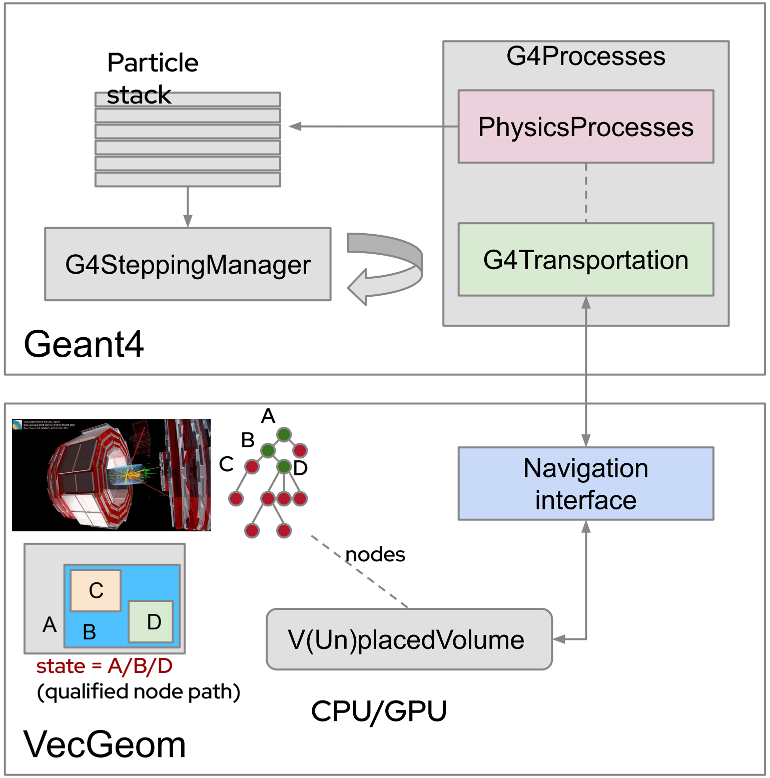
VecGeom

VecGeom surface modelling

andrei.gheata@cern.ch

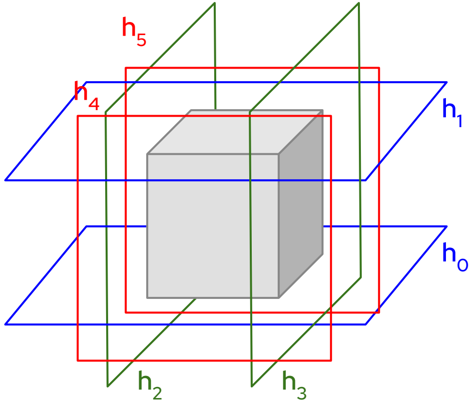
VecGeom: navigation back-end for Geant4

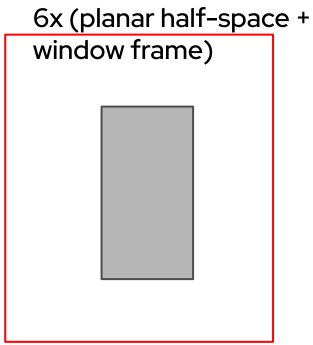
- Collaborative effort to develop most efficient navigation algorithms on top of Geant geometry description
  - Independent of the transport simulation toolkit
  - Supporting GPU (CUDA) as back-end
- State of the art geometry navigation backend for Geant4
  - **Hierarchic** CSG (Boolean combinations) solid modeling based on **containment**
  - Actively maintained and developed
  - Main bottleneck for GPU sim workflows (see Ben's talk)

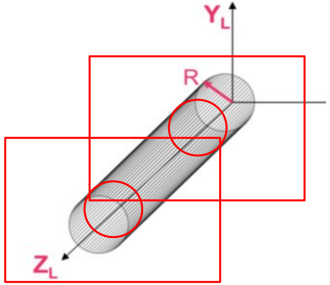


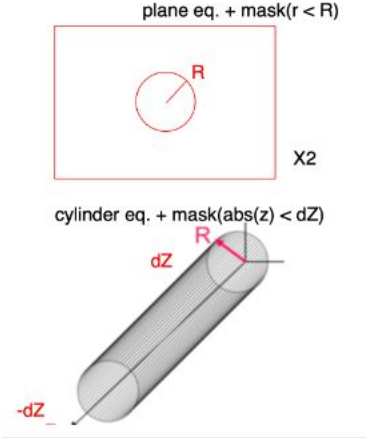
Bounded surface modeling – a different approach for the GPU

