VecGeom navigation : Status and Outlook (Focus on G4-VecGeom integration)

Sandro Wenzel (CERN, ALICE)

Outline

Like to talk about

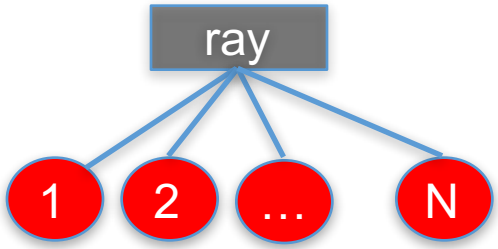
- Reminder of VecGeom navigation module; main highlights
- Towards full integration of VecGeom into G4
  - What would this mean?
  - How can it be attacked? What are challenges?
  - Current development status
- R&D: benefiting from industry ray-tracers (such as Intel Embree)
  - application to both “navigation” and complex “solids”

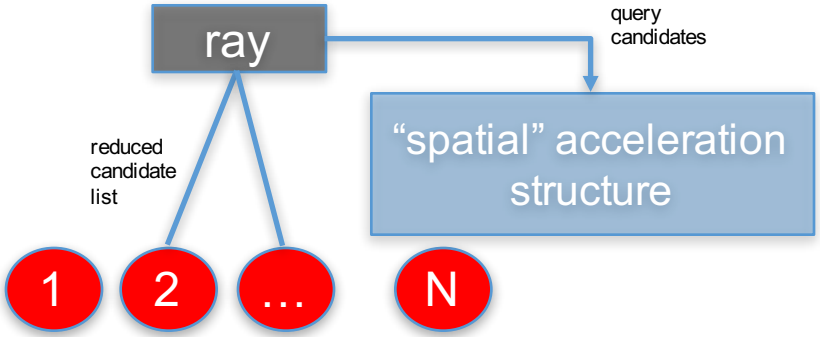
VecGeom: Key Reminders

- VecGeom is evolution of G4 / TGeo / Usolids geometry with the goal to
  - use SIMD acceleration as much as possible (multi-particle API, single-particle API)
  - modernize / revise / optimize algorithms in general
- Two main components provided for use in simulation:
  1. **elementary and composite geometry primitives** (box, tube, boolean, ...)
     - distance / containment / etc algorithms
     - bricks to build complex geometries
  2. geometry modelling + **navigation**
     - "fast" determination of the next (straight) line intersection of a ray and the distance
     - determination of next "geometry path" at boundary traversal
- **Part 1 is already (partially) available for use in G4**, with proven benefits for users using lots of complex geometry primitives (polycone, polyhedron, tessellated, multi-union)
- **Part 2 not yet available for use in G4**

Spatial acceleration structures for hit-detection: Novelty in VecGeom

- Given ray in 3D world: determine intersecting objects from N candidates
- Brute force algorithm checks against all N candidates
- Spatial acceleration structures ("voxelization" structures) provide reduced O(1) candidate list in typically O(log(N)) time

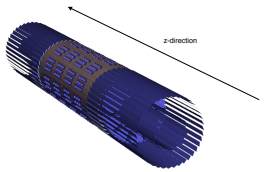




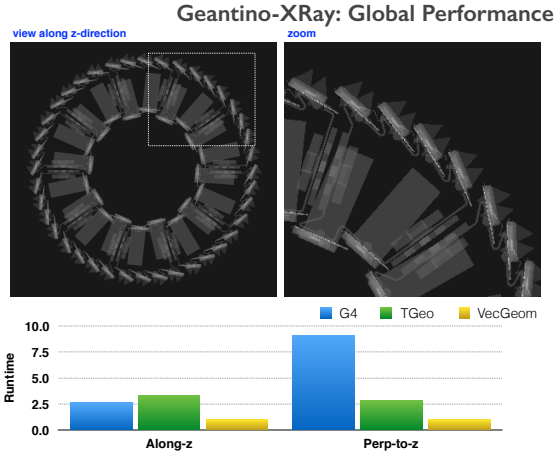
- Querying acceleration structure is core of algo; Optimizing this has large impact
- **VecGeom implements structures which benefits from SIMD** (vectorized search/traversal of structure); Increasing size of vector registers on future hardware will automatically make algorithm faster

