VecGeom

Status of developments in VecGeom

andrei.gheata@cern.ch

26th Geant4 Collaboration Meeting

September 13, 2021

Build and integration

- Simplify/unify the use of C++ compiler flags
  - Final goal to export CMake/pkg-config files to clients of *libvecgeom* so that these are applied automatically
  - Cleanup duplications, removing unnecessary options
  - Provide C++17 support for Intel compilers using usage requirements
  - More common options supported by Clang/Intel/GNU and some exposed as CMake option
  - Good step towards better CMake target-based compilation options support
- Adopting features of modern CMake
  - In particular for CMake CUDA first class support
  - Adding static/shared support for separable compilation and RDC
    - Including usage requirements/recipes for clients (immediate ones are AdePT/Celeritas)
  - Work in progress
- Added support for CUDA compilation with clang (more pedantic than nvcc)

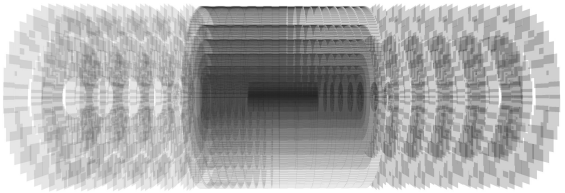
Persistency: vgdml status

- Several fixes and improvements introduced recently
  - Possible now to validate geometry against Root (and Geant4 in AdePT - WIP)
- Support for changing the internal length unit when reading geometry
  - Avoid unit conversions for every navigation call when client library has different unit
- Reading auxiliary information and storing it in maps
  - New API exposing these maps (*ReadMiddlewareData* provided as usage example)
- Missing features of the parser:
  - *replicavol* now skipped, requires functionality in core vecgeom library
  - support for arb8 tag which is the GenTrap in VecGeom
  - No writer available yet, but also no urgent use case for it

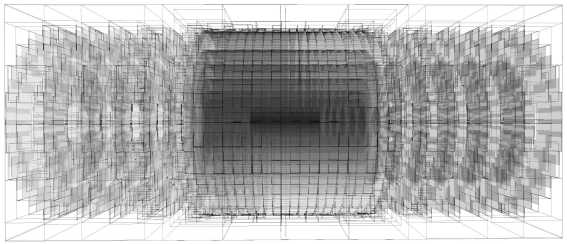
andrei.gheata@cern.ch

Navigation: BVH acceleration

- Acceleration structure for reducing the number of candidate checks
  - Used natively in RTX hardware, available via Nvidia Optix
  - We had an implementation in vecgeom (HybridNavigator2)
    - SIMD, but not GPU friendly
- New BVH navigator available now for both CPU and GPU
  - Important performance gain compared to “looper” version
    - Speedups: CPU: ≈38.24x, GPU: ≈58.2x for trackML, ~2x for cms2018 in AdePT GPU examples
  - Only up to 30% slower on CPU compared to the AVX2-accelerated HybridNavigator



