	0.97	0.99	0.90	0.87	0.03	1.04

Consistent : chi2/ndf ~ 1

Very good Opticks/Geant4 agreement

- identical geometries
- identical optical physics

XYTZ: position, time ABCR: polarization, radius

Opticks/Geant4 Rainbow Step Sequence Comparison

Flags:

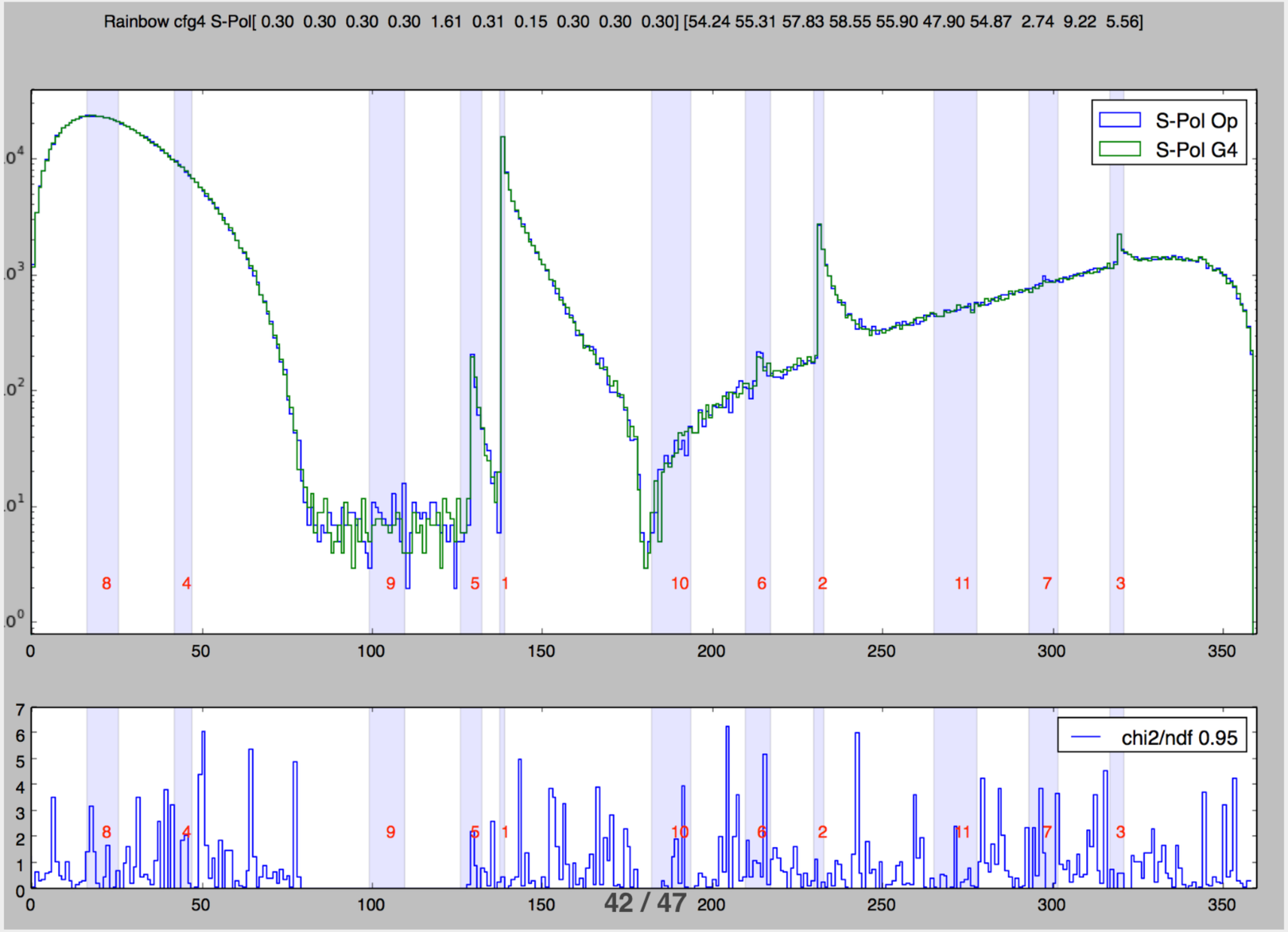
- BT/BR: boundary transmit/reflect
- TO/SC/SA: torch/scatter/surface absorb

