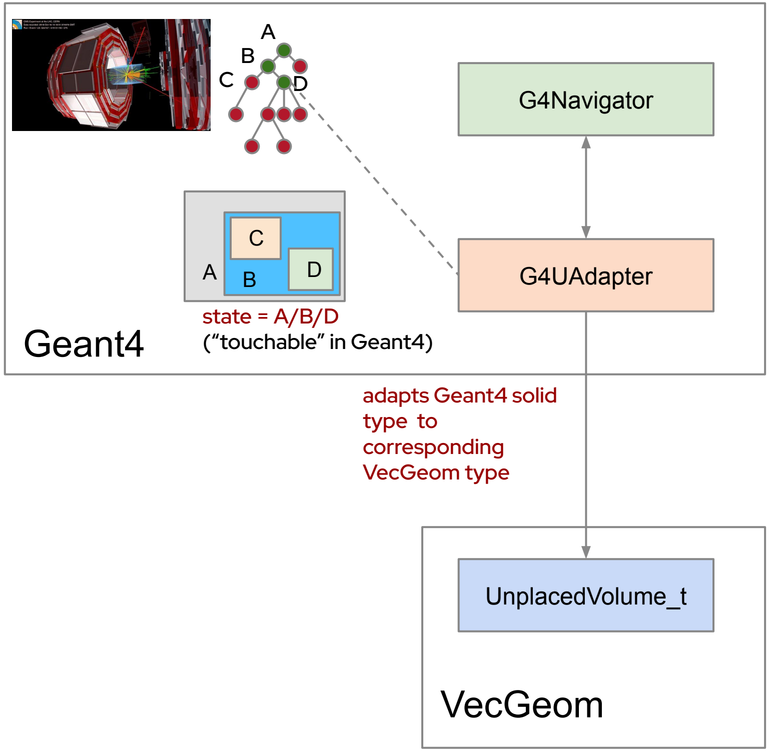
Surface-based GPU-friendly geometry modeling for detector simulation

Andrei Gheata (CERN), for the VecGeom & AdePT projects

26th INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY & NUCLEAR PHYSICS (CHEP2023) - Norfolk, May 8-12, 2023

VecGeom: navigation back-end for Geant4

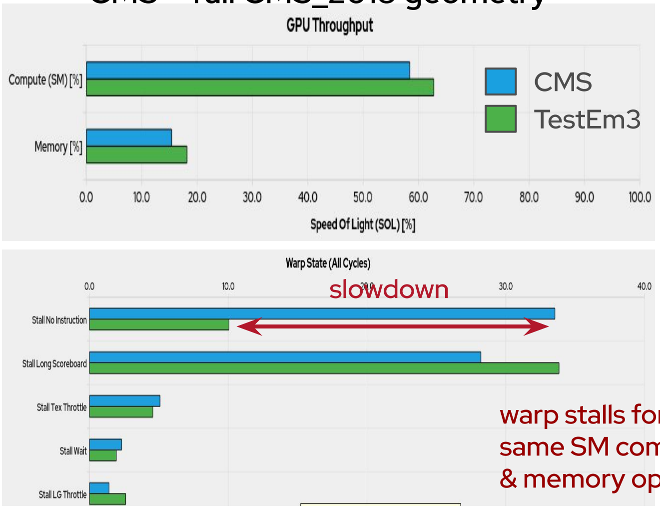
- VecGeom: efficient navigation algorithms behind particle transport simulation
  - multi-purpose 3D constructive solid modeller
  - Independent of the transport simulation toolkit
  - Targeting single and multi-particle workflows on multiple CPU back-ends (scalar and SIMD) but provides also GPU support
- Library evolution
  - Production: committed long-term CPU scalar support
  - GPU simulation projects: AdePT and Celeritas
    - Actively improving the GPU support



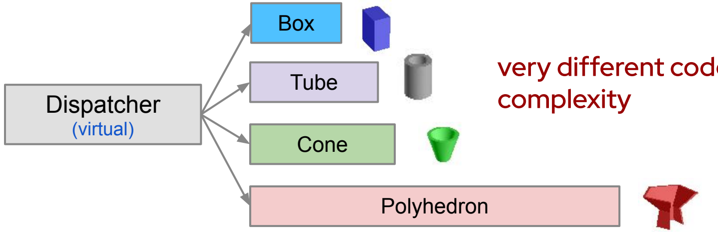
Current solid modeling on GPU

- Several GPU unfriendly features
  - Virtual dispatch
  - Recursive code (relocation)
  - (Very) different algorithm complexity
- AdePT project: geometry complexity worsens performance → main bottleneck
  - Longer stalls within warps for the same SM compute - divergence limiting warp-level concurrency
  - Large stacks & register-hungry code limiting the number of warps running concurrently w/o dumping registers to memory