TrackML

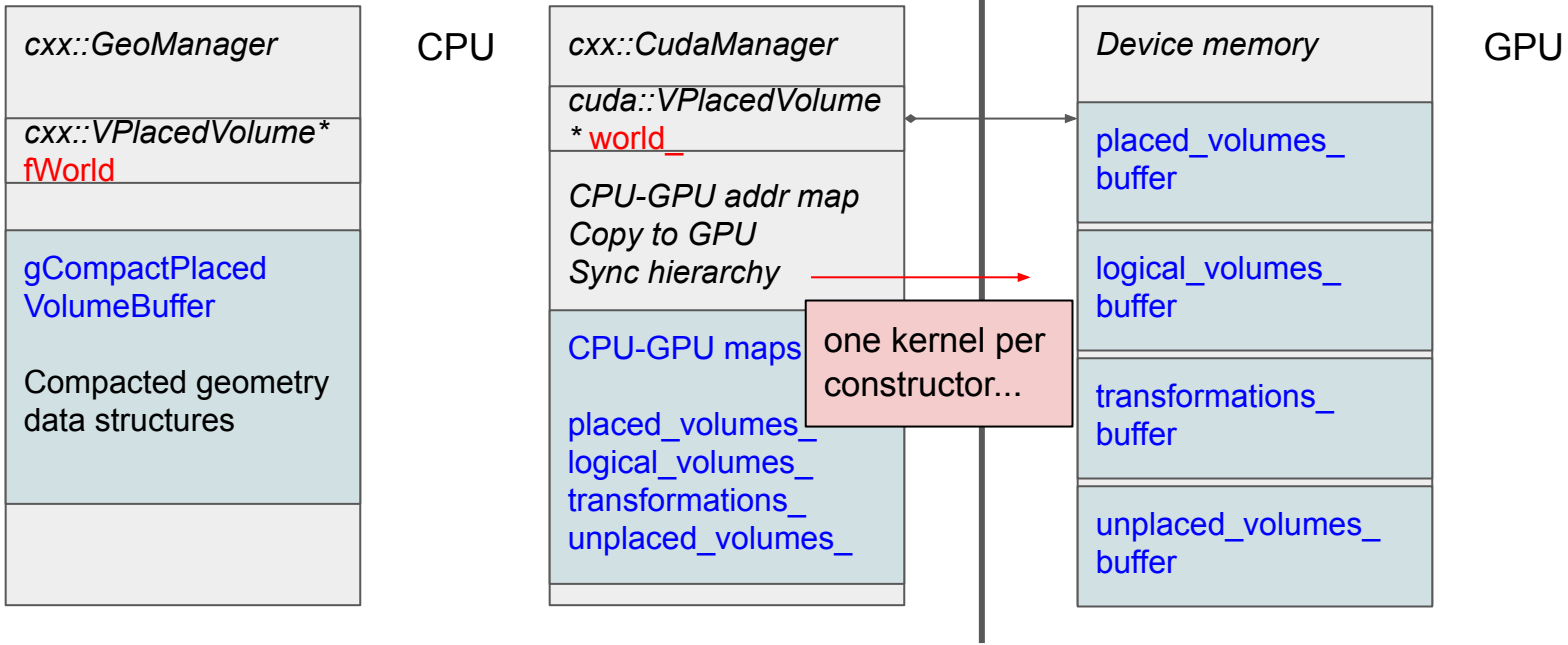


G. Amadio, SFT R&D meeting, 23 Mar 2021

CudaManager improvements

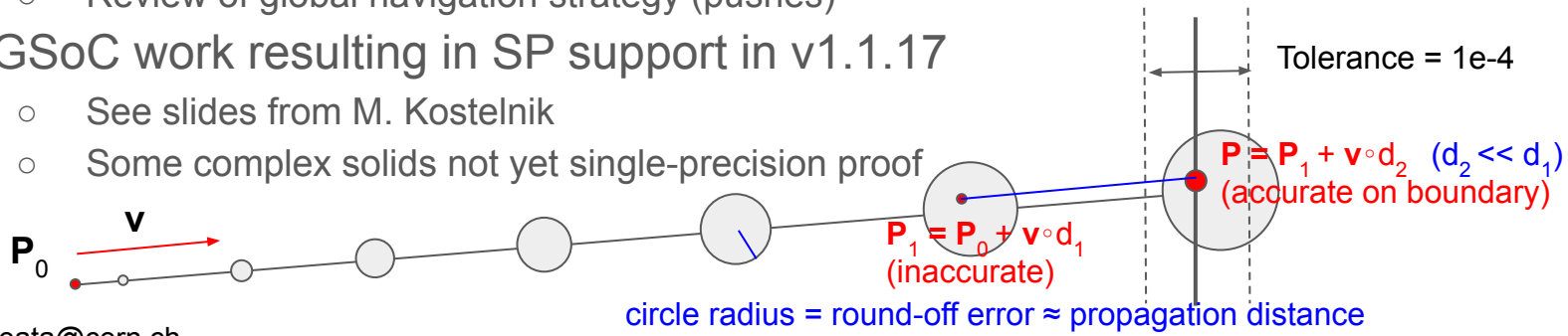
now bulk copying all placed volumes and transformations gives ~10x

