VecGeom

Alternative VecGeom GPU support

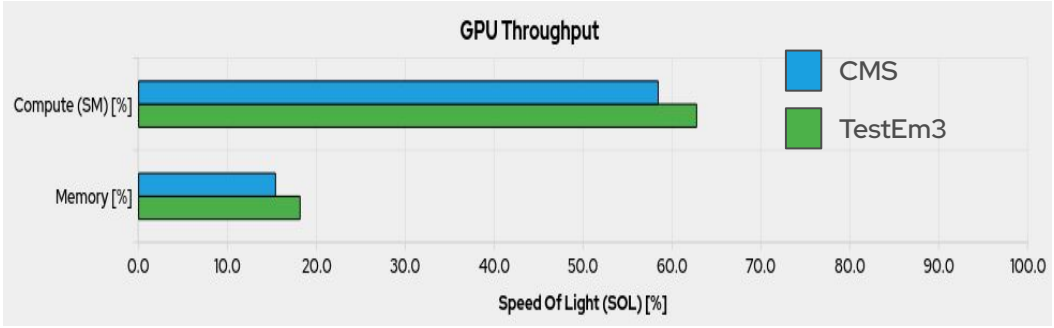
andrei.gheata@cern.ch

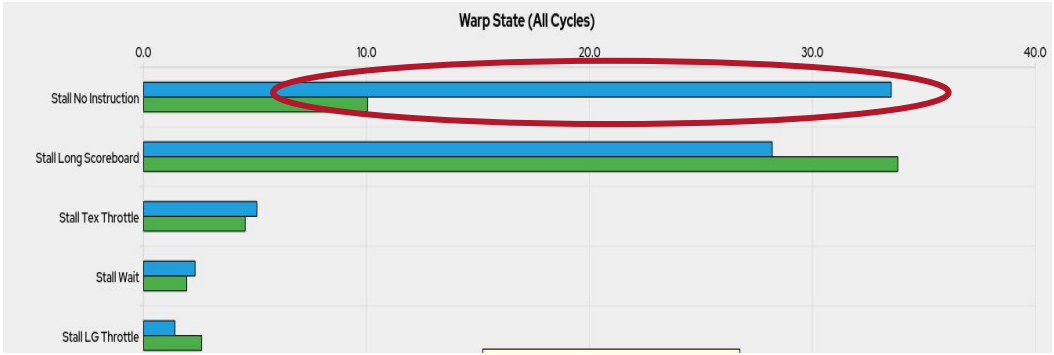
VecGeom@GPU : current limitations

- The current GPU implementation was extensively tested in AdePT and **improved** in many aspects over the last 2 years
  - Initialization time, memory footprint, navigation correctness and optimization, support for single precision
- Some major issues that cannot be “tweaked” still remain
  - The implementation is CUDA specific and non-portable
    - non-NVIDIA hardware currently not accessible
  - Performance is hindered by several factors:
    - virtual function calls
    - code complexity making geometry kernel code register hungry -> limits achievable (efficient) warp concurrency on the device
    - divergence coming for very different computation paths in the same kernel -> limits the low-level thread concurrency