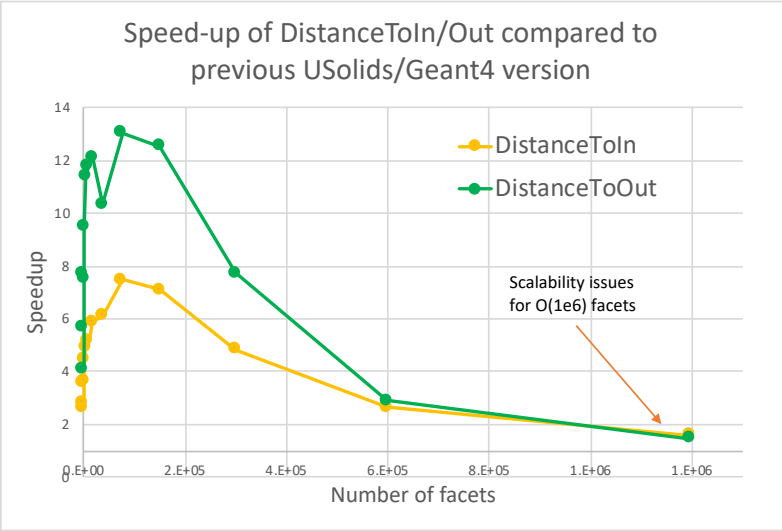
Reminder of a few benchmarks

- **Good speedup in global navigation tests**
  - Pure geantino traversal of complex geometries, comparing G4, TGeo, VecGeom navigation algorithms
  - Speedup due to solid algorithms + SIMD acceleration structure
  - Presented at CHEP16



- Example for ALICE ITSSPD module
- Perfect agreement between G4/ TGeo/VecGeom
- Observe generally **factors > 2.6x** speed improvement against other packages
- Another indication of global performance advantage of VecGeom

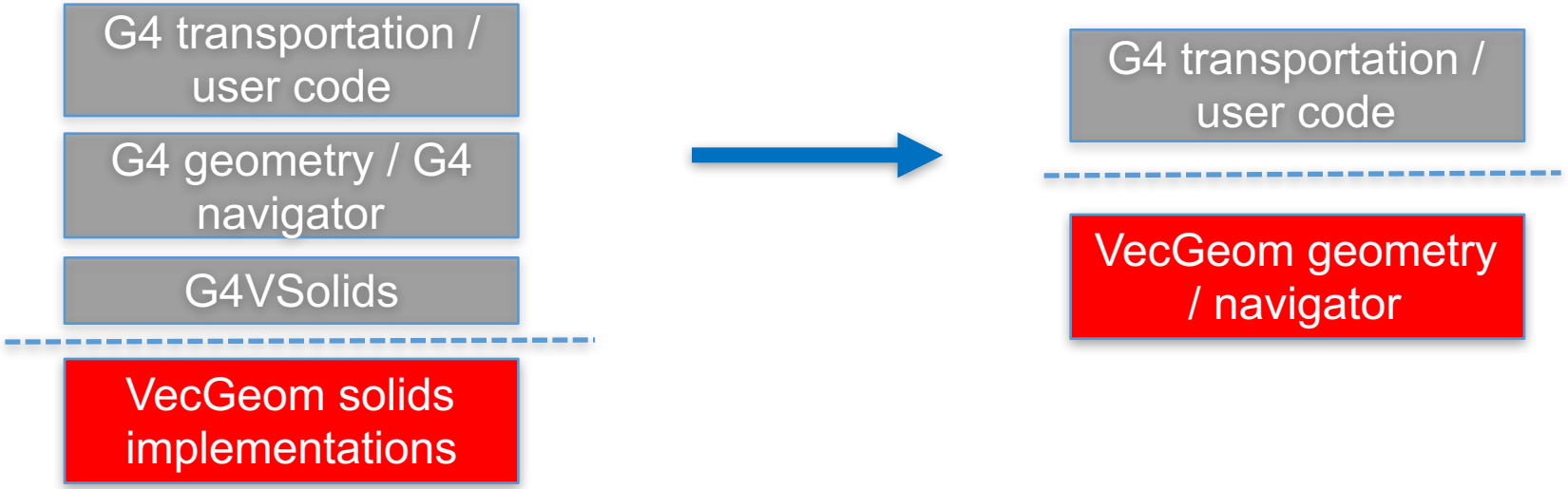




- **Application of SIMD spatial structure to tessellated solid (and multi-union) leading to substantial speedup**
  - In range of typical applicability excellent performance improvement due to SIMD + algo
  - Performance degradation for very large N due to incomplete/missing implementation in VecGeom (see later slides for solution)
  - Presented at CHEP18 (by M. Gheata)