S. Hageboeck



Single-precision VecGeom

- Goal: make VecGeom a single-precision library on-demand and understand implications on particle transport code (-DSINGLE_PRECISION=ON)
- Tedious work of fixing boundary tolerance algorithm inconsistencies
  - In particular reducing propagation-related rounding errors
    - Propagating from an inaccurate point close to boundary better than propagating from an accurate point far away -> approach solid first
  - Thorough review of *ShapeTester* as main tool to detect solid errors + shape fixes
  - Review of global navigation strategy (pushes)
- GSoC work resulting in SP support in v1.1.17
  - See slides from M. Kostelnik
  - Some complex solids not yet single-precision proof

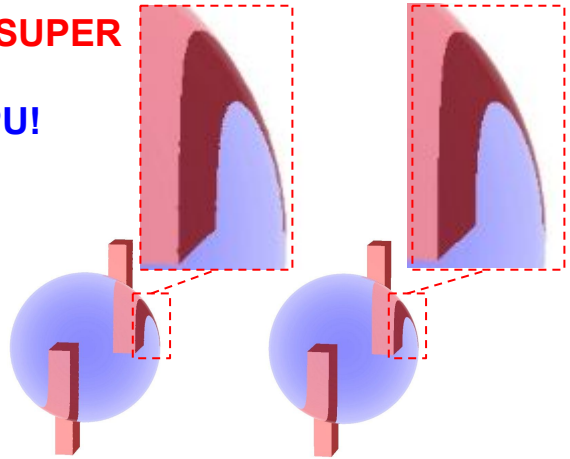


andrei.gheata@cern.ch

Performance impact of single-precision geometry

- Tested impact on performance in the AdePT examples
  - After doing several fixes for importing VecGeom precision type and using it in the navigators
    - LoopNavigator (simple looper for daughters), and BVHNavigator
- RaytraceBenchmark example (using BVHNavigator)
  - Reading a GDML file and modeling reflections/refractions and specularity
  - Validated by the output image
    - Very simple geometry: ~ 7.5% speedup
    - Complex geometry (trackML): ~ 44% GPU, ~ **13.6% CPU!**
- Physics-enabled GPU examples
  - Exa9 + trackML + LoopNavigator: ~ 2.8x speedup
  - Exa11 + trackML + BVHNavigator: ~ 30% speedup

RTX 2080 SUPER



andrei.gheata@cern.ch

GPU - what is missing?

- Not much functionality-wise - we can simulate CMS on GPU using VecGeom!
  - Some solids are not GPU-aware: tessellated, extruded, multi-union solids
- What about performance & portability?
  - Low device occupancy: extreme cases in AdePT using LoopNavigator show as low as 10%
  - Large thread divergence leading to serializing consecutive geometry calls to solids on GPU
  - Virtual function calls preventing compiler optimizations (for simple geometry setups)
    - But makes also VecGeom unfriendly to portability frameworks and non-CUDA compilers
- VecGeom GPU support was prototyped using CUDA exclusively
  - namespaces, compilation procedure, macro annotation
  - Preserving the CPU C++ object model and API rather than specializing for GPU
  - Navigation using layers of specialized helper classes, optimizing for CPU/SIMD but not really for the GPU use case