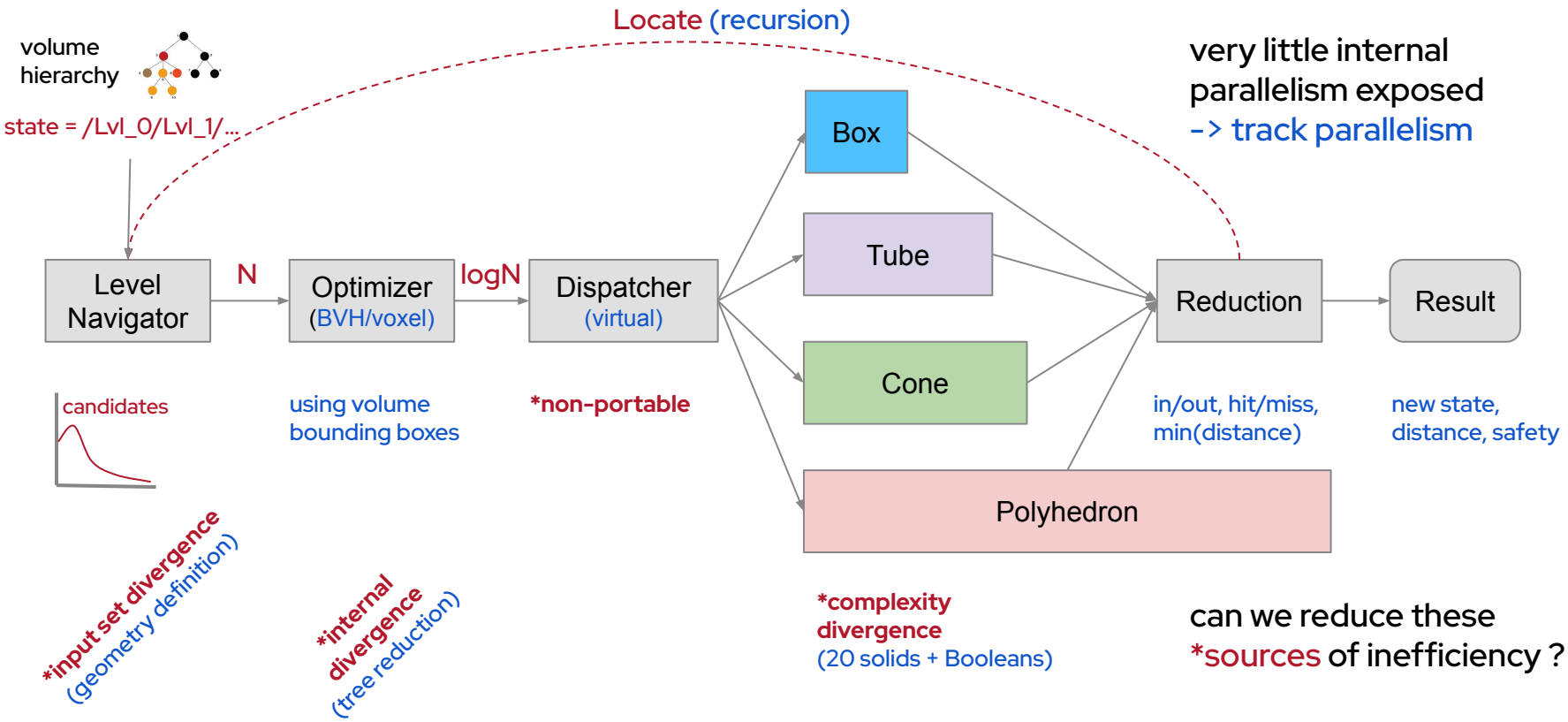
Divergence increases with complexity…

- Moving from simple to complex geometry : longer stalls within warps for the same SM compute
- One of the potential show stoppers for GPU vs. CPU simulation efficiency in complex setups



