EPJ Web of Conferences **245**, 02024 (2020) https://doi.org/10.1051/epjconf/202024502024
CHEP 2019

A VecGeom navigator plugin for Geant4

Sandro Wenzel[1,*], *John* Apostolakis[1], and *Gabriele* Cosmo[1]

[1]CERN, Route de Meyrin, 1211 Geneva, Switzerland

Abstract. VecGeom is a geometry modeller library with hit-detection features as needed by particle detector simulation at the LHC and beyond. It was incubated by a Geant-R&D initiative and the motivation to combine the code of Geant4 and ROOT/TGeo into a single, better maintainable piece of software within the EU-AIDA program.
So far, VecGeom is mainly used by LHC experiments as a geometry primitive library called from Geant4, where it was shown to provide 7 – 12% reduction in CPU time due to its faster algorithms for complex primitives [1].
In this contribution, we discuss how VecGeom can be used as the navigating library in Geant4 in order to benefit from both its fast geometry primitives as well as its vectorised navigation module. We investigate whether this integration provides the speed improvements expected, in addition to the gain obtained from geometry primitives. We discuss and benchmark the application of a VecGeom-navigator plugin to Geant4 for a simplified geometry and show paths towards production usage.

1 Introduction

VecGeom [2] is a C++ geometry library for use in particle detector simulation. Its development started as part of the GeantV R&D initiative [3, 4] with the original goal to provide geometry kernels that are able to process multiple tracks/particles concurrently by utilizing vector micro-parallelism, implementing the single instruction - multiple data (SIMD) principle [2, 5]. Early in the project, it was realised that providing such a new library also provides a chance to review the existing geometry code used in HEP such as those provided by Geant4 [6] and ROOT/TGeo [7]. In this regard VecGeom joined forces with the USolids [8] project already following similar geometry code modernisation efforts. In addition to providing vector interfaces for GeantV, VecGeom also offers traditional functionality for the use in Geant4. In short, VecGeom provides improvements over other existing libraries in the form of: a) improved and revised algorithms, b) strong separation of state and algorithm for use in highly multithreaded environments, c) support for vectored input data, d) SIMD acceleration for vectored input data as well as e) SIMD acceleration of navigation functions for scalar input data in complex geometries [9] (including multi-union or tessellated solids [10]). Besides, the VecGeom code base also compiles on CUDA supported GPUs which might be of interest to those attempting to port detector simulation to GPU. The code base makes heavy use of template kernels and uses the VecCore library [11, 12] in order to reduce the amount of code for easier maintenance. Lastly, we would like to mention that VecGeom offers a geometry

*e-mail: sandro.wenzel@cern.ch

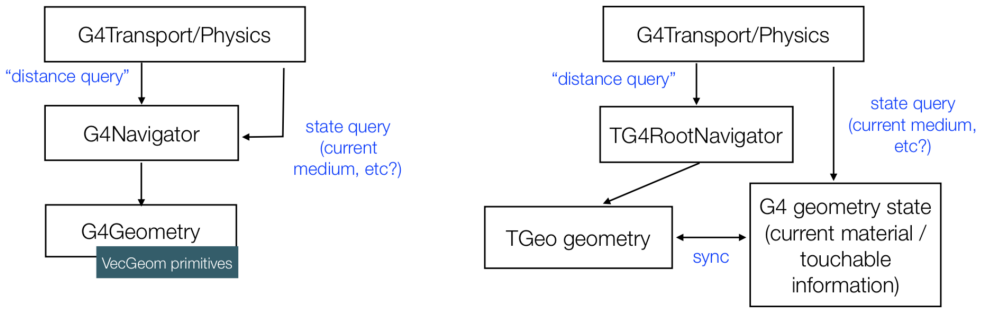

Figure 1. (Left:) Currently, Geant4 is able to dispatch to VecGeom only at the solid primitives level and the Geant4 navigator does not know about this integration. VecGeom solids are hidden behind facade classes looking like Geant4 native types. (Right:) Visualisation of the scheme in which TGeo was integrated into Geant4 [13] as part of the VMC project: A special navigator class was hooked into Geant4. Duplication of geometry and syncronisation of state is necessary due to different type system of Geant4 and TGeo.

primitives specialisation mechanism: Algorithms can be specialised for specific topologies of a shape primitive and a factory mechanism allows to create these specialised instances at run-time, without any intervention from the user.