TestEM3 = sampling calorimeter 50 layers
CMS = full CMS_2018 geometry



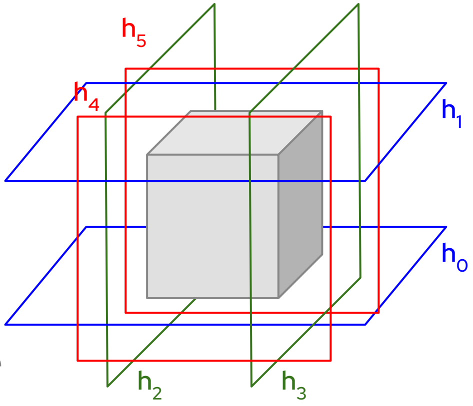
warp stalls for the same SM compute & memory ops

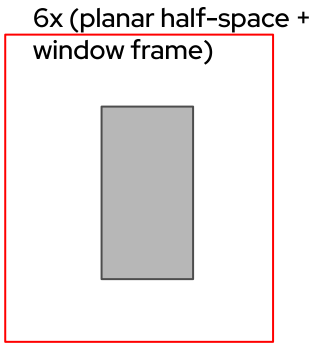


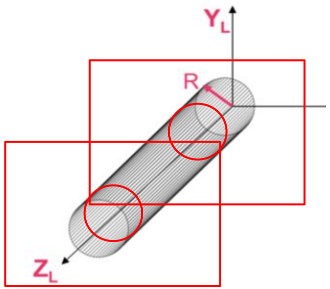
very different code complexity

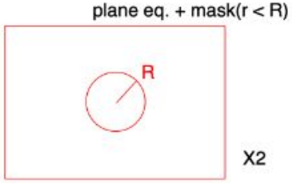
Bounded surface modeling – a different model friendly to GPUs

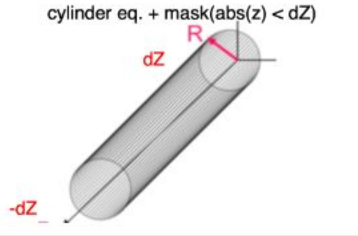
- 3D bodies represented as Boolean operation of half-spaces*
  - First and second order, infinite
  - Just intersections for convex primitives
    - e.g. box = h_0 & h_1 & h_2 & h_3 & h_4 & h_5
  - Similarities with the Orange model
    - Bootstrapped by collaborating with the Orange team
- Storing in addition the solid imprint (frame) on each surface: FramedSurface
  - Similarities with detray (ACTS)
  - Frame information is redundant
    - allows for more efficient navigation overall, using pre-computed information