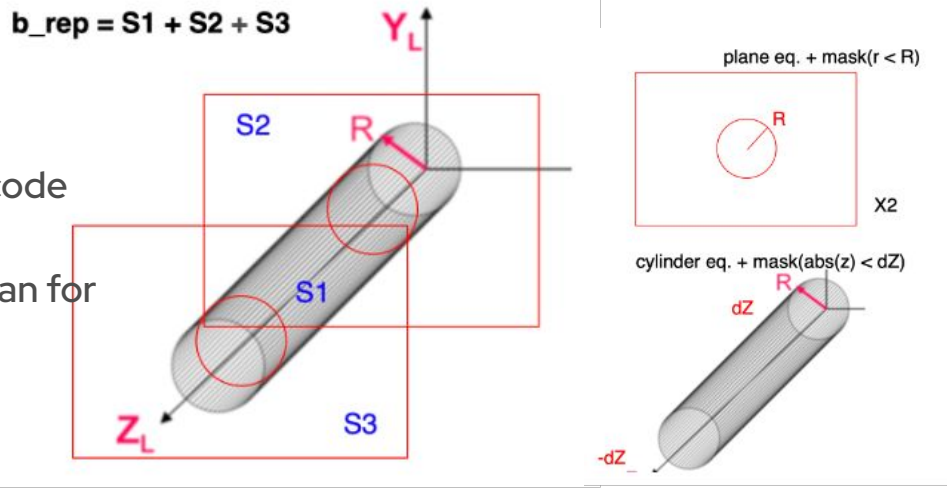


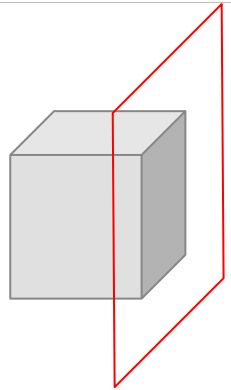
Navigation workflow per track query

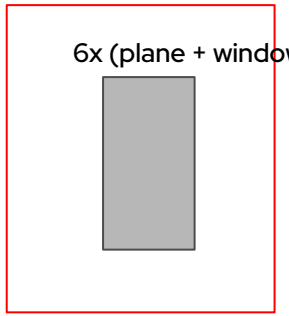


Addressing the problem: surface models

- Rationale: factoring the navigation problem at lower level
  - More simple and uniform code, even if code path is sometimes longer
  - Less branching for primitive surfaces than for primitive solids
  - Allow reducing the number and size of divergent critical sections
- Each face of a solid described as half-space + frame = FramedSurface
  - Same functionality as triangles in a tessellation, but providing accurate modelling
  - Box: 6 x (plane + window frame)

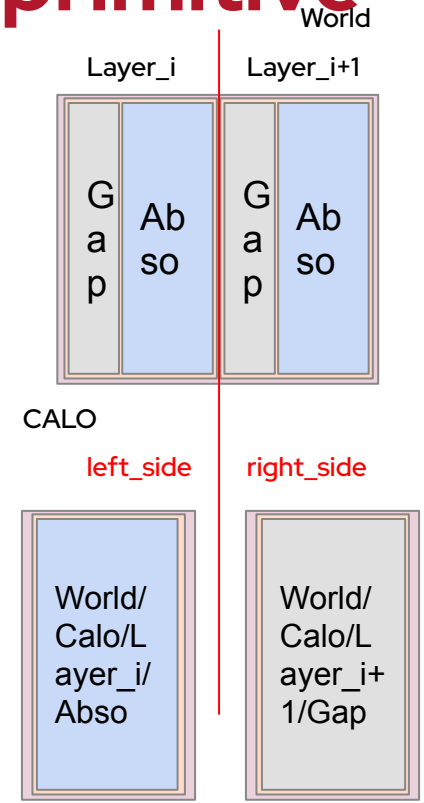






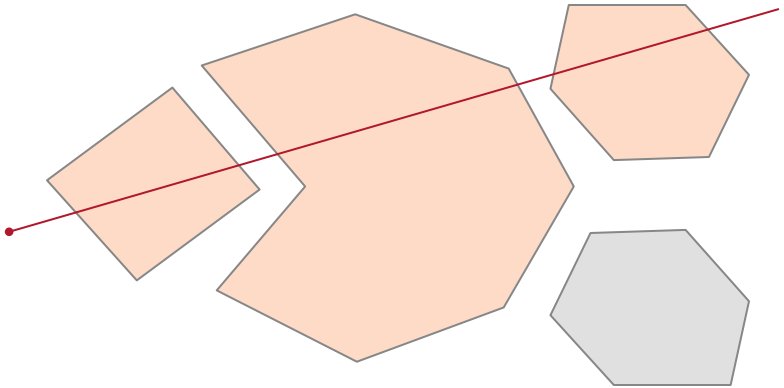
CommonSurface – the navigation primitive

- In Geant models many touchable volumes can have common surfaces
  - ancestor – descendents, neighbour volumes, descendents of neighbours, …
- Each touchable contributes with a **FramedSurface** to a single Side of the common surface
- The model is composed by CommonSurface objects associated to touchables
  - provide transitioning between touchable states when crossing sides