- 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
- Storing in addition the solid imprint (frame) on each surface: FramedSurface
  - Frame information is redundant
    - helps taking navigation decisions more efficiently (hitting a framed surface means hitting the real solid)



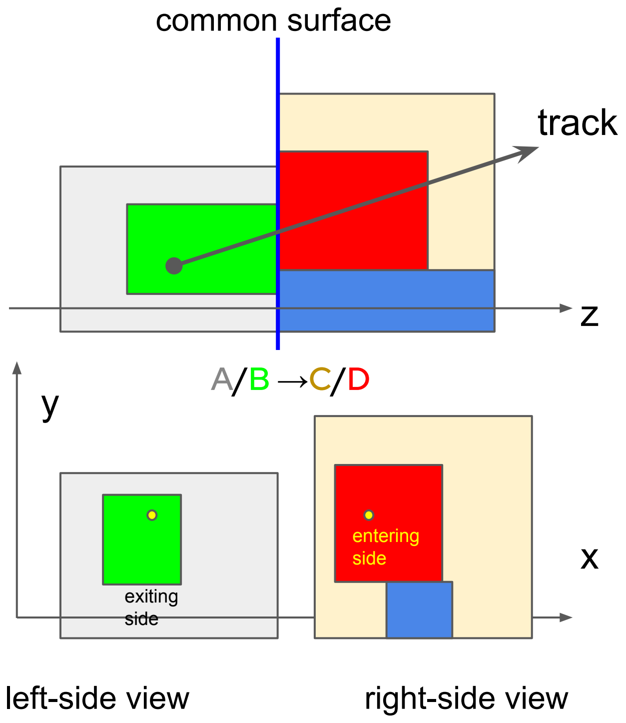






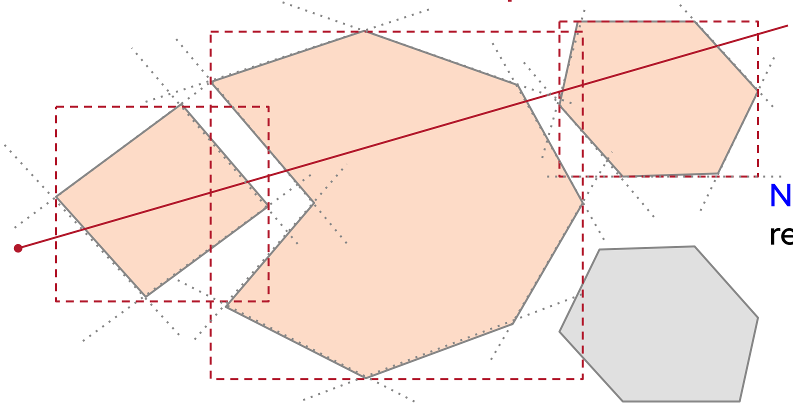
CommonSurface – the navigation primitive

- In Geant volumes can share common surfaces
  - Define "***CommonSurface***" as boundary between volumes
- A common surface is made of two sides, each having hierarchic **FramedSurfaces**
  - Checking frame masking conditions for the track crossing point on each side is equivalent to relocating to the next volume
  - Much cheaper than current volume relocation, non-recursive algorithm



Why use frames ?

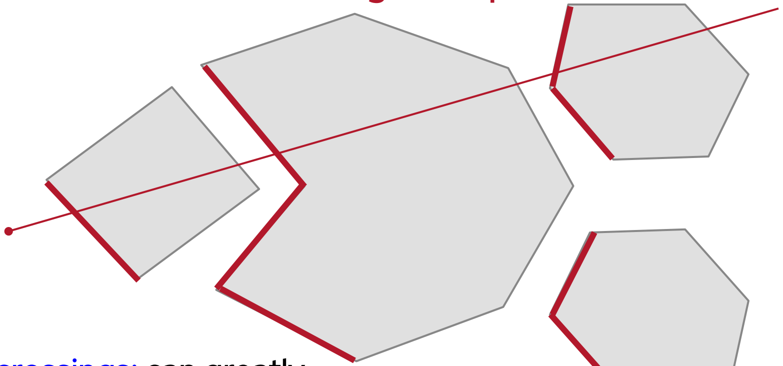
volumes/unbounded surfaces, bbox optimized