The present work is concerned with increasing the level of integration of VecGeom inside Geant4. So far, with Geant4 v10.5, users can optionally use the VecGeom geometry primitives (solids), instead of the ones provided natively by Geant4. As visualised in Fig. 1(left), the integration of VecGeom solids is provided via special bridge/facade classes. This change is transparent to the Geant4 geometry transport engine (G4Navigator) and user code. This integration was tested by the CMS experiment at the LHC, showing significant positive impact ($7-12\%$) on the CPU performance [1]. However it exploits only a subset of VecGeom features. Notably, improvements provided in the VecGeom navigation system, such as SIMD accelerated hit-detection [9], are not yet utilised. This leaves room for further speedup of Geant4-based detector simulation which is one of the key motivations of the present work. In addition, some HEP experiments perform their Geant4 detector simulation through the Virtual Monte Carlo (VMC) system [13, 14], and in this case typically the ROOT/TGeo geometry is called by the Geant4 transport (see Fig. 1(right)). For this scenario there is currently no VecGeom integration available at all and overcoming this limitation constitutes a secondary goal of our investigations.

2 Review of the Navigation System

There are different ways to integrate VecGeom navigation in Geant4. One suitable path is provided by the architecture of Fig. 1(right), similar to the approach used for the integration of TGeo navigation in Geant4. Geant4 transport would talk to a special VecGeom aware navigator, implementing the necessary interfaces defined in the `G4Navigator` base class. In the following, we shortly review important aspects of the navigation system to explain why VecGeom is expected to give further benefits compared to the ones already used. In analogy to computer games (see Fig. 2), the navigator is responsible to determine the straight-line hit-location of a particle (banana) with objects (houses) present in the current material (sky) and the distance/step to this location. This is done by the interface called `ComputeStep`. Important other functions are estimating the isotropic distance `ComputeSafety` of the particle

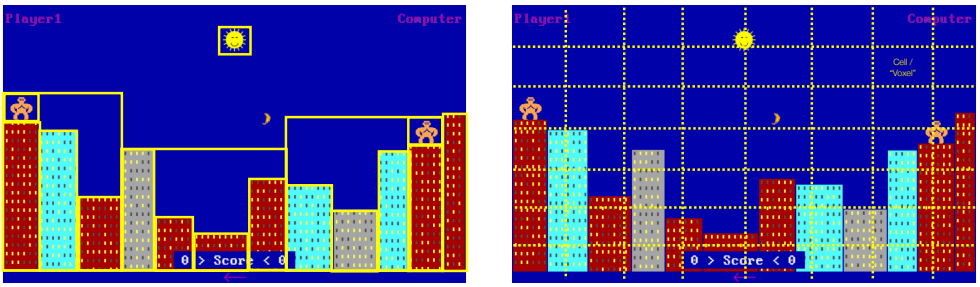

Figure 2. Visualisation of typical acceleration structures, using an analogy to a 2D computer game. (Left:) Hierarchy of bounding boxes, which can be constructed from the most elementary bounding boxes. (Right:) A simple cell/grid structure in which each cell can store information about its local neighourhood.

to any object in the scene and determining the current object in which the particle is contained, given just its global coordinates (`Locate`). There are significant overlaps with tasks in ray-tracing for instance.

In a brute force approach, these tasks get linearly more expensive with increasing the number of objects in the current scene. This is why typically some form of accelerating data structure, offering some form of in-memory lookup to beat linear search, is employed. Using it, one can quickly rule out most of the objects, and then perform calculations only for a remaining small number. One example of such data structures are simple grid-like space partitions where each partition (or cell or voxel) keeps a number of candidate objects. Another way involves structures constructed out of the bounding boxes of the objects contained in the scene. Many different variations exist and the data structures can be composed or combined at various levels to form a hierarchy. In a previous paper [9], we discussed a specific form of bounding volume hierarchy (BVH) that can offer SIMD accelerated queries, a technique also used by industry ray-tracers [15].