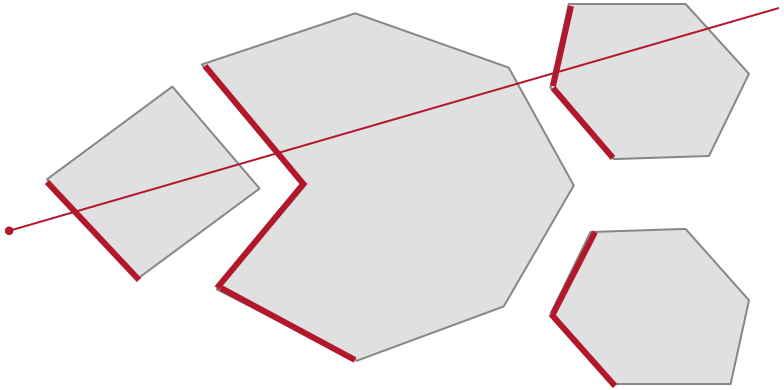


Potential for work reduction

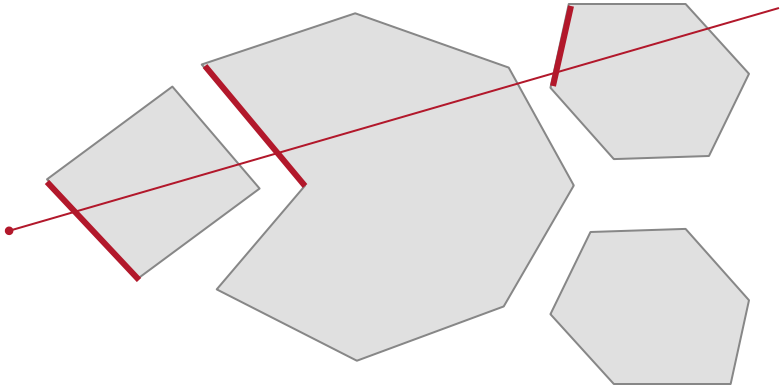
volumes, bbox optimized



surfaces

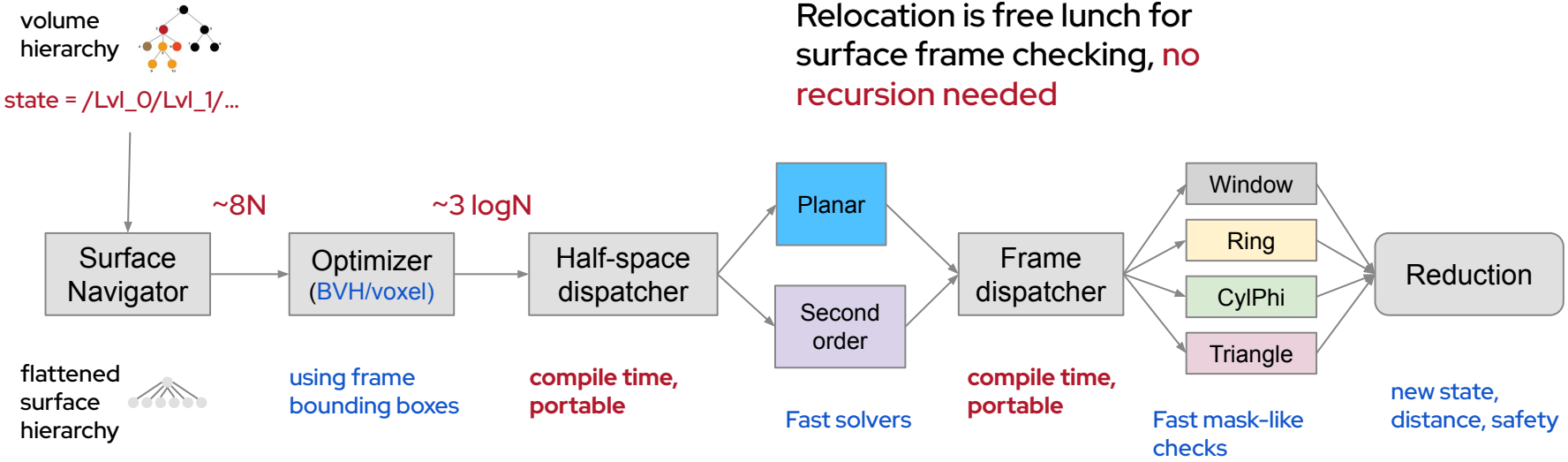


surfaces, bbox optimized



Components to be thoroughly checked, using (or not) bbox optimizations. Surfaces allow for fast exclusion based on normal-direction dot product sign