3 solid / 18 unbounded surface checks

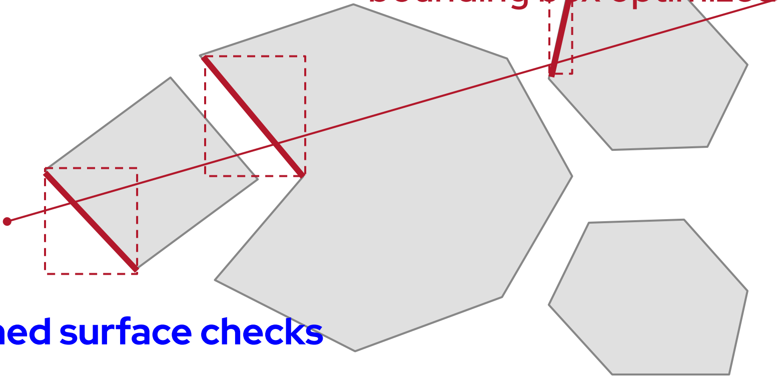
High potential for work reduction compared to solid or unbounded models

bounded surfaces, normal-optimized, no bounding box optimization



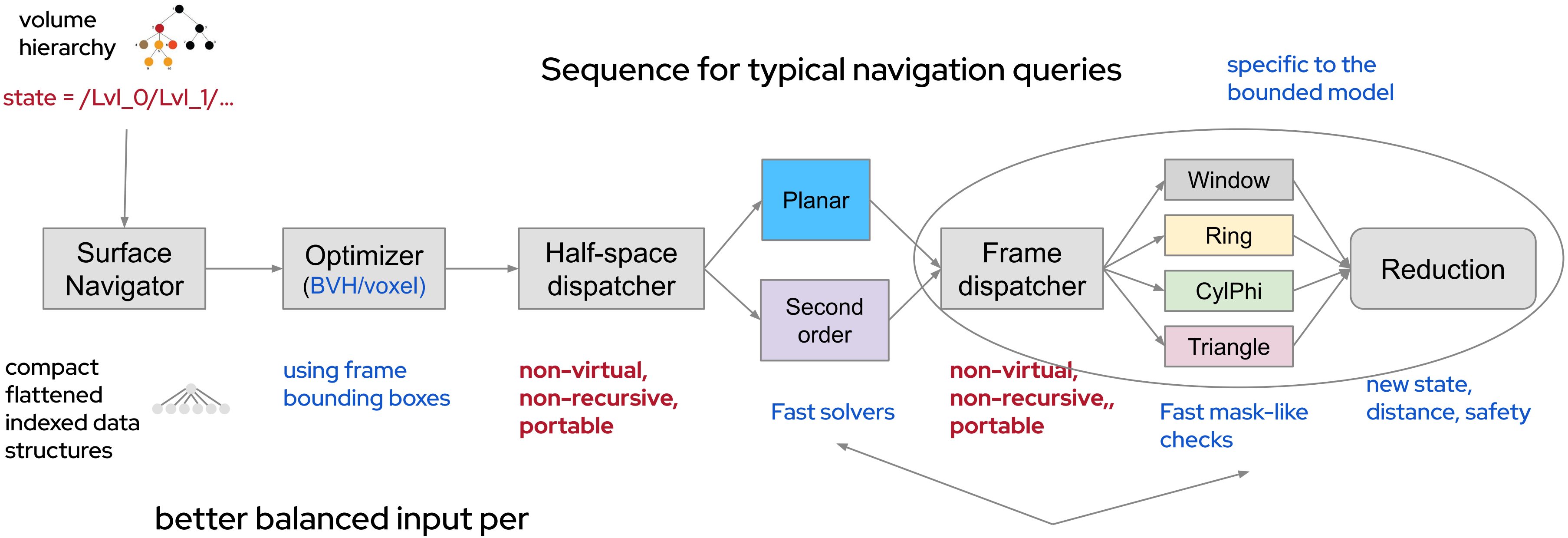
No virtual crossings: can greatly reduce candidates to be checked

bounded surfaces, bounding box optimized



3 framed surface checks

Implementing a GPU-friendly computation pipeline



🤞 better balanced input per particle due to flattening and mixing surfaces from different volumes