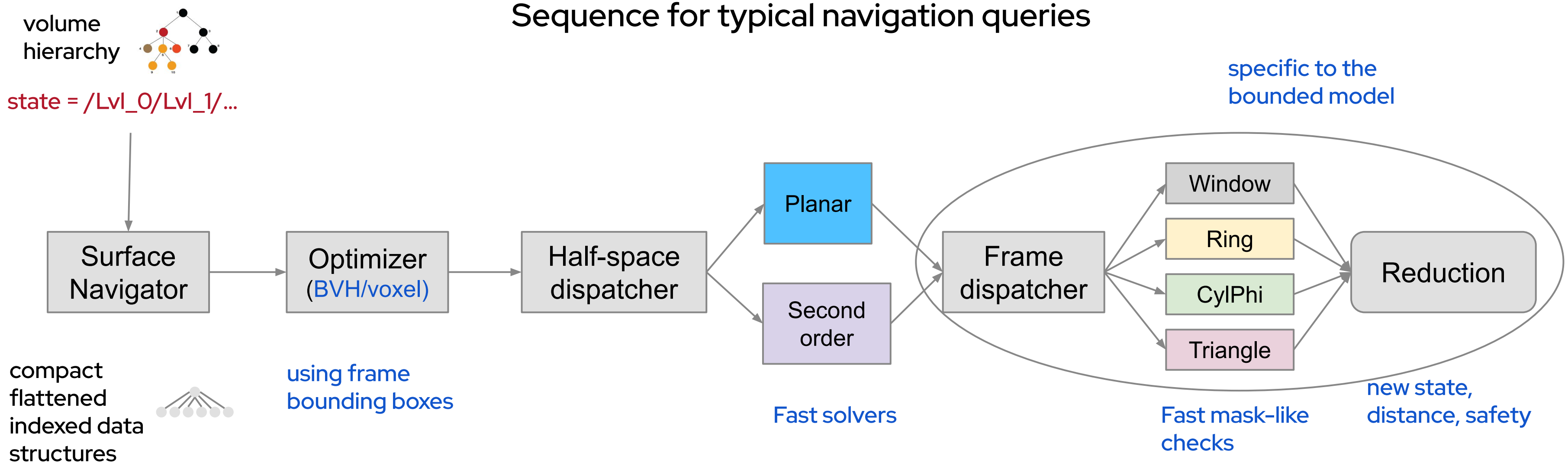






Motivation for surface modeling

Sequence for typical navigation queries



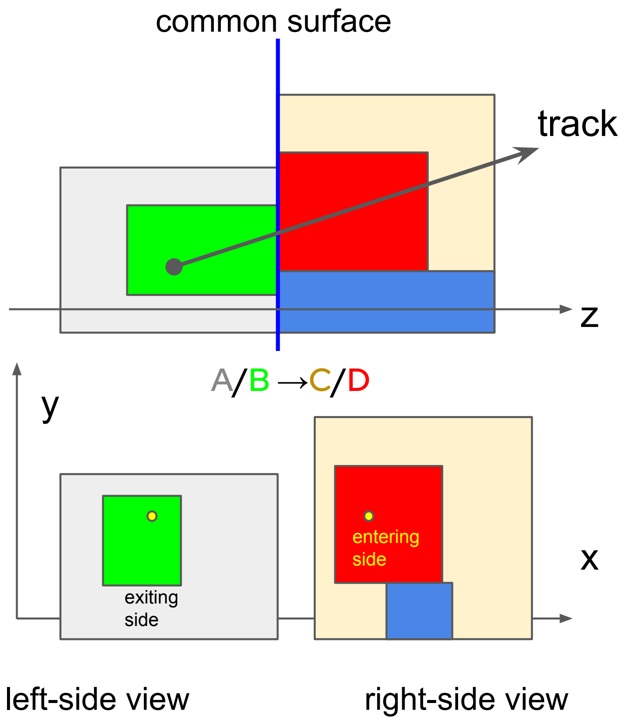
More effective optimizations due to framed surface bounding boxes

Portable code with non-virtual dispatching and non-recursive algorithms

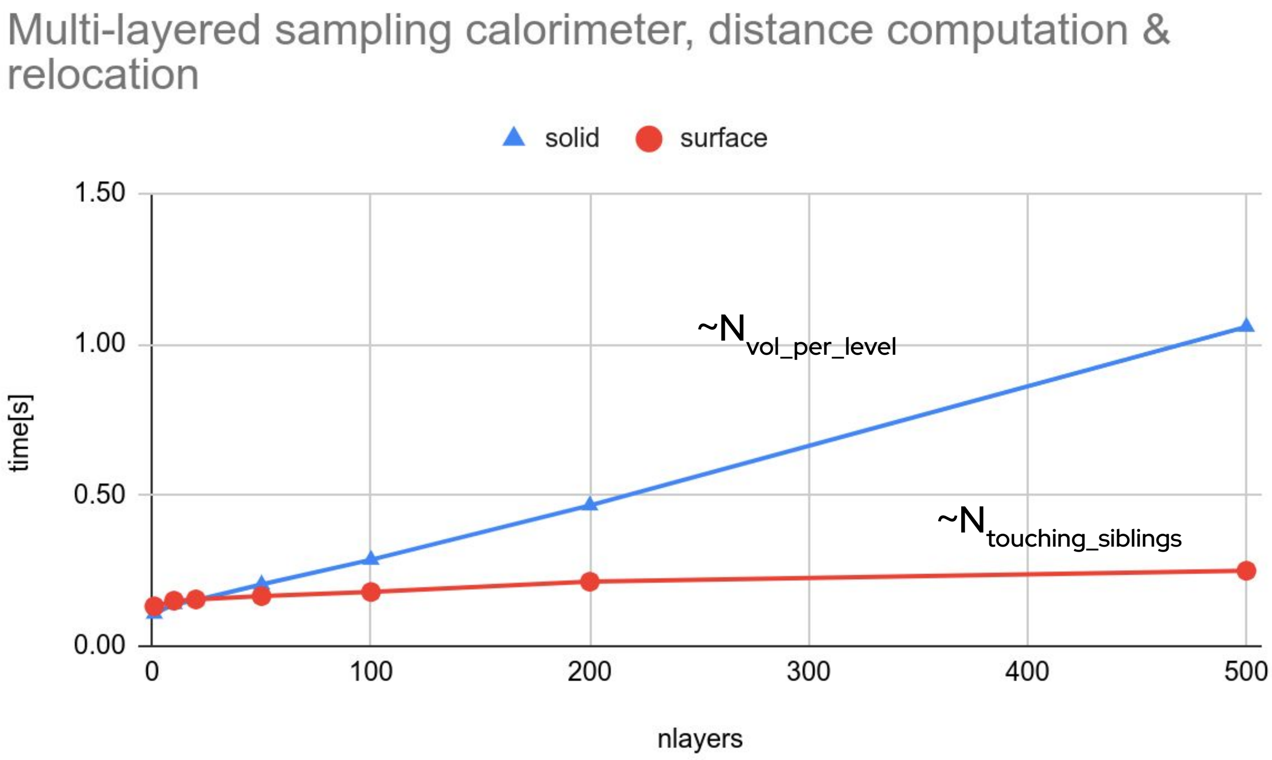
Simpler code with faster divergent sections