Interface G4 to VecGeom navigation

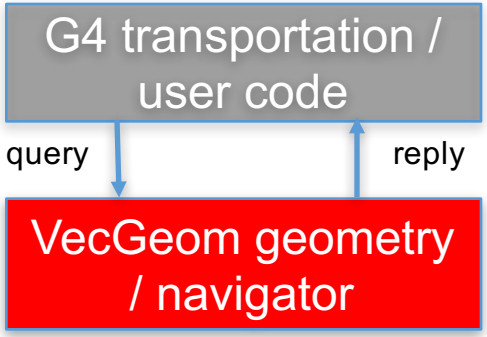
- **Good reasons to believe that interfacing the VecGeom navigation module would be beneficial for a standard G4 simulation**
  - beyond the benefit available today
  - besides SIMD, other advantages like strong solid specialization would be made available, especially important for simple solids
- **What does this mean in the ideal case? – Moving the boundary!**



Challenges / Options

Challenges:

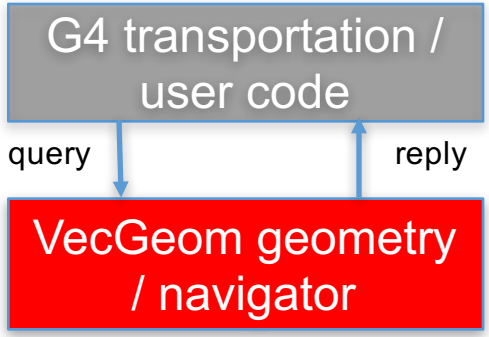
- G4 transportation and VecGeom speak different types
  - Example: **LogicalVolume** and **NavigationHistory**
- G4 has public APIs to query G4 geometry states
- Users often specify geometry natively with G4 and query/use G4 geometry objects in sensitive action; They surely want to benefit from VecGeom without changing their code



Challenges / Options

Challenges:

- G4 transportation and VecGeom speak different types
  - Example: **LogicalVolume** and **NavigationHistory**
- G4 has public APIs to query G4 geometry states
- Users can construct geometry natively with G4 and query/use G4 geometry objects in sensitive action; They surely want to benefit from VecGeom without changing they code