🤞 faster divergent sections with fewer branches

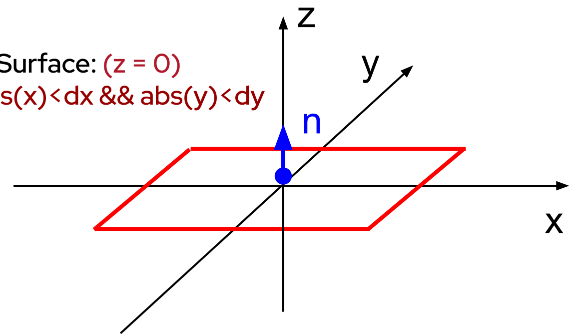
Conversion of solids to framed surfaces

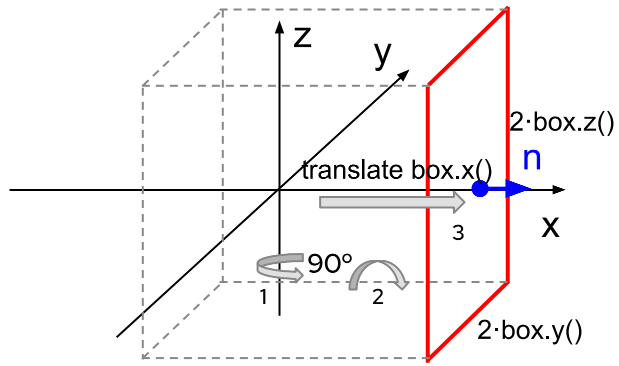
- Any solid surface can be constructed from predefined unplaced/frame types
  - Conversion done behind the hood, implementation completely transparent
- Only box, tube, trapezoid for now
  - The full supported set TBD

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

see full box implementation here

UnplacedSurface: (z = 0)
Frame: abs(x)<dx && abs(y)<dy





Making a box from framed surfaces

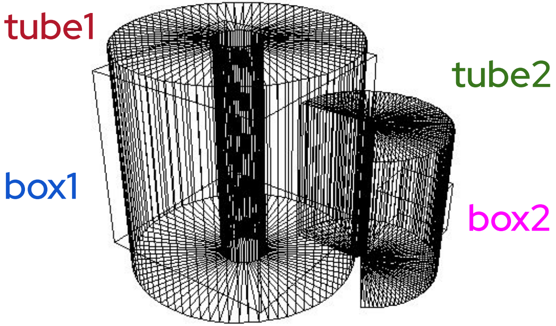
The complex cases: Boolean solids

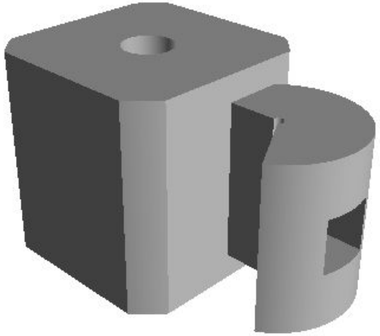
- Composite solids support intersection (&), union (|) and subtraction (&!) of arbitrary number of components
- Building logic expressions in terms of surface id's, using De Morgan's rules

((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))

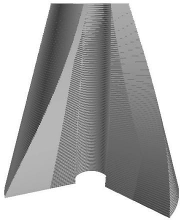




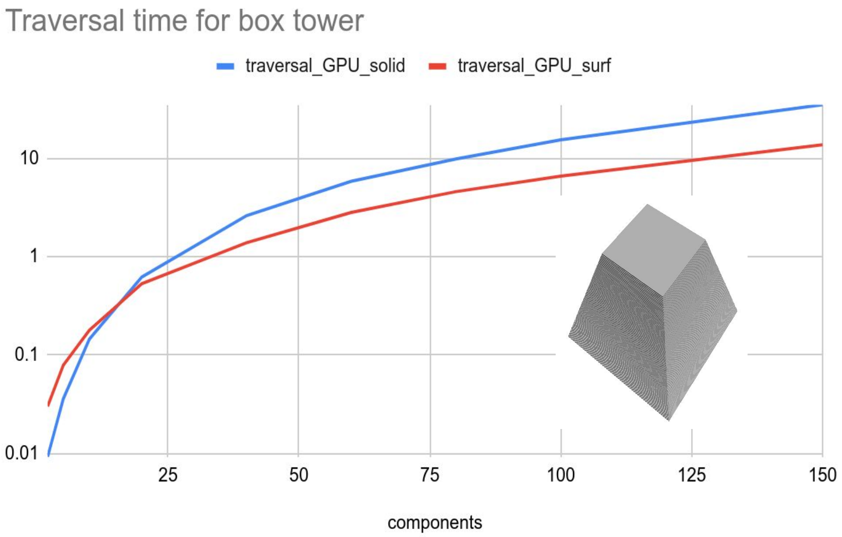
(tube1 & box1) | (tube2 & ! box2)