More efficient particle relocation

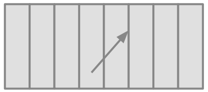
- In Geant volumes can share common surfaces
  - Define "***common surfaces***" as transition boundaries between volumes, pre-computed and deduplicated
  - Volumes contribute with frame imprints on each side
- Locating the particle crossing point on the frames on each side defines a relocation procedure
  - More efficient linear search, involving only a limited set of neighbors and not all daughters of a volume



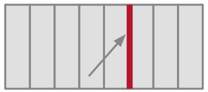
Relocation performance



solids: loop over objects at the same level

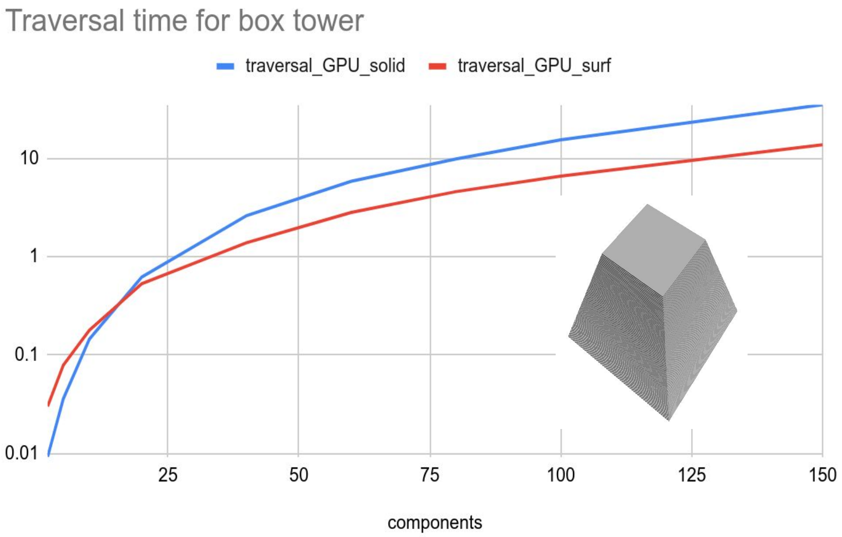


loop over touching framed surfaces



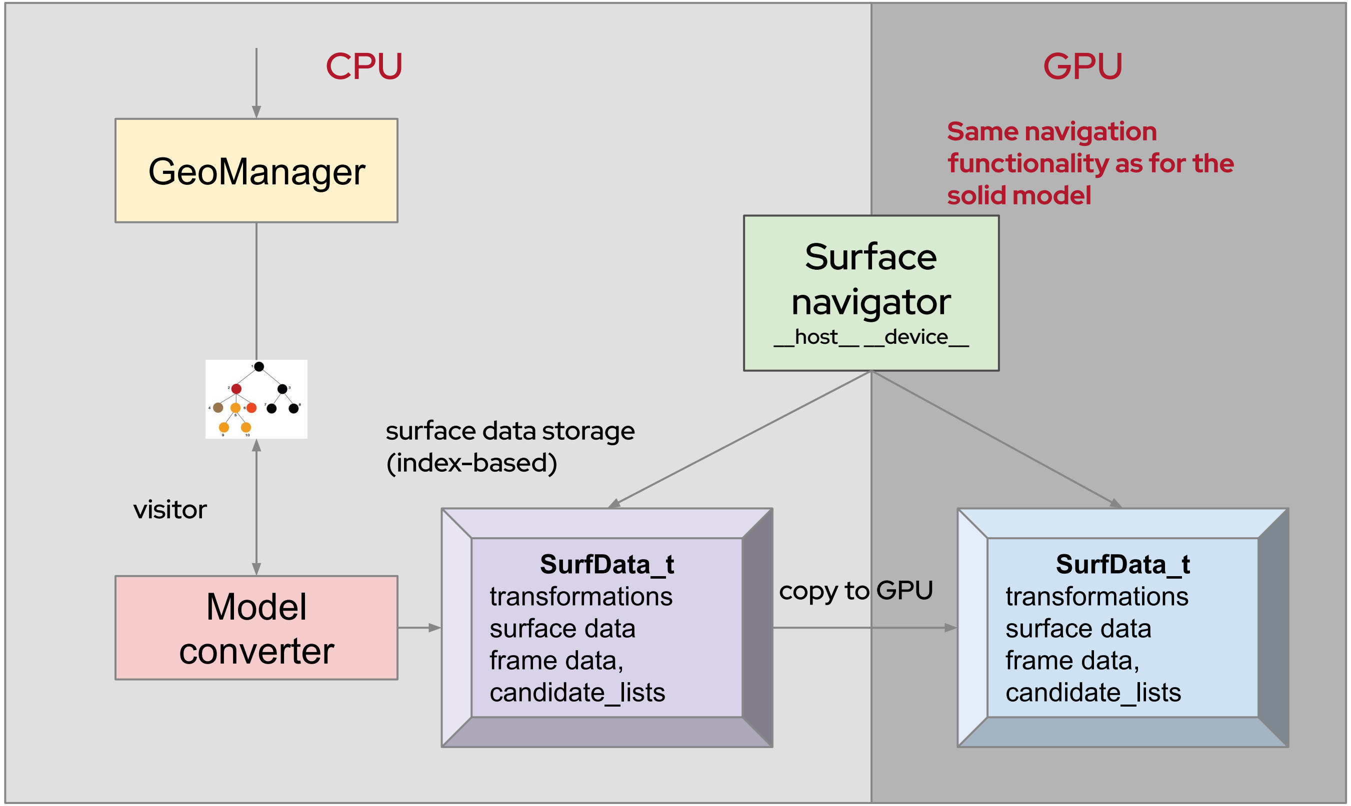
Scaling for the Boolean implementation

- Current implementation validated for correctness against the VecGeom solid model
  - Infix logic expression evaluation
  - Tested union of up to 150 layers of disks subtracting a box, more exhausts CUDA stack space for the solid approach