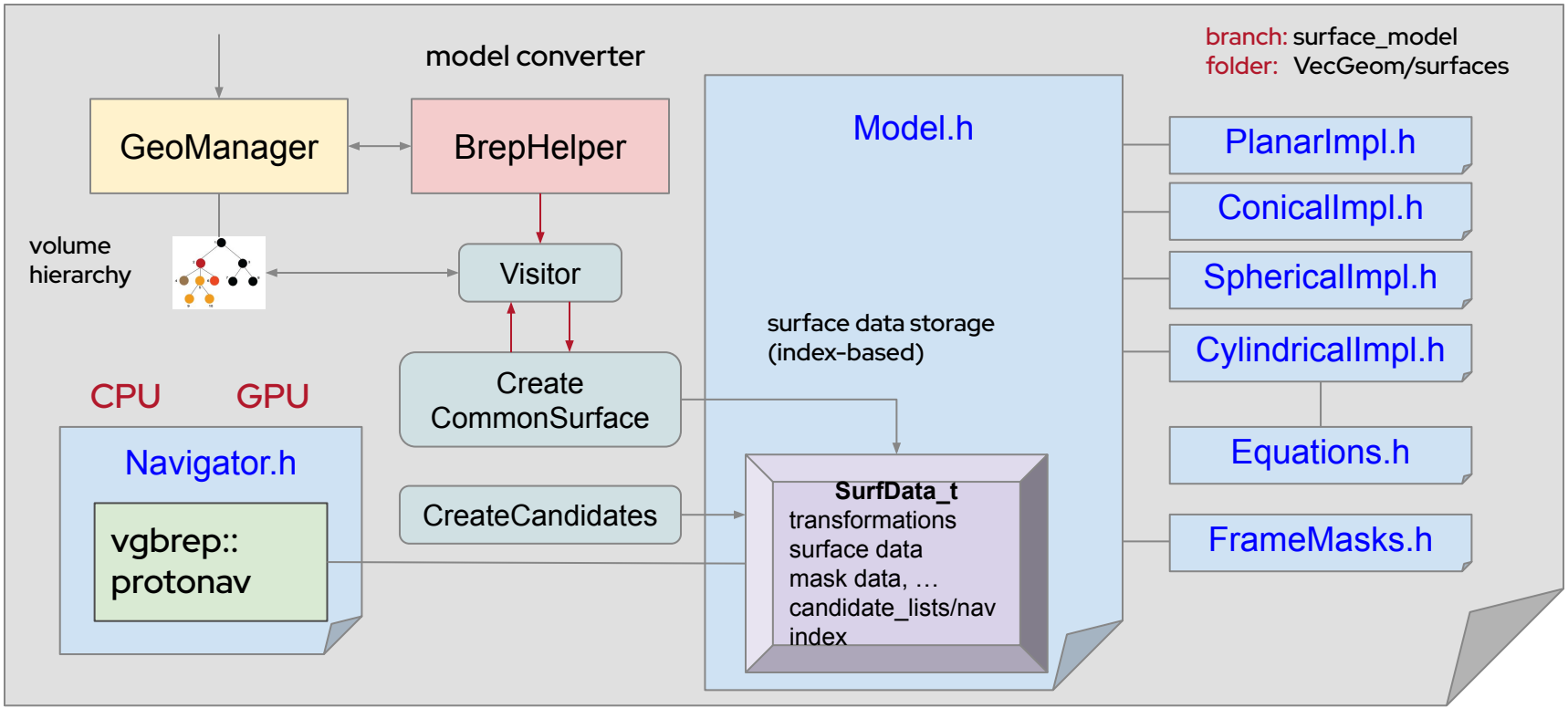
Navigation for a bounded surface model



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

☝ faster divergent sections with fewer branches (2 half-space types and 6-8 masks) (?)

Header-based implementation



Memory and performance

- The first implementation for distance and safety computation available
  - Box, Tube & Trd tested so far
- Memory footprint scales with number of physical volumes
  - Compared to equivalent volume-based flattened hierarchy, use average #sides as multiplier
  - Can reach GByte range for complex setups
  - May require multi-level surface scenes, trading-off CPU for lower memory footprint
- Performance per solid comes with small overhead on CPU for simple solids
  - Same checks but data not local
- Improvements observed when relocation becomes important
  - Relocation is built-in for the surface model (no extra penalty)
- Too early for more elaborated performance statements now

Next steps

- Ready for implementing GPU awareness
  - Header-only implementation, POD types with indices to be transferred
  - No additional code should be needed except function annotations & copy to GPU
  - The data store is percolated through interfaces
- Comparative test of performance in AdePT for increasing geometry complexity
  - Plug-in into geometry-agnostic examples comparing with the volume approach
- If successful, expand the model and implement the missing features
  - Evolving the model to support more solids now much easier
  - Challenges foreseen for the Boolean solid implementation, which may need to map into a volume-based approach