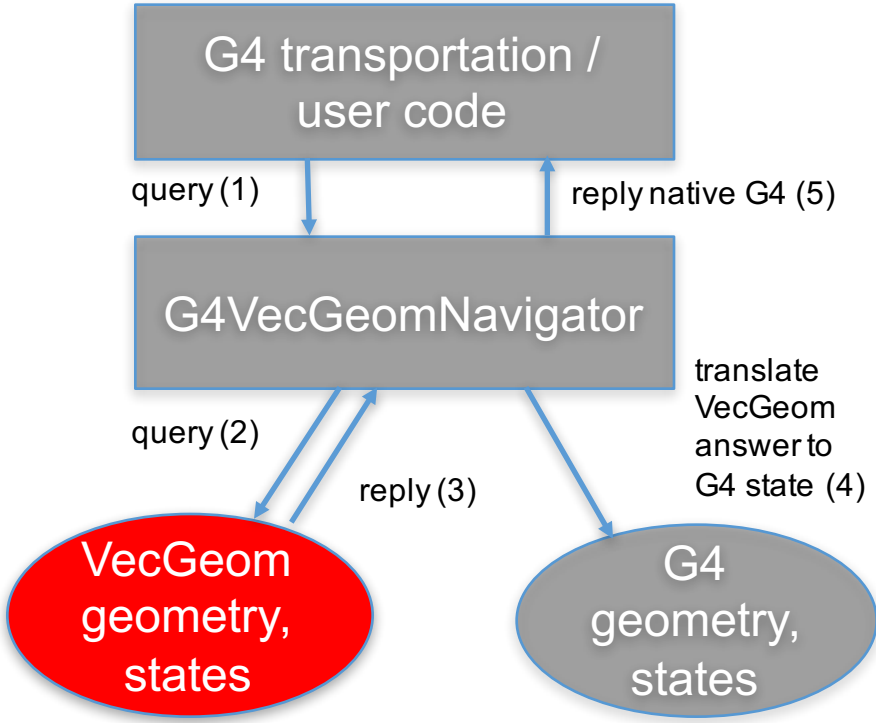


Hence, I see 2 (extreme) options (with variations in between)

1. **User friendly, easy-to-implement, but "some overhead" option:**
   - Simultaneous existence of G4 and VecGeom geometry with necessary synchronization/translation of states and objects between 2 worlds
2. **No overhead, full integration but much more complex and potentially user-disturbing**
   - Complex changes required in VecGeom and in G4; API and type evolution, more abstraction layers, …

VecGeom – G4 integration: Option 1

- Going back to developments / technique pioneered by ALICE a few years back for TGeo
- Interface navigation yet **hide** VecGeom from G4 transportation and the user code via the **virtual nature of G4Navigator**
  - just plug a specialized navigator instance
- We keep 2 geometry instantiations and sync their state
- **G4VecGeomNavigator** always queries VecGeom algorithms but translates answer to G4
- The translation is an overhead in this method that should be minimized



Option 1: Status of Development

- Large parts of development done (momentarily in a fork of Geant4_VMC repository)
- Most of G4Navigator interfaces implemented
- Can fully create in-memory copy of VecGeom geometry given original G4 geometry; synchronization of geometry state implemented
- For testing, use a generic G4 application that can take any GDML geometry as input
  - Same application can either use native G4 or VecGeom navigation (very user convenient)
  - **`G4SimTest --geometry gdmlfile [--use-vecgeom]`**
- `G4VecGeomNavigator` validated for geantinos
- For real particles, still debugging some problems potential due to bugs in safety calculations (May 2018)
- No performance numbers yet

Further plan and action list

- Plan to get to get this working in the coming months (scheduled some time)
- Will test and profile extensively the ALICE use-case, by replacing TG4TGeoNavigator with G4VecGeomNavigator
- Likely, some minor adjustments or specializations need to be provided in VecGeom to yield best performance for G4 use-case
  - things related to caching of transformations for single-particle propagation (not canonically done in VecGeom)
- Once working, can try to think about Option 2 …
  - requires refactoring / abstracting / “templatizing” the geometry state classes; Some discussion started with J. Apostolakis
  - might/will have some impact on user-API, user code.

Interfacing external ray-tracing libraries

- VecGeom provided a demonstration of speedup with SIMD accelerated navigation… but
  - not perfect or complete yet …. see for instance large-N scaling problems in current version (highlighted by use in tessellated solid)
  - very limited time for further systematic developments and maintenance
- Can we benefit here from collision detection components from outside of HEP (such as “ray-tracing”)?
  - might not need to reinvent the wheel, could benefit from best algorithms out-there
  - could use industry tested stuff developed by larger community
  - could have less maintenance burden
- Of course external libs usually not a universal solution …
  - floating point precision … so we can’t just use them anywhere
  - algorithms focus on collision detection (much less on isotropic distance)
- One open source candidate is Intel Embree