Statistically consistent photon histories in the two simulations : Multiple orders of rainbow apparent

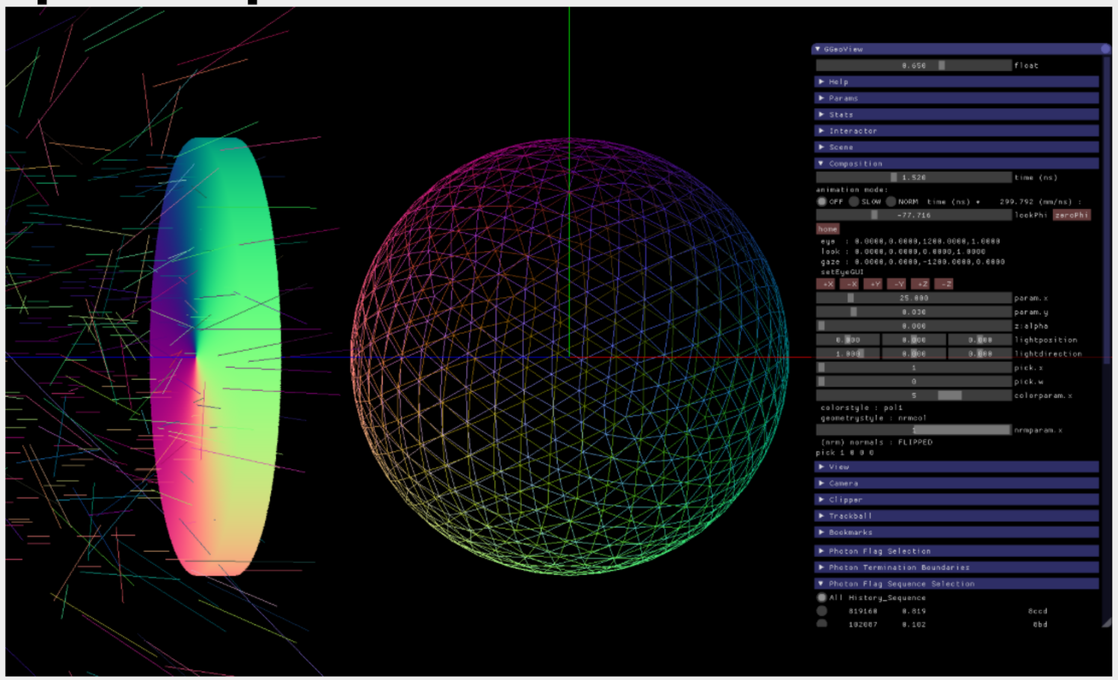
64-bit uint	Opticks	Geant4	chi2		(tag:5,-5)
8ccd	819160	819654	0.15	[4] TO BT BT SA	(cross droplet)
8bd	102087	101615	1.09	[3] TO BR SA	(external reflect)
8cbcd	61869	61890	0.00	[5] TO BT BR BT SA	(bow 1)
8cbbcd	9618	9577	0.09	[6] TO BT BR BR BT SA	(bow 2)
8cbbbcd	2604	2687	1.30	[7] TO BT BR BR BR BT SA	(bow 3)
8cbbbbcd	1056	1030	0.32	[8] TO BT BR BR BR BR BT SA	(bow 4)
86ccd	1014	1000	0.10	[5] TO BT BT SC SA	
8cbbbbbcd	472	516	1.96	[9] TO BT BR BR BR BR BR BT SA	(bow 5)
86d	498	473	0.64	[3] TO SC SA	
bbbbbbbbcd	304	294	0.17	[10] TO BT BR BR BR BR BR BR BR BR	(bow 8+ truncated)
8cbbbbbbcd	272	247	1.20	[10] TO BT BR BR BR BR BR BR BT SA	(bow 6)
cbbbbbbbcd	183	161	1.41	[10] TO BT BR BR BR BR BR BR BR BT	(bow 7 truncated)