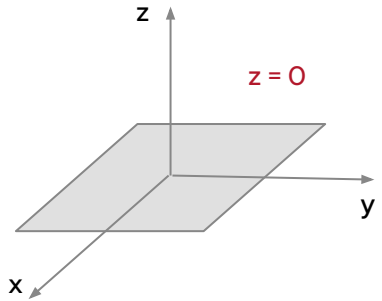
Backup: implementation details

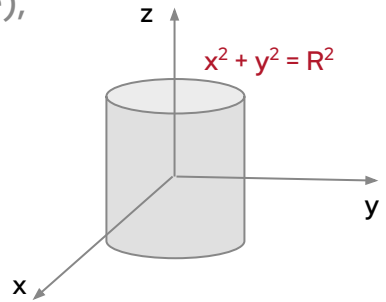
Reference system

- In practice, touchable volumes may be translated and/or rotated
  - So are their surfaces, and we are not dealing with just planes
  - The intersection of a ray with a rotated tube or cone is complicated…
    - …without axis-aligning the tube
- We need to represent surfaces in the most simple reference system
  - As we do it for primitive solids
  - So we need to support local references:
    - RealSurface = UnplacedSurface + transformation
  - disadvantage: needs systematic coordinates conversion to solve the equations
  - advantages: reduces the algorithm complexity, but also the size of surface data and number frame types to be supported

UnplacedSurface

- Representing the half-space (infinite) surface support
- Pick simplest representations for all surface types
  - Plane: use systematically (z = 0) planes
    - surface data - none
  - All second order:: Z-axis aligned, as the Geant primitives
    - surface data - tube (radius at z=0), cone (radius at z=0 and slope), torus (ring + torus radii), ...
- Functionality: ray intersect, safety, inside (half-space)
- Needs to provide a normal for any point on the surface

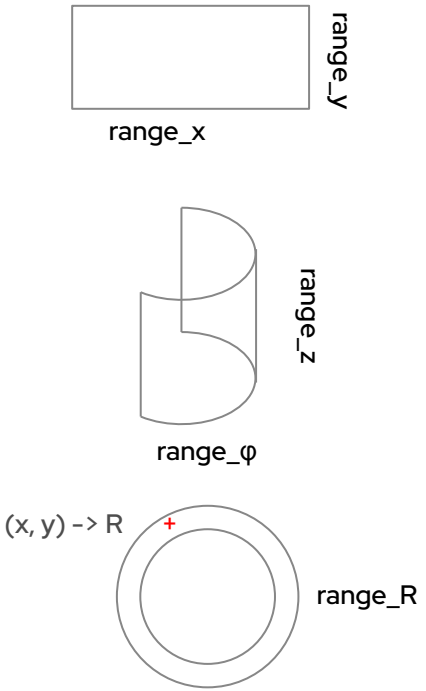





```
enum SurfaceType { kPlanar, [kCylindrical, kConical, kSpherical, kTorus,] / kGenSecondOrder };
```

Frames

- Delimiting the real object side imprint on the infinite unplaced surface
- Typically a set of ranges in two coordinates, depending on the surface/frame types
  - Can also be 1D (e.g RangeZ)
- Functionality: inside/outside frame
  - Needs conversion of cartesian coordinates of the surface point to the appropriate system
  - Simple checks: e.g. rangeX.Inside(x) && rangeY.Inside(y)



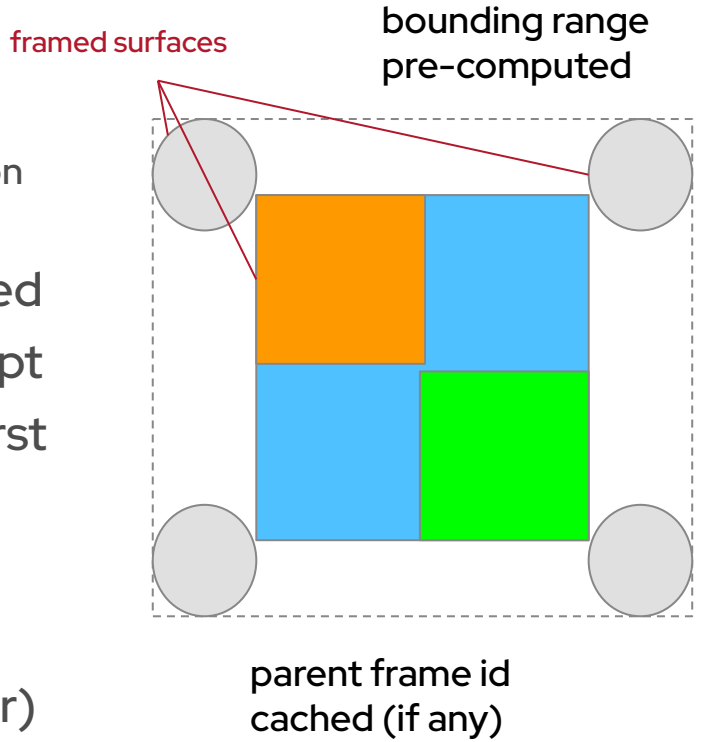

```
enum FrameType {kRangeZ, kRangeCyl, kRangeSph, kWindow, kTriangle, … }
```

