

VecGeom persistency using ROOT

Andrea Senacheribbe

Supervisors: Andrei Gheata, Witold Pokorski

Email: andrea.senacheribbe@gmail.com

Group: Experimental Physics - SoFTware Development for Experiments (EP-SFT)

Git branch: <https://gitlab.cern.ch/VecGeom/VecGeom/tree/asenache/vecgeom-493>

Keywords: VecGeom, perisistency, ROOT I/O, genreflex, dictionary generation

Summary

This project aims to create a persistency layer for VecGeom geometry objects, using ROOT I/O and genreflex tool. The final produced code can correctly save and load a simple VecGeom geometry with all available volumes and navigators.

Contents

1	Introduction	2
1.1	VecGeom	2
1.2	ROOT I/O	3
2	Objective	4
3	Procedure	4
4	Results	5
5	Conclusions	5
	References	6

1 Introduction

1.1 VecGeom

VecGeom (**V**ectorized **G**eometry) [1] is a detector geometry description library for high energy physics, written in C++. It is used both for simulation and reconstruction with tools like Geant4, GeantV and ROOT.

The purpose of geometry libraries is to keep track of all physical parts (volumes) of the detector and resolve geometry queries. Example of such queries can be to detect collisions of particles with shapes, determine if a particle is inside a shape or compute the minimum distance before collision.

VecGeom was designed from the beginning with code generalization (C++ templates), vectorization and GPU support in mind.

The use of C++ templates reduces classes duplication and allows a better maintainability of the code. Vectorization instead allows a faster and more efficient processing of the queries, since particles in the same logical volume are grouped together in vectors for calculation. In the code, the vectorization employs SIMD instruction on the CPU, through the library VecCore. Finally VecGeom can also run on GPU thanks to CUDA instructions support.

VecGeom can be used in scalar mode, with tools that don't support computation on multiple particles (like Geant 4) or in fully vectorized mode (for example with Geant V). A comparison between different geometry libraries is reported in fig. 1. VecGeom is faster than all the others, both in scalar mode (since better internal algorithms are implemented) and in vector mode thanks to SIMD computation [1]. Since geometry tasks use a large fraction of the CPU during particle simulations, these improvements can have a big impact.

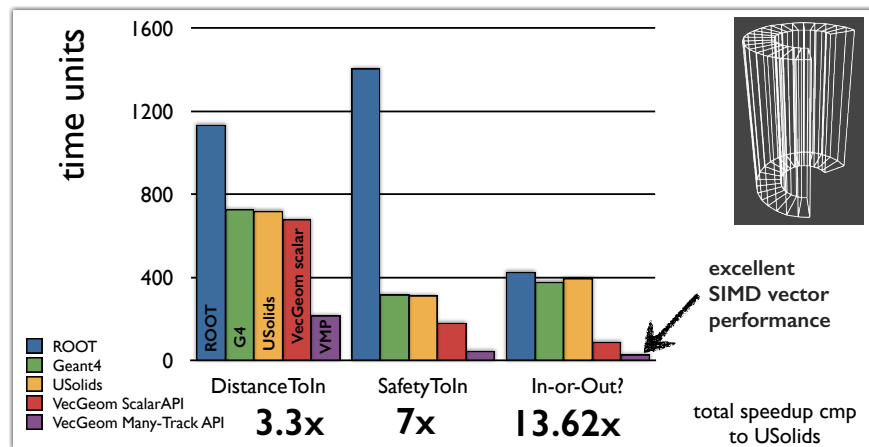


Figure 1: Comparison of VecGeom in scalar mode (Scalar API) and vector mode (Many-Track API) against other geometry libraries: ROOT TGeo, Geant4 and USolid (now merged in VecGeom)

Currently VecGeom (scalar mode) is used by CMS experiment at CERN with Geant4 (see fig. 2). Overall simulation speed-up is $\approx 7\%$ - 13% for the whole detector geometry [2].

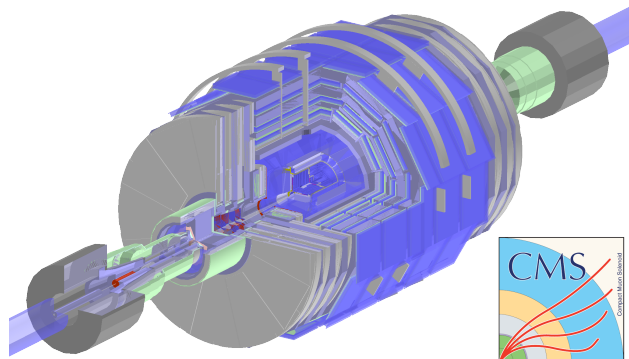


Figure 2: Part of the CMS detector represented in VecGeom

1.2 ROOT I/O

ROOT Data Analysis Framework [3] is a C++ framework for physics data processing used in HEP and developed at CERN.

Other than analysis and visualization, it can provide storage capability of data and C++ objects in compressed binary form (ROOT file). Arbitrary classes can be streamed from memory to file and then read back to memory, recreating exactly the same initial state. To do this, ROOT should be informed about the class structure (dictionary) and each class should expose a streamer method, able to stream all its data members.

ROOT provides reflection mechanisms for the automatic generation of dictionary and streamers, through the `genreflex` executable [4]. The tool requires a header file with all class definitions and a selection XML file with the information about which class and data member to save. It outputs then a C++ file with the dictionary and streamers to be compiled and linked with final executable.