- Un-optimized version so far, but scaling looks good
  - 2x slower for 5 components, **2x faster for 50 components on GPU**
  - more details in the backup

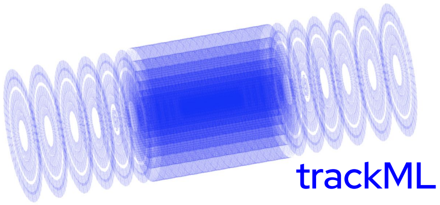


Ray-tracing example traversing all volume boundaries until exiting the setup

Header library with transparent usage



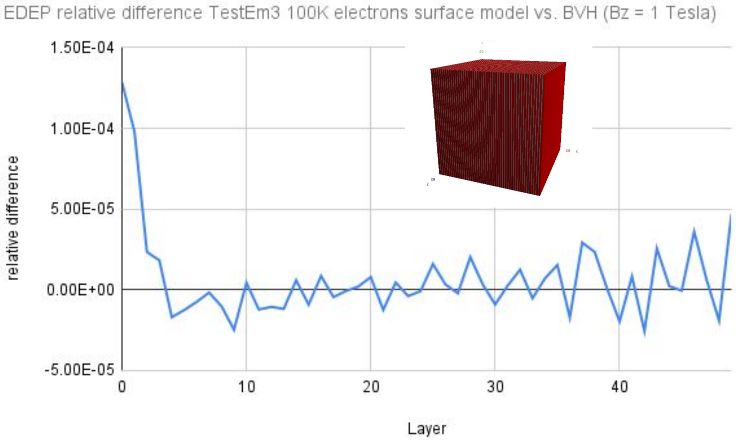
Preliminary performance



- Unit tests available for correctness checking against VecGeom solid model
  - Box, tube, trapezoid, polyhedron, Boolean solids
  - TestEm3 - a simple layered calorimeter made of box slabs
- Ray-tracing benchmark, working with generic GDML input (supported solids only)
  - Testing full navigation functionality on CPU and GPU
  - Validated & benchmarked against existing VecGeom solid navigators
- Results (compared to solid looping navigation) for trackML setup
  - Safety computation: ~2x slower on CPU, **~2x faster on GPU**
  - Propagation + relocation: ~2x faster on CPU, **~6x faster on GPU**
  - Memory: ~1 kByte per “touchable” volume