FramedSurface – the base primitive

- For navigating we need to deal with touchable object surfaces
  - UnplacedSurface + Frame + Transformation3D + navigation state index = FramedSurface
- The transformation describes the global object positioning in a scene
  - The “world” volume is the natural scene, because a navigation state relates to that.
  - Complex geometries may need composed scenes (e.g. 2 levels). In such case a logical volume becomes a scene, and navigation states have to be represented by tuples (e.g. <world_index, subscene_index> which can be mapped to a regular navigation state)
- Functionality:
  - dispatch to UnplacedSurface + Frame via the Transformation3D
  - connect a surface to the geometry navigation state

Side

- Collection of **FramedSurfaces**
- Left/right arbitrary
  - The first **FramedSurface** used to create a **CommonSurface** defines “left side” and the common surface reference frame
- Identical frames on the same side are detected and only the ones matching deeper states kept
- **FramedSurfaces** are sorted highest depth first
  - priority to highest depth volumes
- The search of frames to be checked when traversing a side have rules imposed by the geometry hierarchy/containment (see further)



Constructing common surfaces

- Check if translations are compatible

  `tdiff = t1.Translation() - t2.Translation()`

  `ldir = t1.TransformDirection(tdiff);`

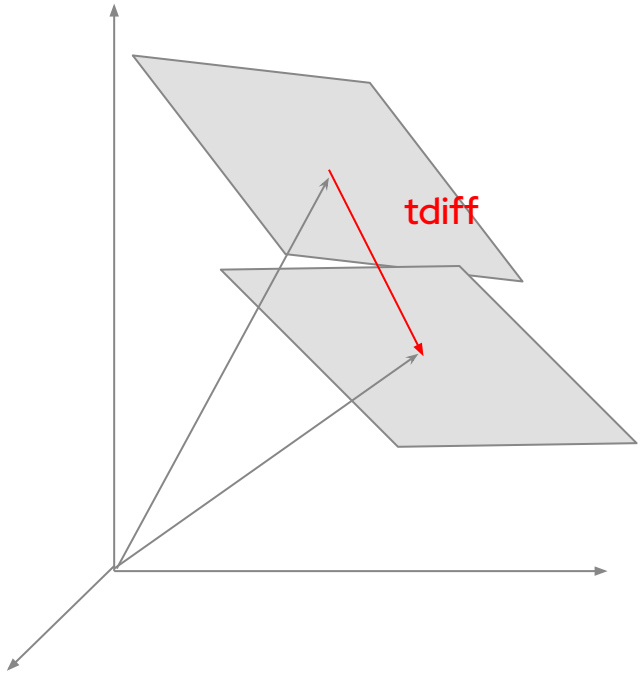
  - planes: `abs(ldir.z()) < tolerance`
  - tubes: `R1 ~ R2 && ldir.Perp2() < toleranceSq`
- Check if rotations are compatible