1M Rainbow S-Polarized, Comparison Opticks/Geant4

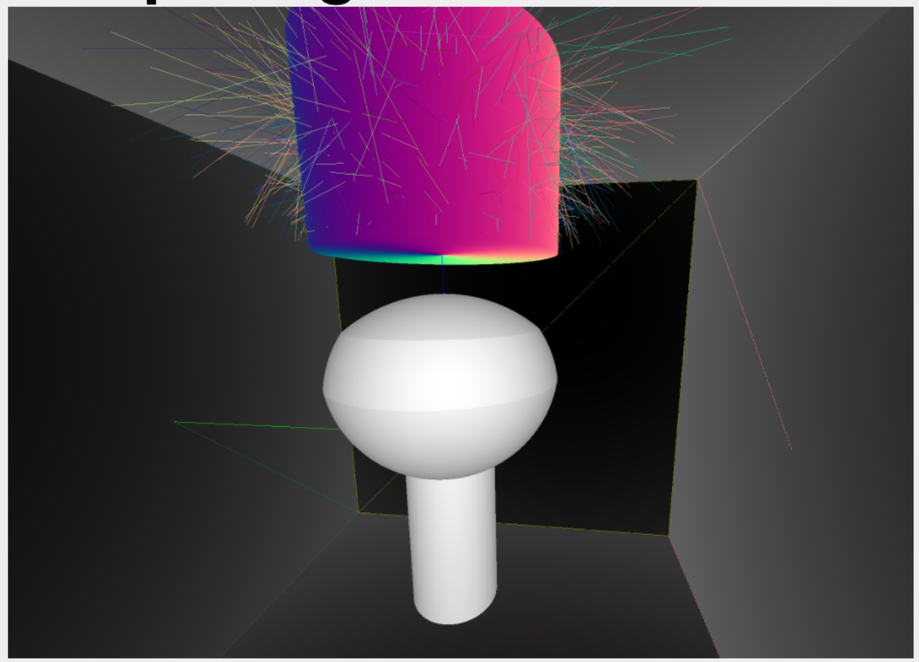
Deviation angle(degrees) of 1M parallel monochromatic photons in disc shaped beam incident on water sphere. Numbered bands are visible range expectations of first 11 rainbows. S-Polarized intersection (E field perpendicular to plane of incidence) arranged by directing polarization radially.



Compare Opticks/Geant4 Simulations with Simple Lights/Geometries



1M Photons -> Water Sphere (S-Polarized)



0.5M Photons -> Dayabay PMT

Photon step records

128 bit per step : highly compressed position, time, wavelength, polarization vector, material/history codes

Photon flag sequence

16x 4-bit step flags recorded in uint64 sequence, indexed using Thrust GPU sort (1M indexed ~0.040s)