Integration in AdePT GPU prototype

- Optional usage of the surface model in AdePT example
  - No relevant changes needed other than triggering the model conversion and the navigator type
- Sampling calorimeter simulation
  - block of Pb + LAr box layers (w/ constant Bz field)
  - 10 GeV electrons shot towards the calorimeter along X axis
- Numerical divergence small and understood



	BVH	Loop	surf
no field	152s	162s	156s
Bz=1T	194s	bug	184s

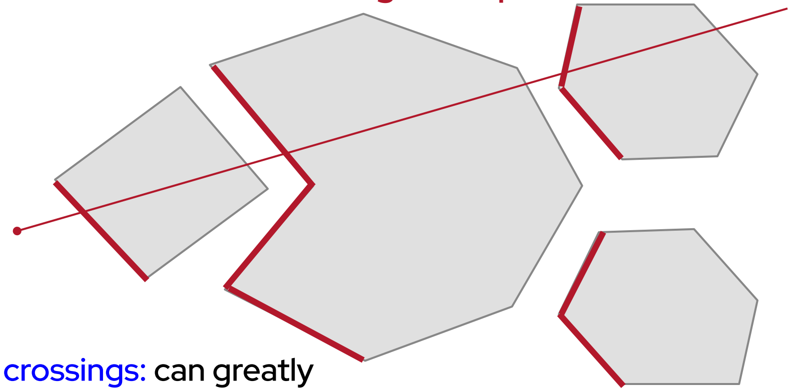
Outlook

- As GPU simulation gains in weight and geometry is on critical path, VecGeom develops dedicated surface-based GPU support
  - Surface model enriched with solid frame information
  - Collaboration with Celeritas/ORANGE team on commonalities and convergence paths
  - Transparent implementation, better work-balanced and friendlier to GPU
- Currently implemented all the features required by particle transport, for a subset of solids
  - Integrated with AdePT, already usable with very simple setups
  - Promising preliminary numbers
- Coverage and optimizations are essential for testing realistic setups
  - Having the full set of supported 3D solids
  - Working on alternatives to reduce the memory footprint
  - Implementing BVH acceleration structures

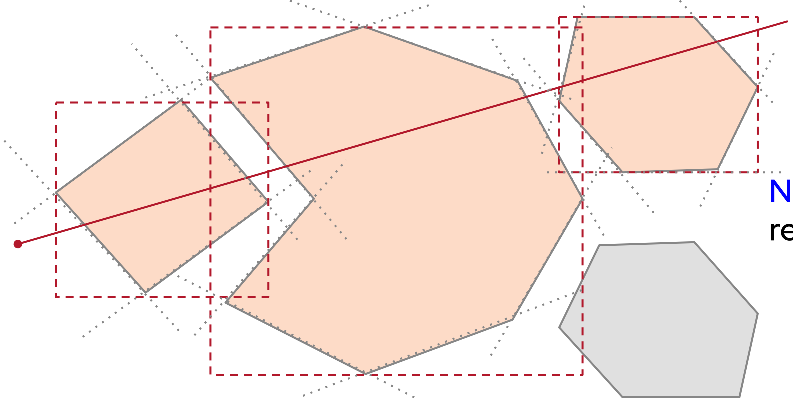
Backup

Why use frames ?