2 Objective

The scope of this project is to persistify VecGeom geometry, using genreflex and ROOT I/O. The backend implementation (dictionary and streamers generation, proxies, read and write calls to ROOT) should be transparent to the user, who has to be provided with a simple API to save and load the geometry.

3 Procedure

At the beginning, I started by reading the documentation and checking examples about ROOT I/O and VecGeom. This first phase was necessary to get familiar to the software, the tools and the programming style. After compiling and installing VecGeom and ROOT, I created a CMAKE project for ROOT persistency in `persistence/root`. It is enabled by the flag `ROOT_PERSISTENCY` (OFF by default).

When everything was compiling and working correctly, I proceeded with the instrumentation of the basics unplaced volumes in the XML selection file. For classes where a public constructor with no arguments was missing, I had to add a special constructor (`TRootIOctor *`) that is called in the reading phase to recreate the objects. This constructor is preferable to the default one, since it cannot be called by mistake from the user.

To test the correct generation of the dictionary and the streamers, I wrote a test executable that saves the volumes to a ROOT file and reads them back.

After completing this first steps, I moved to placed and logical volumes. Since those classes are linked, I had to describe both together. Here I encountered some problems with `vecgeom::Vector` and I discovered alignment issues in vector mode compilation.

Instead of `std::vector`, VecGeom employs `vecgeom::Vector` as vector container. This created problems to ROOT I/O, since it doesn't know how to stream and properly call the container methods. The solution was to define an alias `vector_t` which is pointing to `std::vector` for standard build and to `vecgeom::Vector` for CUDA builds (since anyway ROOT persistency is not to be supported on GPU).

For what concerns the alignment problem, it is due to the fact that VecGeom classes change according to the architecture and the vectorisation backend used. The compilations flags (in particular microarch flags) used for VecGeom are not necessary the same used for compiling ROOT and this creates discrepancies between the real VecGeom classes and the one seen by ROOT. Moreover also inside ROOT the compilation flags are not passed consistently. To continue with the project, I temporary switch to VecGeom in scalar mode.

After having addressed these problems, my last steps were to finish with all remaining unplaced and placed volumes and to persistify the navigators. A proper interface for the streaming was also created inside `RootGeoManager`.

4 Results

The structure of the selection XML file is presented in listing 1. The classes are described with the `<class>` tag and by default all its data members are streamed. To specify rules for single data members, the `<field>` tag is used. For instance `comment="[fSize]"` annotates the size of a dynamic array, while `transient="true"` makes the data transient (not streamed to file).

Listing 1: Selection.xml

```
<class name="vecgeom::UnplacedMultiUnion"/>
<class name="vecgeom::MultiUnionStruct">
  <field name="fNeighbours" comment="[fNboxes]"/>
  <field name="fNneighbours" comment="[fNboxes]"/>
  <field name="fLast" transient="true"/>
  <field name="fBuffer" transient="true"/>
</class>
<class name="vecgeom::LoopUnplacedVolumeImplHelper<vecgeom::MultiUnionImplementation,vecgeom::VUnplacedVolume>"/>
<class name="vecgeom::CommonUnplacedVolumeImplHelper<vecgeom::MultiUnionImplementation,vecgeom::VUnplacedVolume>"/>
<class name="vecgeom::PlacedMultiUnion"/>
<class name="vecgeom::LoopSpecializedVolImplHelper<vecgeom::MultiUnionImplementation,-1,-1>"/>
<class name="vecgeom::CommonSpecializedVolImplHelper<vecgeom::MultiUnionImplementation,-1,-1>"/>
<class name="vecgeom::PlacedVolumeImplHelper<vecgeom::UnplacedMultiUnion,vecgeom::VPlacedVolume>"/>
```

Additionally, I wrote the interface which can be called by the user to stream the GeoManager with all the volumes and navigators used in the geometry.

Listing 2: API for streaming the geometry

```
bool RootGeoManager::Export(const char *filename); // write the geometry to filename
bool RootGeoManager::Import(const char *filename); // read the geometry from filename
```

Finally, the executable `RootPersistencyTest` checks the streaming of a complete geometry, while `RootPersistencyComponentsTest` streams and compares just individual components.

5 Conclusions

As future work, the alignment problem should be addressed for allowing persistency also in vectorized mode. A complete Xray track test can be developed to further test the correct streaming of the geometry. Finally a complete detector geometry (for example CMS, fig. 2) can be streamed and used as a real example.

I learned so many things working on this project: from geometry libraries and ROOT, to advanced C++ programming, git and how to collaborate to an open-source project.

It was a great opportunity to join the work of the SFT group, understand how research is conducted at CERN and which are its engineering challenges.

References

- [1] J Apostolakis et al. “Towards a high performance geometry library for particle-detector simulations”. In: *Journal of Physics: Conference Series*. Vol. 608. 1. IOP Publishing. 2015, p. 012023.
- [2] Mihaela Gheata. *A vectorization approach for multifaceted solids in VecGeom*. CHEP 2018 Conference, Sofia, Bulgaria. URL: <https://indico.cern.ch/event/587955/contributions/2937590/>.
- [3] *ROOT a Data analysis Framework*. URL: <https://root.cern.ch/>.
- [4] *genreflex in ROOT User Guide*. URL: <https://root.cern.ch/root/html/doc/guides/users-guide/ROOTUsersGuide.html#genreflex-a-comfortable-interface-to-rootcling>.