Different strategies are employed by the Geant4 native navigation system and the one in VecGeom. In case of Geant4, the same optimisation structure, up to 3 levels deep within each logical-volume, is employed for all the navigation tasks and the state of this structure is deeply coupled to the navigator. This structure was heuristically chosen as a good compromise and offers rather good performance, somewhat tunable by the user. VecGeom has a more flexible and modular approach, offers a wider set of structure implementations which can individually be selected per logical volume (per scene). The modularity brings the advantage of being extensible. We have already interfaced VecGeom to external libraries such as Intel Embree [15] which provides a very good performance for the most complex geometries (typically the case in ray-tracing tasks). Configurability allows to select separate optimisation structures for each task, potentially chosen according to the type of geometry. This way it can enable substantial additional tuning potential. Detailed tuning can be computationally expensive but it seems feasible to use GPUs for this task. Since detector geometries often evolve slowly, expensive structures could be precalculated and made persistent instead of recomputing them during each simulation run. In cases where geometries change often during a run, one can envision default optimisation structures, as in Geant4, which can be used if tuned ones are not available.

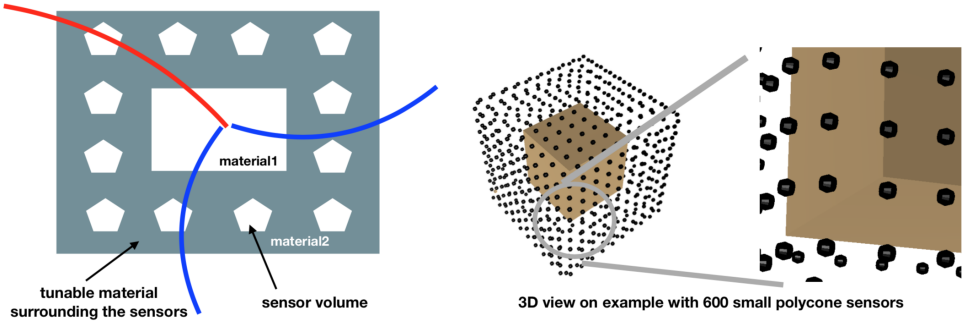

Figure 3. Layout of the geometry test setup (2D cut and 3D view). We use a simple layered geometry with configurable materials as a toy detector. One layer contains a variable number of sensor objects (type configurable). Particle collisions take place at the origin.

3 Implementation Status

In this work we aim for a prototype integration of VecGeom's navigation system into Geant4 with the goal to estimate the potential CPU saving opportunities for HEP simulations. The target is to be able to run a generic Geant4 application on a user provided geometry taken from a GDML [16] file. The application should be able to switch between using the native Geant4 system, the TGeo navigation or the VecGeom navigation for comparison and benchmark reasons. We target creating the least possible user disturbance and minimal code changes inside Geant4. Both were achieved by providing VecGeom navigation as a sub-class of `G4Navigator`, which is already designed to be extensible and plugable. This approach follows what was done for TGeo in the context of VMC, providing a fully self-contained new navigator class `G4VecGeomNavigator` implementing all virtual functions of `G4Navigator`. In practice, it turned out to be easier to take a somewhat refined approach in at first instance. In this approach we make use of a mixed system which only partially overrides the G4Navigator:

1. The `ComputeSafety` function is always dispatched to VecGeom.

2. The `ComputeStep` and `Locate` functions are dispatched to VecGeom only when Geant4 uses the `G4VoxelNavigation` in its original implementation; this works by implementing a sub-class of it, which dispatches to VecGeom.

In this way, we reuse existing logic to handle the navigation state needed by the Geant4 engine. The caveat is that not all navigator queries coming from Geant4 transport are dispatched to VecGeom. However we expect that in typical HEP geometries a large majority of the most expensive queries are sent to VecGeom.

In order to drive and test the implementation, we have created a simple layered geometry test case as shown in Fig. 3. The simulation complexity of this toy detector geometry is tunable in distinct ways by either allowing to change the material composition (switching between showering and pure tracking limit), by changing the number of sensor volumes as well as by selecting their geometric form (tube, box, polycone, etc.). It is clear that this example does not reach the complexity of realistic LHC detectors. Yet we chose it as a reasonably challenging setup for validating our approach and getting first performance indications, while being simple enough to obtain robust navigation as it avoids complicated corner cases. Our