Sequence index -> interactive OpenGL selection of photons by flag sequence

OptiX Performance Scaling with GPU cores

OptiX sample rendering with 2 GPU IHEP workstation,

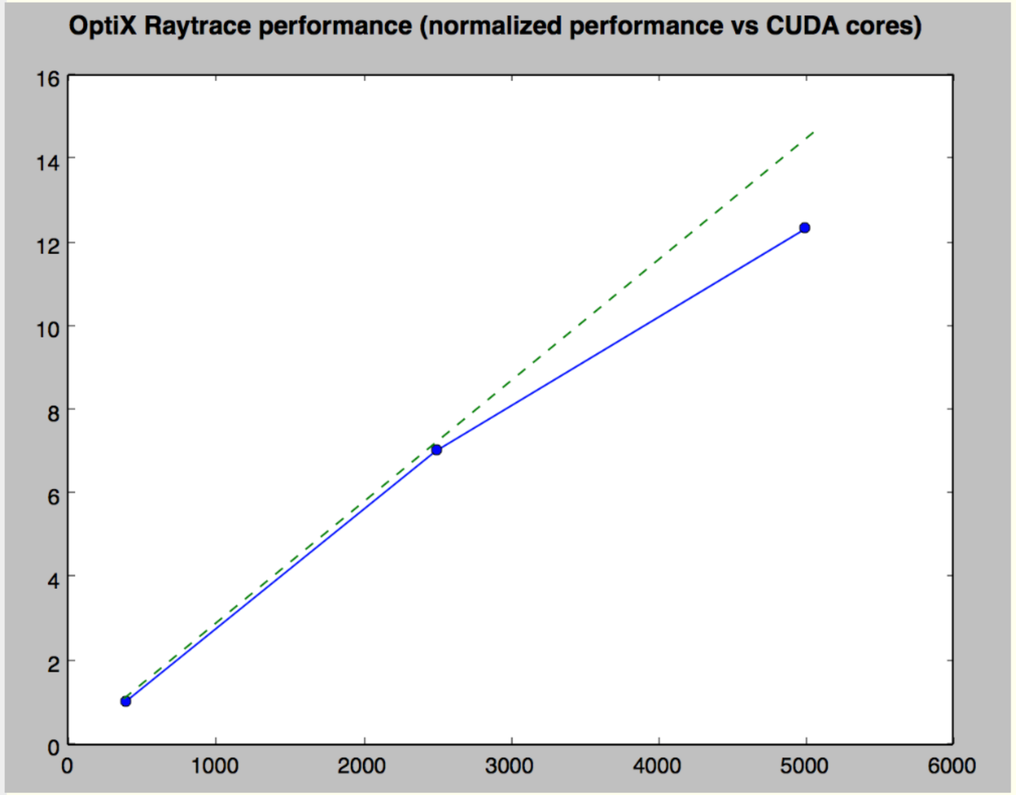
- 2 Tesla K20m (4992 cores) 28.0 ms/f
- 1 Tesla K20m (2496 cores) 49.1 ms/f
- 1 GeForce GT 750m (384 cores) 345.1 ms/f

Performance linear with GPU cores, compared to laptop:

- 13x cores, 12x performance
- performance scales across GPUs

Benefit from multiple GPUs with no development effort.

Performance Linearity with CUDA cores



Torus : much more difficult/expensive than other primitives

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

- ray-torus intersection -> solve quartic polynomial in **t**
- A t^4 + B t^3 + C t^2 + D t + E = 0

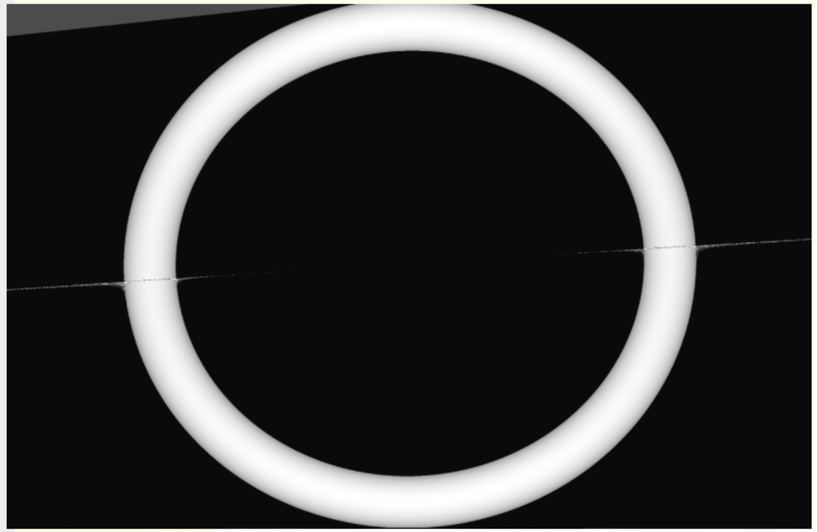
High order equation

- very large difference between coefficients
- varying ray -> wide range of very coefficients
- numerically problematic, requires double precision
- several mathematical approaches used, work in progress