bounded surfaces, normal-optimized,
no bounding box optimization

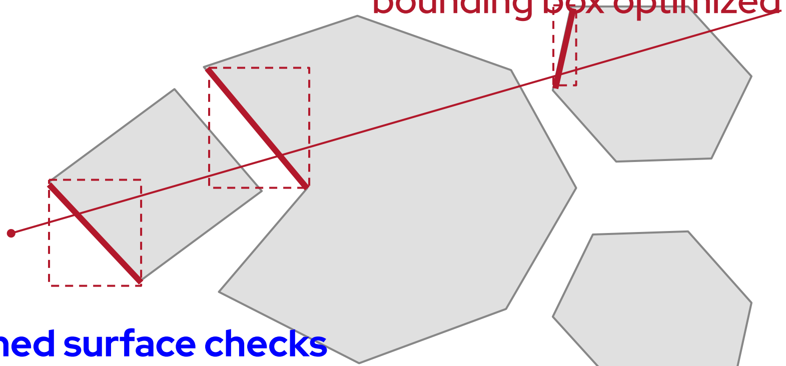


volumes/unbounded
surfaces, bbox optimized



No virtual crossings: can greatly
reduce candidates to be checked

bounded surfaces,
bounding box optimized



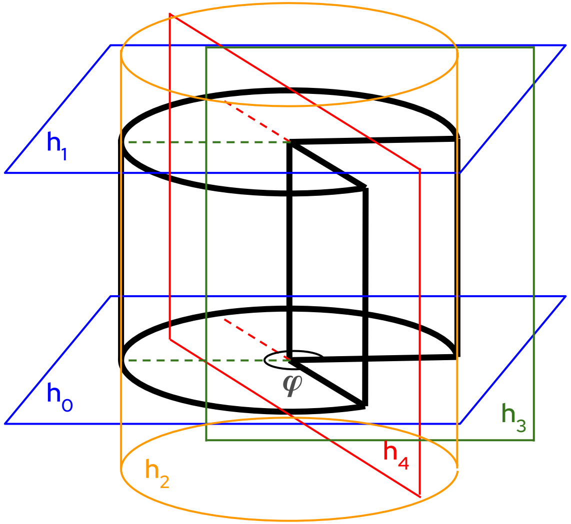
**3 solid / 18 unbounded
surface checks**

High potential for work reduction
compared to solid or unbounded
models

3 framed surface checks

Boolean evaluation for more complex solids

- Cut tube: tube & wedge
  - tube = h_0 & h_1 & h_2
  - wedge = ($\varphi < \pi$) ? h_3 & h_4 : h_3 | h_4
- Inside: Evaluation of the Boolean expression (half-space information only)
  - Inside(h_0 & h_1 & h_2 & (h_3 | h_4))
- Distance/Safety: Ignore Boolean expression for primitives (real surfaces)
  - ToIn/ToOut inferred from the start state (surfaces crossed from the wrong side ignored)
  - Distance(h_i) < dmin && frame.crossed
  - Safety reduction takes into account convexity
- Boolean solids: complete evaluation of Boolean expression needed
  - The Boolean expression can generate virtual framed surfaces

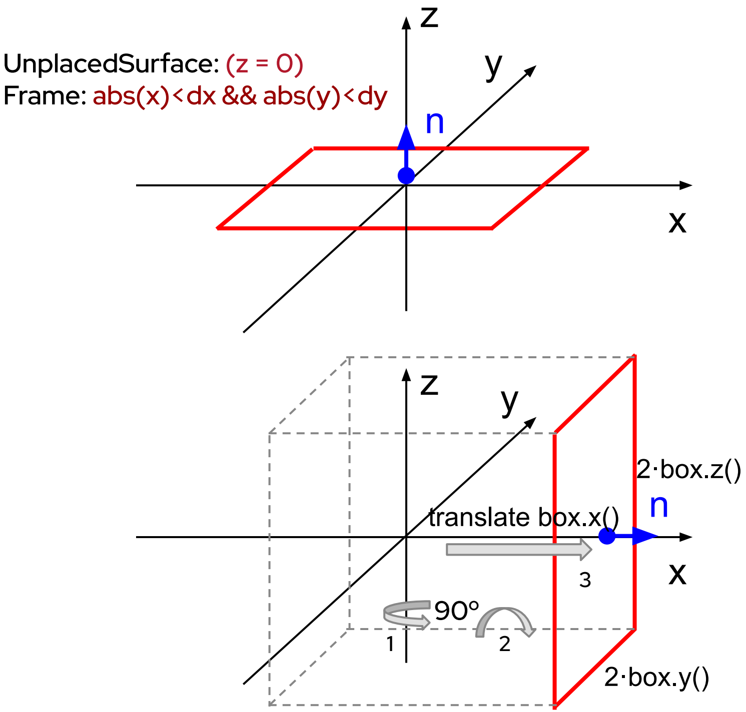


Conversion of existing solids

- Any solid surface can be made from predefined surface & frame types
  - Conversion transparent to user code
- Only box, tube, trapezoid and polyhedron for now
  - And their Boolean combinations

```
CreateLocalSurface(
  CreateUnplacedSurface(kPlanar),
  CreateFrame(kWindow, WindowMask_t{box.y(), box.z()}),
  CreateLocalTransformation({box.x(), 0, 0, 90, 90, 0}));
```

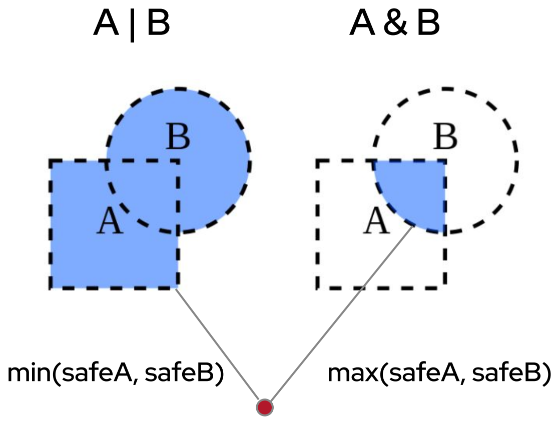
see full box implementation [here]()