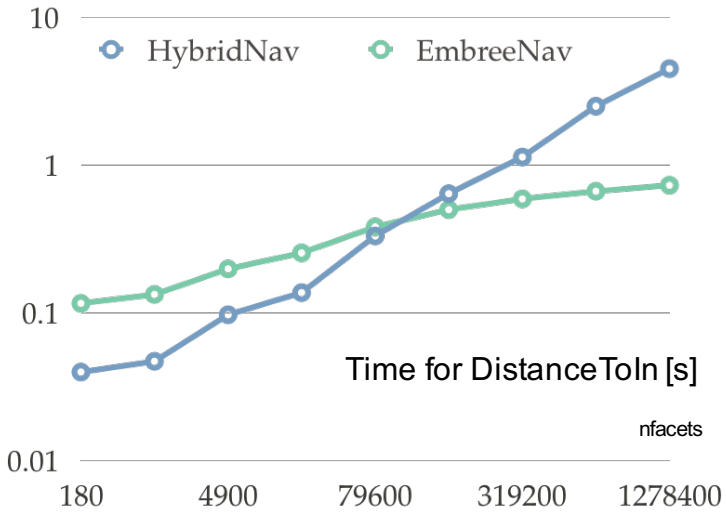
https://embree.github.io/

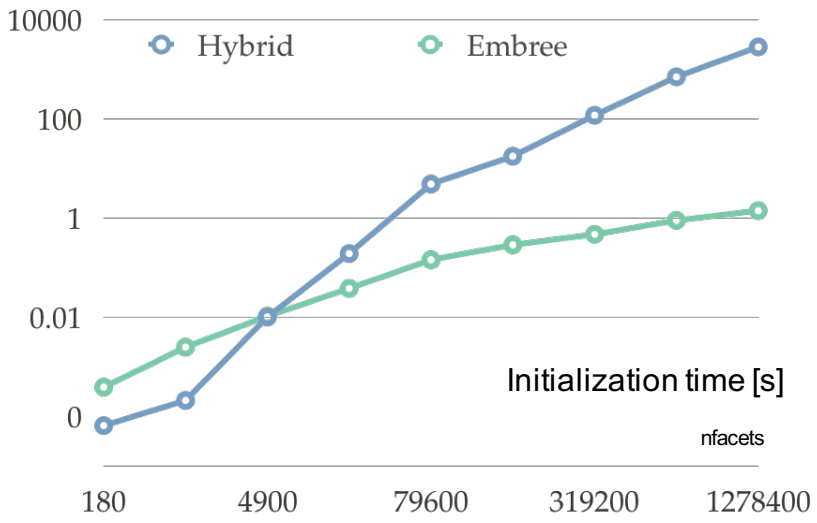
Intel Embree: Plus points

- "Intel Embree is a library offering high-performance **kernels** for ray-tracing applications" https://embree.github.io/
- Some points that should be very interesting for us:
  - offers SIMD spatial acceleration structures (in terms of bounding boxes) … quite similar to what is implemented in VecGeom
  - tuned implementations for different SIMD architectures and automatic runtime dispatch to best architecture
  - better scaling in the very large N limit (number of objects, or facets)
    - VecGeom focused on small or medium N; did not implement large N version yet
    - Embree naturally optimized for large N
  - multi-threaded support to build-up spatial acceleration structures
- Could we just benefit from this inside VecGeom navigation to complement our usability? Yes we can!

preliminary

Initial Embree Study





Initial test using an Embree spatial acceleration structure instead of VecGeom one inside tessellated solid

- Embree has advantage for nfacets > ~ 80000 for DistanceToIn
- Embree better initialization times for nfacets > ~5000 (depending on number of threads)

VecGeom (original) performs nicely in the lower N limit!

Embree will give us optimized access to large N limit!

For more info, see report here

Summary

- Reminder of benefits of VecGeom: SIMD support across solids and navigation !
- Interfacing with G4 underway, focusing on Option 1 to start with
  - benefit to be demonstrated
  - Option 2 can be addressed thereafter
- Using industry proven kernels inside VecGeom will solve some remaining problems and keep VecGeom state of the art

Backup Section

SIMD-enabled spatial acceleration structure

- VecGeom is building a spatial acceleration structure based on the **axis-aligned bounding boxes of objects**
- These bounding boxes can be used to **construct a regular tree** (or similar), where
  - each node has exactly the same number of M daughters
  - a node represents either a bounding box itself or some hyper-box formed of other boxes.
  - finding the best hyper-box nodes needs clusterization algorithms
- The regularity of such a tree enables **SIMD accelerated traversal**; i.e., a particular node on the tree can be found using vectorized search.
- Such algorithms have been discussed in recent literature and are used on industry ray-tracers (see later)

- research paper link
- CHEP16 presentation; CHEP16 proceedings