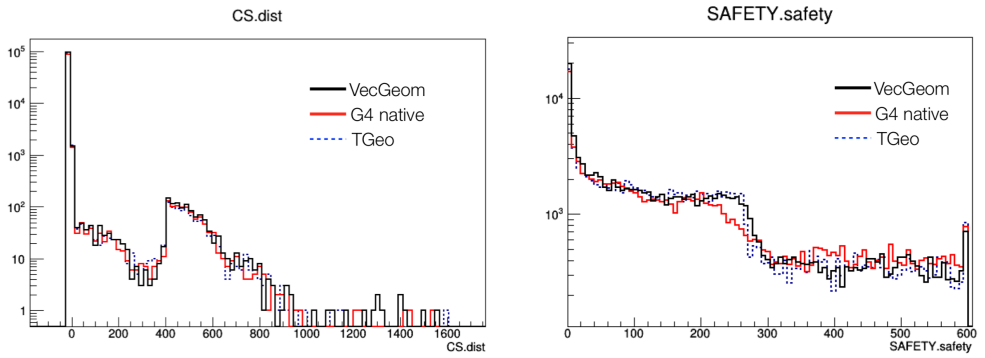

Figure 4. Distribution of step (left) and safety values (right) in mm returned to the Geant4 transport engine from the Geant4, TGeo and the new VecGeom navigator implementation. The simulation was asked to process a number of simple events in the test geometry with 600 sensors. All distributions are fully consistent and agree in the total number of entries and their form. Note that the curves for safety do not need to overlap strictly because safety may be underestimated.

validation is in the following way: we run the same simulation test (a modified application from VMC) using this geometry with either G4 native, TGeo or VecGeom navigation on a number of simple collisions emitted from a simple box generator (electrons only). The simulation is instrumented to record the number of function calls to the navigation system and to the geometry primitives. We also store and compare the values of returned distances and safeties from calls to `ComputeStep` and `ComputeSafety`, respectively. We deem the new navigator implementation as validated if the overall number of calls to the VecGeom geometry library and the value distributions are consistent with the two other reference cases. Figure 4 provides a validation example for the case of a geometry with 600 sensors made of polycone solids embedded in a low-density gas.

4 Performance Benefits

We now discuss the performance benefits of the prototype VecGeom navigator described in the previous section, using the same combination of geometry setup and generic application as in the validation step. We measure the total time spent in simulating a fixed number of events with charged primary particles for the various navigator implementations and as a function of increasing geometry complexity, which we define here as the square root of the total number of sensors. A uniform magnetic field is also enabled. For VecGeom we employ a shallow bounding volume hierarchy acceleration structure for the `ComputeStep` calls and a sparse but regular voxel structures for `Locate` and `ComputeSafety` functions which was pre-calculated before the simulation. In the case of `ComputeSafety`, specialised optimisation voxels store the list of the candidate objects for the safety search. There is a very fast (hash) mapping from a Cartesian coordinate to its voxel. In addition, VecGeom was configured with solid specialisation enabled; this allows it to dispatch to the best (template specialised) implementation of a (placed) solid. In summary, the benchmark should use most if not all of the features provided by VecGeom. When using Geant4 native navigation, two different configurations are tested: the one still using the native Geant4 solids (currently the default) and the one already using VecGeom geometry primitives. In addition, the following technical specifications are used here: Benchmarks are run on an Intel Xeon CPU E5-2697

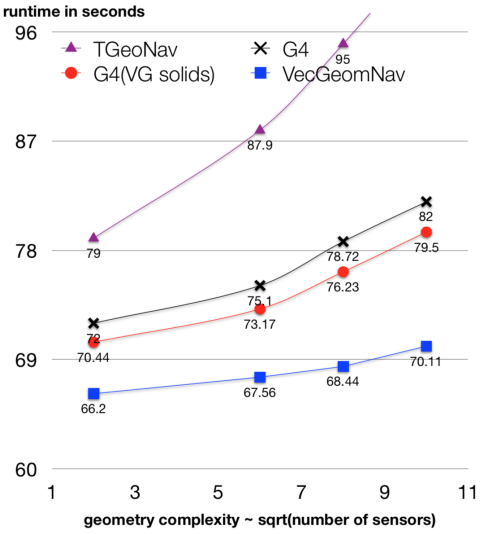

Figure 5. Benchmark of Geant4 simulation in a large volume of thin gas filled with a varying number of polycones, using different navigation options: Geant4 navigator with native solides (G4), Geant4 Navigator with VecGeom solids "G4 (VG Solids)", TGeo Navigator ("TGeoNav") and the VecGeom Navigator ("VecGeomNav"). The x-axis is the square root of the number of volumes. Error bars are on the order of the size of the symbols but are not included here.

v2 - Ubuntu 18.04 - server with task pinning and timings are the mean of 5 runs. GCC v7.2 is used as the compiler and the Geant4 version is 10.5.1.