More implementation details in the backup

Scaling for the Boolean implementation

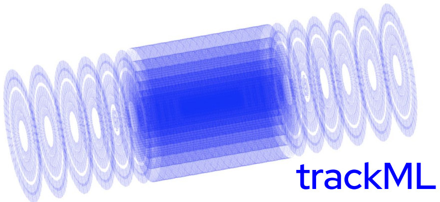


- Current implementation validated for correctness against the VecGeom solid model
  - Tested union of up to 150 layers of disks subtracting a box, more exhausts CUDA stack space for the solid approach
- Un-optimized version so far, but scaling looks good
  - 2x slower for 5 components, 2x faster for 50 components on GPU
  - Finding & tagging the real surfaces of a Boolean composite can help a lot



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

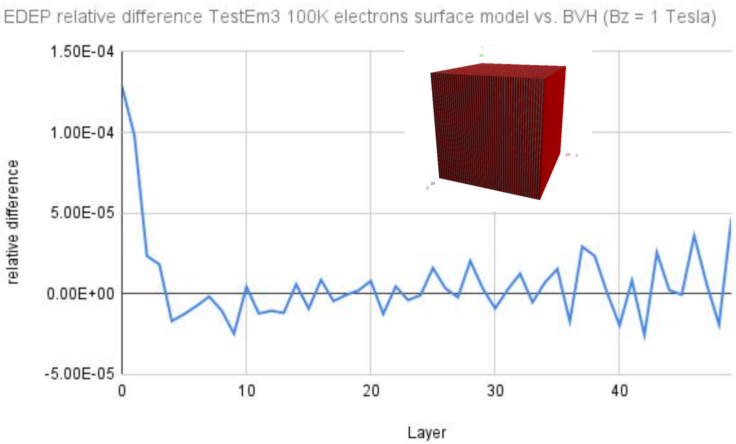
Preliminary performance



- Unit tests available for correctness checking against VecGeom solid model
  - Tube, trapezoid
  - TestEm3 - a simple layered calorimeter made of box slabs
- Ray-tracing benchmark, working with generic GDML input (supported solids only), validated/benchmarked against non-optimized solid navigator
  - Sampling points and directions in the bounding box of the setup
  - Computing location path and safe distance for each point
  - Propagation + relocation between boundaries until exiting the setup
- Results (compared to volume 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

TestEm3 integration in AdePT

- Navigation interfaces of AdePT integrated in the *SurfNavigator* namespace
- Sampling calorimeter block of Pb + LAr box layers in constant Bz field (or no field)
- 10 GeV electrons shot towards the calorimeter along X axis
- Validated to 0.1 per mil level against existing solid navigators (BVH-optimized and simple looper)



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

Next priority items

- Geant 3D solid coverage
  - Currently only few solids supported, we need to write converters for an extended set
- Support for logical scenes of surfaces
  - Pay the price of extra frame transformations for releasing the memory pressure
- Navigation optimization
  - Adapting existing BVH support to framed surfaces