Current placed volume navigation helper layers

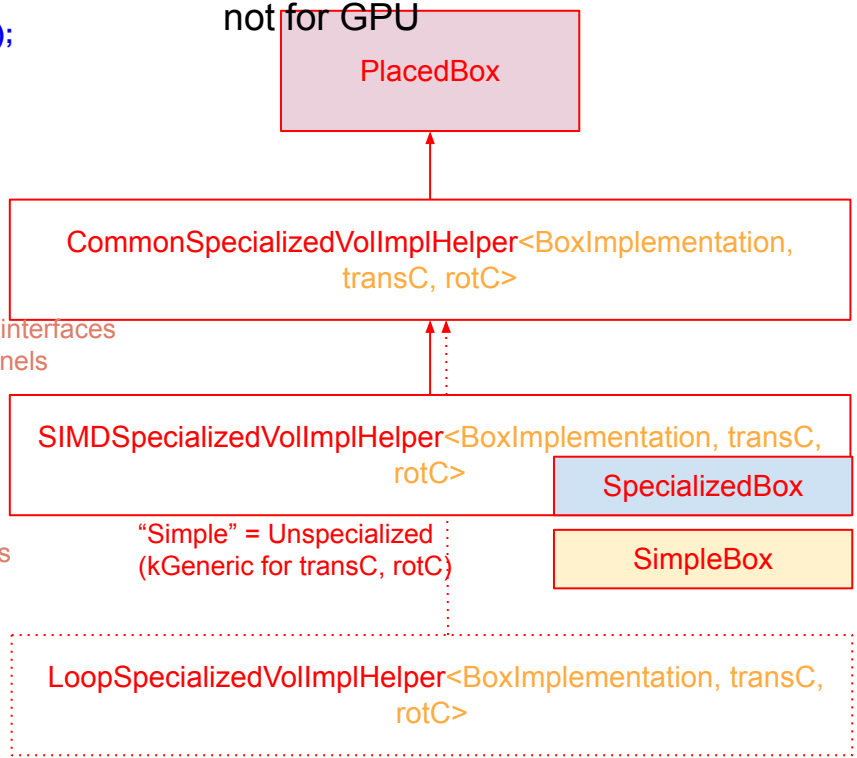
```
Transformation3D::Transform<transC, rotC, Precision>(point);
Transformation3D::TransformDirection<rotC, Precision>(direction);

// Dispatch scalar interfaces to the implementation kernels
Contains() override
Inside() override
DistanceToIn() override
PlacedDistanceToOut(Vec)() override
SafetyToIn(Vec)() override

// Implementation kernels aggregating SIMD types from SOA3D vector interfaces
// Dispatch head as SIMD and tail as scalar to the implementation kernels
Contains() override
Inside() override
DistanceToIn()(point,dir) override
SafetyToIn() override

// Implementation kernels dispatching SOA3D vector interfaces as loops
// to implementation kernels not supporting SIMD (e.g. to
BooleanImplementation)
Contains() override
Inside() override
DistanceToIn()(point,dir) override
SafetyToIn() override
```

complexity proving useful for CPU, but not for GPU



andrei.gheata@cern.ch

Virtual dispatching problem...