Best Solution : replace torus

- eg model PMT neck with hyperboloid, not cylinder-torus

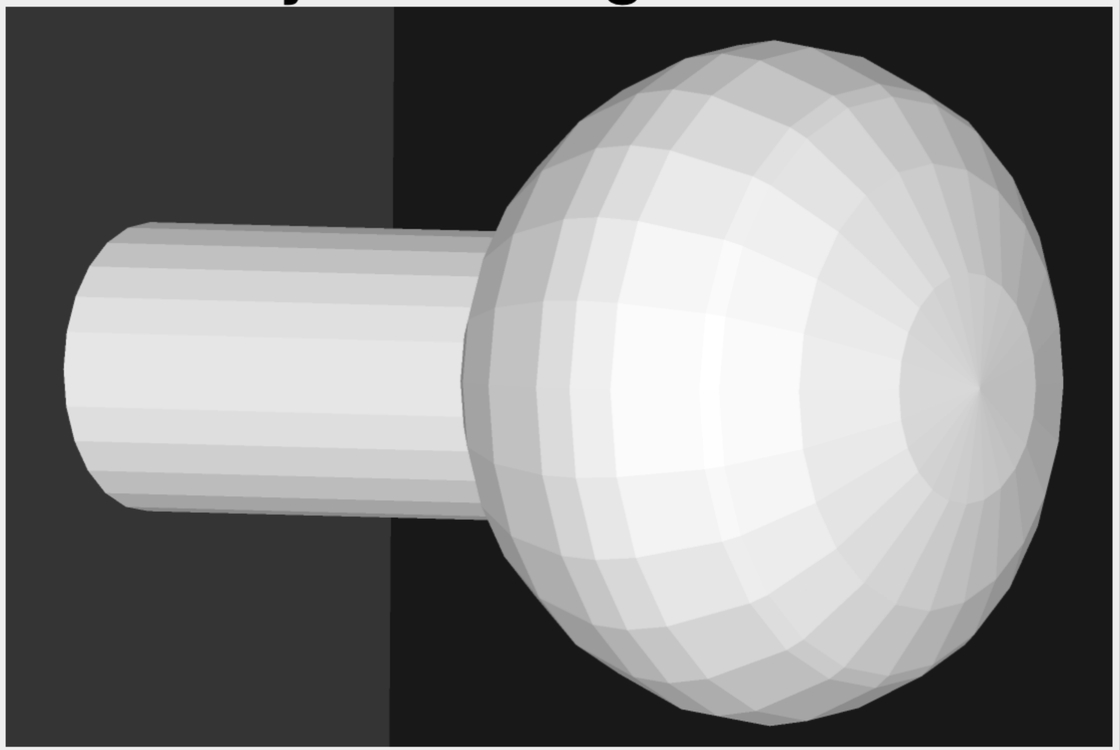
Torus artifacts

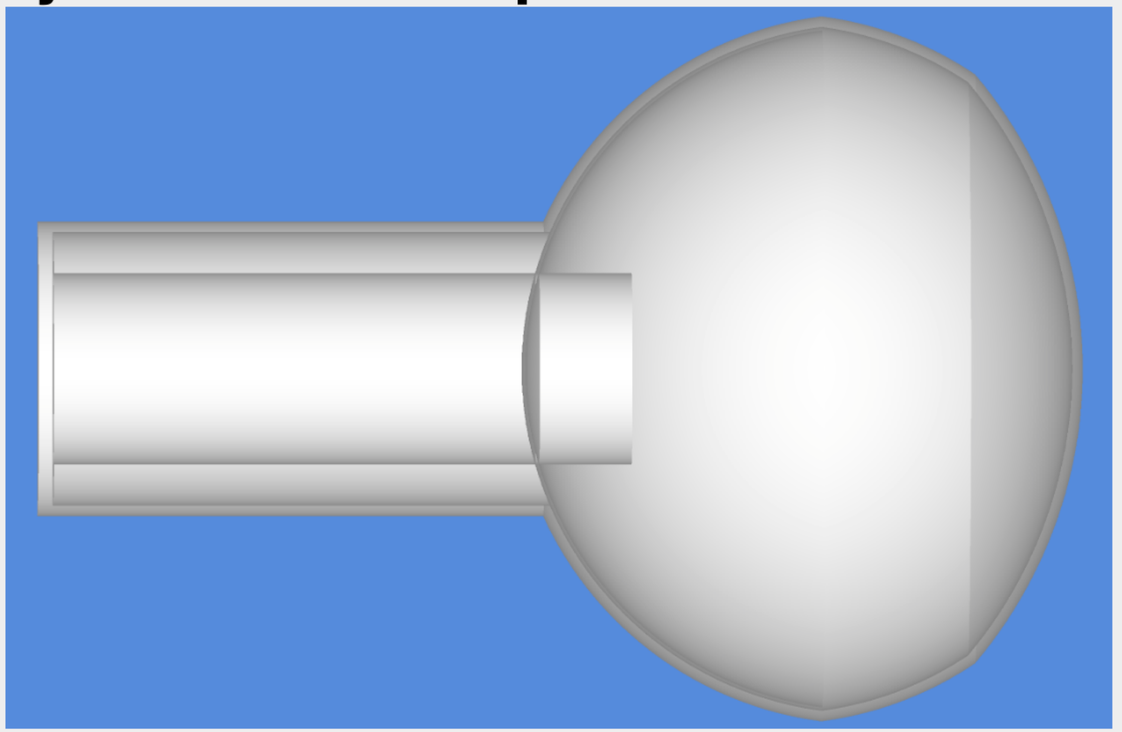


$$\left(R - \sqrt{x^2 + y^2}\right)^2 + z^2 = r^2,$$

$$\left(x^2 + y^2 + z^2 + R^2 - r^2\right)^2 = 4R^2\left(x^2 + y^2\right)$$

Geometry Modelling : Tesselated vs Analytic Photomultiplier Tubes





Analytic : more realistic, faster, less memory, **much more effort**

For Dayabay PMT:

- partition CSG solids into 12 **single primitive** parts (instead of 2928 triangles)
- splitting at geometrical intersections avoids implementing general CSG boolean handling
- geometry provided to OptiX in form of ray intersection and bounding box code

Aim : analytic description of geometry on critical optical path, remainder tesselated

Opticks : GPU Accelerated Optical Photon Simulation us

Version: 0.0.1

Date: August 16, 2016

The Opticks repository http://bitbucket.org/simoncblyth/opticks contains the sources and building externals and Opticks itself. Instructions for using these scripts can be fou

Contents:

Open Source Opticks

- http://simoncblyth.bitbucket.io/opticks/
- http://bitbucket.org/simoncblyth/opticks/

Documentation, install instructions. Repository.

- Mac, Linux, Windows (*)
- 16 C++ projects, ordered by dependency
- ~200 "Unit" Tests (CMake/CTest)
- 12 integration tests: tpmt, trainbow, tprism, treflect, tlens, tnewton, tg4gun, ...
- NumPy/Python analysis/debugging scripts

Geometry/event data use NumPy serialization:

```
import numpy as np
a = np.load("photons.npy")
```

(*) Windows VS2015, non-CUDA only so far