Benchmark timings obtained when simulating 100K primary electrons in a low-density material limit, in which physics processes are reduced, are shown in Fig. 5 and they give rise to the following observations:

1. Navigation with the original Geant4 navigator benefits from using VecGeom solids in the few percent range, in line with previous observations [1].

2. The newly developed VecGeom navigator provides an additional speedup of about 6% – 13%, depending on the complexity. The overall scaling (as a function of geometry complexity) appears to be improved with the VecGeom navigator.

3. There is a clear indication of even larger benefits with respect to the TGeo navigator, which should be of interest to the VMC community.

We come to the same conclusions when repeating this benchmark in a dense material limit, where physics processes are enhanced leading to production of many secondary particles. Overall, these first results are encouraging and should justify an effort to go beyond a prototype stage.

5 Further Roadmap

Given these positive indications, there are some essential steps towards putting the system into production. The immediate next steps should focus on testing the new navigator on realistic detectors, coming up with a reliable and user-friendly workflow to calculate the

acceleration structures, as well as iteratively pushing the boundaries of the integration. In addition, we plan to work on an extension of the scheme to provide the VecGeom navigator also to the VMC/TGeo community as this promises considerable CPU time savings. The path towards this is straightforward and can be achieved by a slight variation of the scheme discussed here. On a different note, it will be interesting to incorporate GPUs/accelerators to speed up the initialisation of the in-memory acceleration structures. Typically, this task can be highly parallelised and should be ideal for such architectures.

6 Summary

We presented a first implementation of a Geant4 geometry navigator that is interfacing the VecGeom library at a considerably higher level than previously available. We demonstrated in a prototype example that, due to this interfacing, the CPU time for detector simulation can be reduced. This should be an interesting development for the ever increasing computing needs for HEP experiments.

We would like to thank the VecGeom development team for their continued effort and contributions to VecGeom, which provided the basis for the work presented here.

References

[1] K. Pedro, EPJ Web Conf. **214**, 02036 (2019)
[2] J. Apostolakis, R. Brun, F. Carminati, A. Gheata, S. Wenzel, Journal of Physics: Conference Series **513**, 052038 (2014)
[3] J. Apostolakis, M. Bandieramonte, G. Bitzes, R. Brun, P. Canal, F. Carminati, J.C.D.F. Licht, L. Duhem, V.D. Elvira, A. Gheata et al., Journal of Physics: Conference Series **608**, 012003 (2015)
[4] G. Amadio, A. Ananya, J. Apostolakis, A. Arora, M. Bandieramonte, A. Bhattacharyya, C. Bianchini, R. Brun, P. Canal, F. Carminati et al., Journal of Physics: Conference Series **762**, 012019 (2016)
[5] J. Apostolakis, M. Bandieramonte, G. Bitzes, R. Brun, P. Canal, F. Carminati, G. Cosmo, J.C.D.F. Licht, L. Duhem, V.D. Elvira et al., Journal of Physics: Conference Series **608**, 012023 (2015)
[6] S. Agostinelli, et al, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)
[7] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, P. Canal, D. Casadei, O. Couet, V. Fine et al., Computer Physics Communications **180**, 2499 (2009)
[8] M. Gayer, J. Apostolakis, G. Cosmo, A. Gheata, J.M. Guyader, T. Nikitina, Journal of Physics: Conference Series **396**, 052035 (2012)
[9] S. Wenzel, Y. Zhang, *Accelerating navigation in the VecGeom geometry modeller*, in *Journal of Physics: Conference Series* (2017), ISSN 17426596
[10] Apostolakis, John, Cosmo, Gabriele, Gheata, Andrei, Gheata, Mihaela, Sehgal, Raman, Wenzel, Sandro, EPJ Web Conf. **214**, 02025 (2019)
[11] G. Amadio, P. Canal, D. Piparo, S. Wenzel, Journal of Physics: Conference Series **1085**, 032034 (2018)
[12] *The veccore library*, `https://gitlab.cern.ch/VecGeom/VecCore`
[13] A. Gheata, M. Gheata, Journal of Physics: Conference Series **119**, 042014 (2008)

[14] I. Hřivnáčová, Journal of Physics: Conference Series **396**, 022024 (2012)
[15] *Embree - fast ray tracing kernels*, `https://embree.github.io/`
[16] R. Chytracek, J. Mccormick, W. Pokorski, G. Santin, IEEE Transactions on Nuclear Science **53**, 2892 (2006)