No custom navigation needed per solid type once converted to surfaces

Logic evaluation for distance queries

- Common approach for Distance and Safety queries
  - Mix in the search all surfaces visible from the current state (Boolean and regular)
  - Negated surfaces have flipped associated half-space
  - Apply a std::min reduction on the distance to the surface half-space, excluding "far-away" candidates
- Distance computation
  - Validate crossing point against the frame information
  - If this hits a Boolean surface, exclude virtual solutions by checking the logic expression
- Safety computation
  - Use frame information to correct the safe distance
  - Use a stack-based infix logic evaluation using min/max as reduction (correct only if surfaces are 'real')



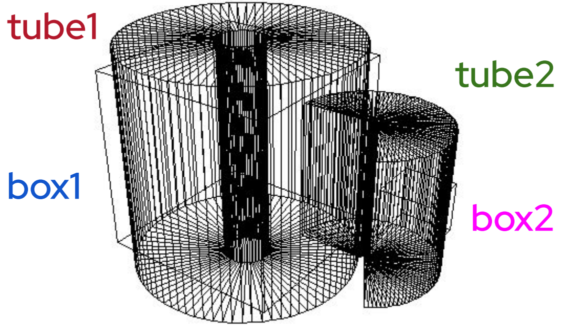
The complex cases: Boolean solids

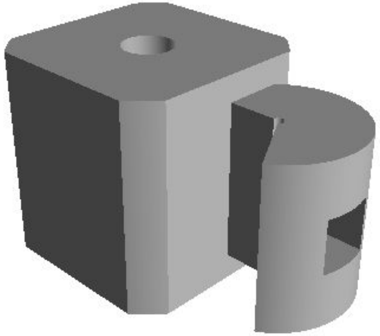
- Composite solids support intersection (&), union (|) and subtraction (&!) of arbitrary number of components
- Building logic expressions in terms of <u>surface id's</u>, using De Morgan's rules

((6 & 7 & 8 & 9) & (10 & 11 & 12 & 13 & 14 & 15)) |
((16 & 17 & 18 & 19 & (20 | 21)) & (!22 | !23 | !24 | !25 | !26 | !27))

- Expression simplification using Boolean algebra rules, keeping left operand the simplest to evaluate for short-circuiting

(6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15) | (16 & 17 & 18 & 19 & (20 | 21) & (!22 | !23 | !24 | !25 | !26 | !27))



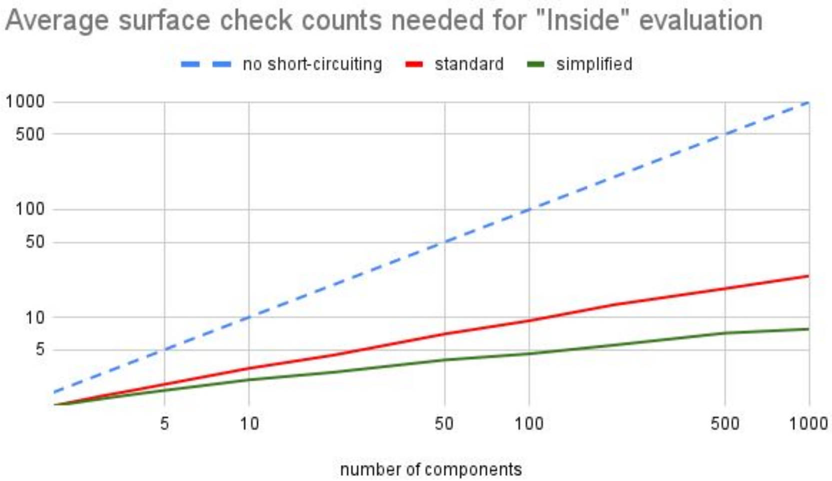


(tube1 & box1) | (tube2 & ! box2)

More implementation details in the backup

Logic evaluation

- Boolean operations can be short-circuited
  - true | any = true, false & any = false
- Infix stackless parsing for Inside evaluation
  - Inserting jumps exiting the current scope



Randomly generated Boolean expression

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(	a	&	b	)	|	(	c	&	!	d	)				
(	a	&	5	b	)	|	15	(	c	&	14	!	d	)