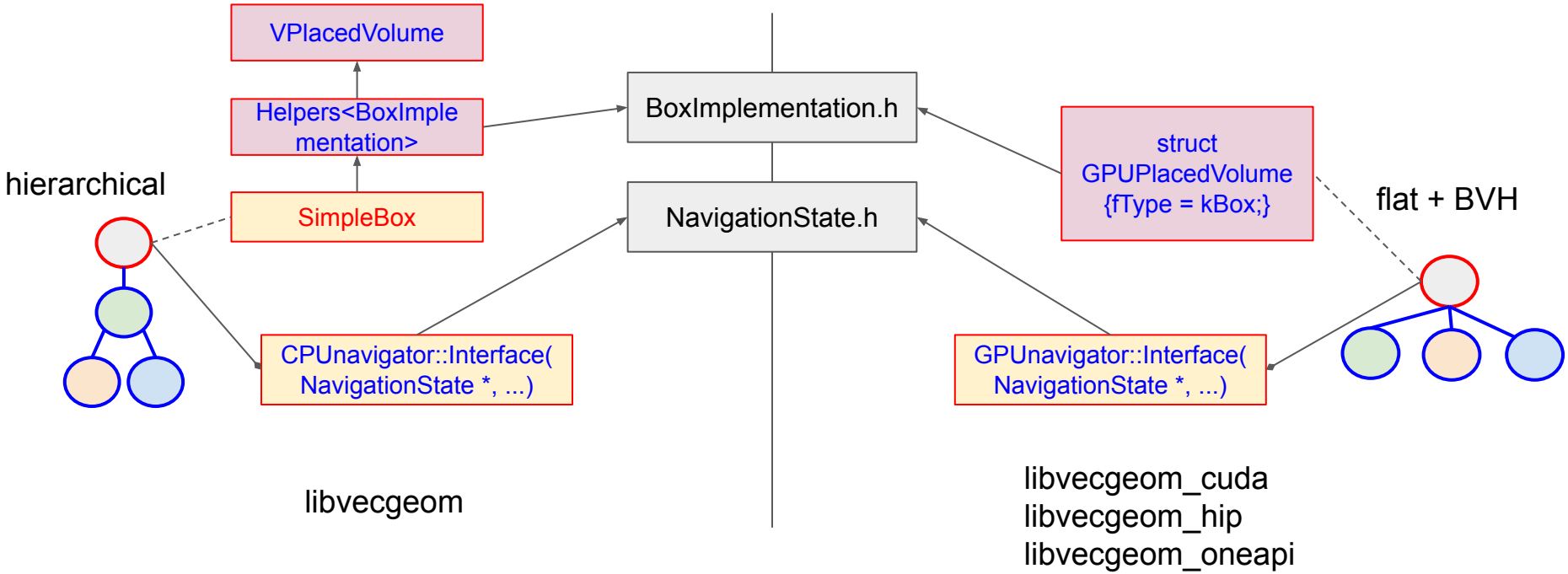
- Virtual calls, function pointers supported by CUDA
  - Not supported by SYCL, HIP, portability libraries -> blocker
  - Much worse for performance for GPU than for CPU
- CSG geometry is about polymorphism (primitive solids)
- Option 1: getting rid of virtual calls
  - Possible: thorough work by Jonas (switch statement dispatch) and investigation by Stephan (std::variant + std::visitor - limited support in CUDA)
  - Small performance improvement observed for low-complexity setups, degradation otherwise
- Option 2: getting rid of polymorphism
  - Transforming the GPU geometry to a small set of primitives (e.g. triangles, polyhedra)
  - Hard to imagine how this would scale for complex setups like CMS/ATLAS
- The dispatching problem is in general much simpler/specialized for a single level, but how can we implement this?

andrei.gheata@cern.ch

Towards specialized navigation helpers for GPU

- Solid-level navigation kernels already independent - good!
- VecGeom top-level navigation interfaces gives minimal insight to VecGeom types
  - The implementation can be specialized (with some care) w/o affecting the user
  - We can imagine different navigation implementation working for CPU/GPU, using the same underlying solid algorithms
    - Single namespace, different libraries, same user API and navigation state
- Separating the types and data structures used for CPU and GPU
  - And using different navigation data structures (e.g. SIMD optimizers for CPU and BVH on flat hierarchy on GPU)
- A preliminary step requires creating the GPU-specific navigation helpers
  - Transforming/simplifying the current multi-specialized layers

Separating CPU and GPU implementations



andrei.gheata@cern.ch

Outlook

- Significant progress in several different areas
  - build system and CUDA support, integration, persistency, navigation, single-precision
- Several obstacles on the path of performance & portability on accelerators
  - Reusing C++ types from host, CUDA entanglement, polymorphism
  - Work needed for making more specialized GPU navigation
- Strategy for simplifying the navigation helpers discussed already
  - Work to be done on the new design and implementation
- Integration work for using VecGeom navigation in Geant4 ongoing
  - See next talk
- Re-designing the accelerator support on GPUs is on the critical path for performance and portability

andrei.gheata@cern.ch