  `z1 = t1.InverseTransformDirection{0,0,1});`

  `z2= t2.InverseTransformDirection({0,0,1});`

  `ApproxEqualVector(z1.Cross(z2), {0, 0, 0})`
- Finding if the side is matching

  `flip = z1.Dot(z2) < 0;`

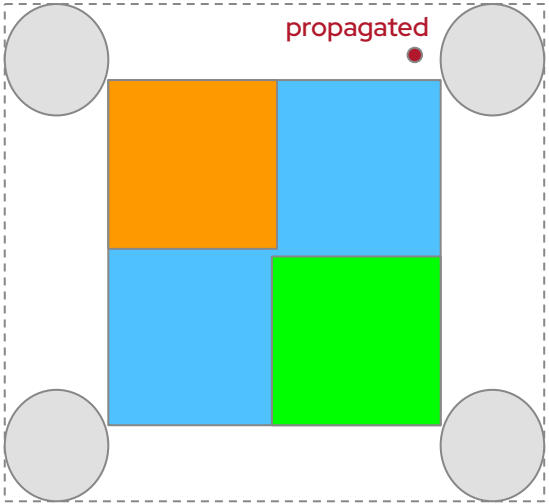


Transitions through surfaces

- Task 1: in_state → surf_candidate (O(ncandidates) if no optimization)
  - We pre-compute a list of candidate surfaces that can be crossed per navigation state index. This gives direct access to surface **id, side** and **index** on the side
  - Check if the side is an exit side for the current state (compare current state with common surface state)
  - Propagate the current point to the common surface and store its local coordinates
- Task 2: check if the current volume is exited
  - The exit side contains a **FramedSurface** with the same state as current one
  - This frame has to contain the propagated point
  - None of the frames with higher or equal state depth must contain this point (otherwise the exit point is in a daughter of the current volume)
  - Cache distance and set *out_state* to be the default surface state

Transitions through surfaces

- Task 3: if the side is an entering side, check if any **FramedSurface** is actually hit
  - If there is a cached parent frame for the side, check it first: if point not in the frame → no hit
  - If not, the bounding range must be hit
  - If yes, then perform a search over all frames on the entering side and cache result (O(n) complexity if no optimization)
- Task 4: Perform **task 3** if needed in case the closest valid surface was exiting the state

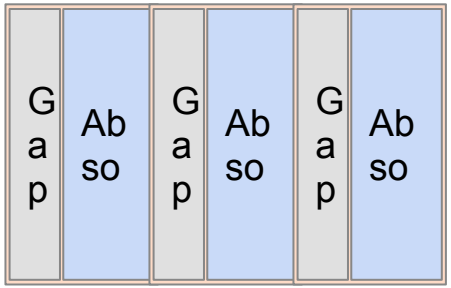


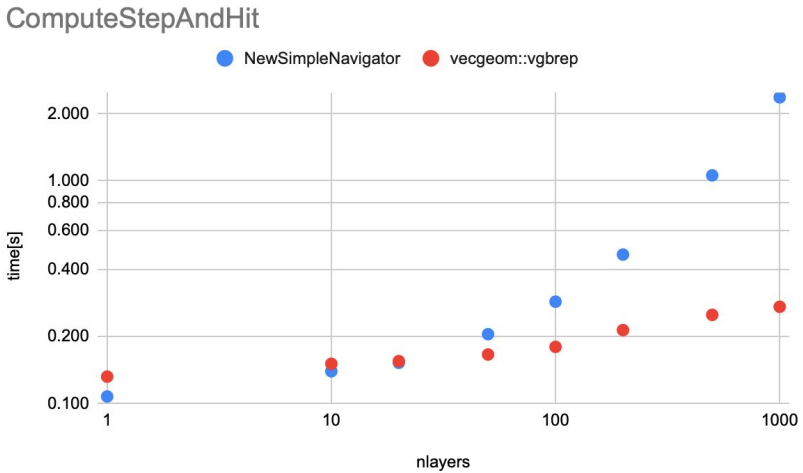
First implementation

- Header-only, available in branch ***surface_model***
- Model.h
  - Data model definition. Indexed component access. Surface store passed via interfaces.
- BrepHelper.h
  - Helper building the data structures based on a closed VecGeom geometry
  - Filling *template <typename Real_t> struct SurfData* representing the store for the surface model data
  - All data indexed, targeting equivalence host/device
- Equations.h
  - Implementations for surface solvers (not implemented)
- Navigator.h
  - vgbrep::protonav namespace implementing navigation methods
  - Currently just ComputeStepAndHit

TestEm3

- Shooting random points/directions in the calorimeter
- Locating in_state using NewSimpleNavigator
- Computing distance to boundary and out_state
- Validating against NewSimpleNavigator
- Comparing performance
  - Crossing internal surfaces more efficient (relocation not scaling with nlayers)
  - Shooting from outside still slow, but several helpers still missing (e.g. normal checks)





Next steps

- This is just a first try, and the model seems to “hold” together and interface well with the current vecgeom approach
- Important functionality missing, in particular global location (which now relies on the current approach)
  - Need to add normals and use them in navigation, safety computation
- Second order surface types and most frame types to be implemented
- Conversion now possible only for boxes -> summer project to do other shapes
- No treatment yet for Boolean surfaces
  - Non-trivial, planning to inspire from Orange