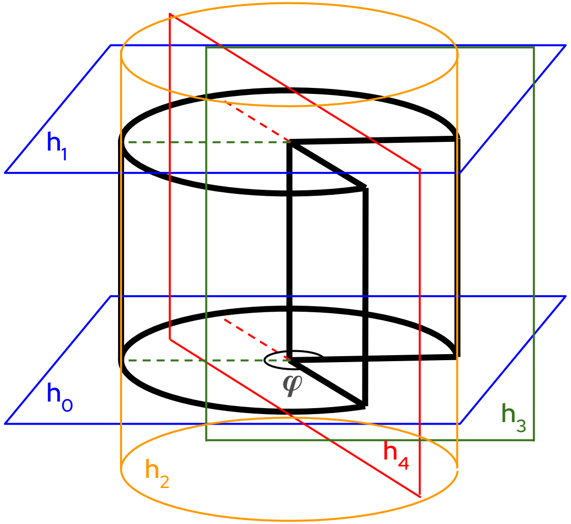
Outlook

- VecGeom went for a surface model approach enriched by solid frame information to be GPU-efficient
  - Even if redundant, the hope is that this allows better work balancing on GPU, avoiding reductions per volume
  - Allows addressing natively isotropic safe distance computation, essential for performance
- Currently implemented all the features required by particle transport, for a subset of solids
  - Integrated with AdePT, already usable with very simple setups
- Extensions and optimizations are essential to judge performance for realistic setups
  - We target the bottlenecks currently observed in AdePT advanced examples

Backup

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



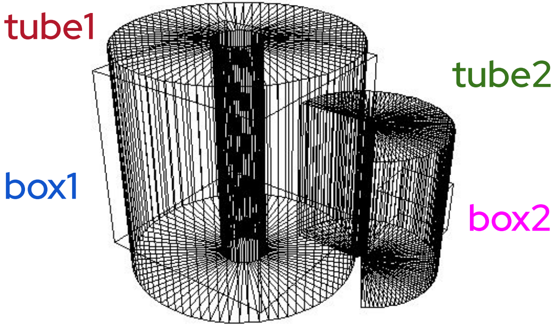
Logic expressions

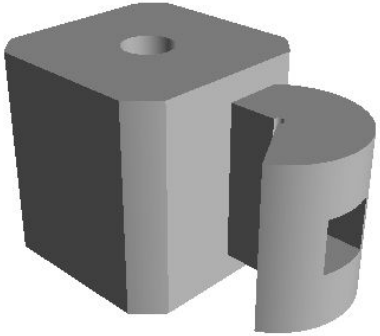
- Composite solids support intersection (&), union (|) and subtraction (&!) of arbitrary number of components
- The logic expressions with solids are expanded in terms of <u>surface id's</u>, using De Morgan's rules

((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))

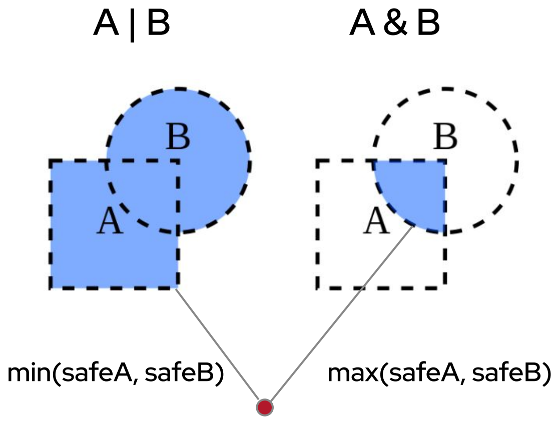




(tube1 & box1) | (tube2 & ! box2)

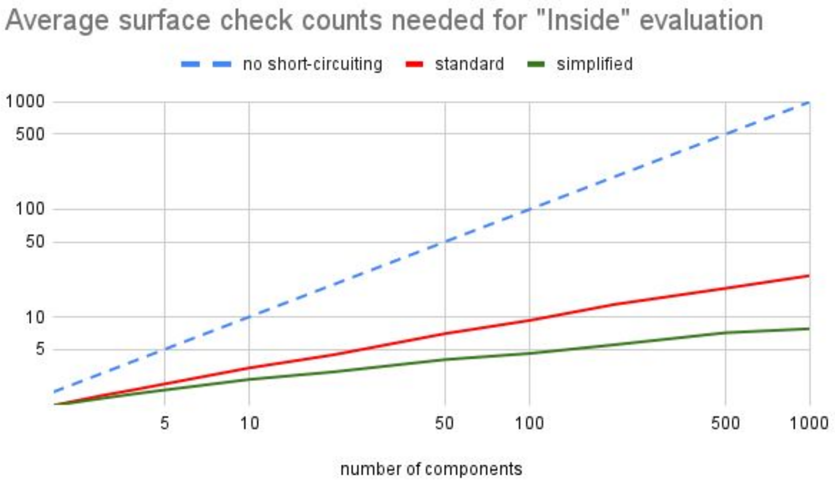
Logic evaluation for distance queries

- 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’)



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	)	