EPJ Web of Conferences **214**, 02025 (2019) https://doi.org/10.1051/epjconf/201921402025
CHEP 2018

A vectorization approach for multifaceted solids in VecGeom

John Apostolakis[1], *Gabriele* Cosmo[1], *Andrei* Gheata[1], *Mihaela* Gheata[1,2,*], *Raman* Sehgal[3], and *Sandro* Wenzel[1] for the VecGeom team

[1]European Organization for Nuclear Research (CERN), Switzerland
[2]Institute of Space Science (ISS), Bucharest
[3]Bhabha Atomic Research Centre (BARC), India

Abstract. VecGeom [1] is a multi-purpose geometry library targeting the optimisation of the 3D-solids' algorithms used extensively in particle transport and tracking applications. The implementations of these algorithms are templated on the input data type and are vectorised based on the VecCore [2] abstraction library in case of multiple inputs in a SIMD vector. This provides additional performance for applications supporting a multi-particle flow, such as the GeantV [3] prototype. VecGeom allows also scalar queries for all the supported solids, an option that started being used in Geant4 [4] since the release 10.2, as optional replacement of the geometry functionality provided by the native Geant4 solids. In single particle mode, VecGeom can still issue SIMD instructions by vectorizing the geometry algorithms featuring loops over internal data structures. This approach has proven to bring very large benefits for the tessellated solids represented in terms of triangular facets. To expose more vectorization in the scalar mode we have extended the approach used for the triangular tessellations to other multifaceted shapes, such as the extruded polygon, the polyhedra and different trapezoids. We hereby present the strategy used to vectorise the different processing phases for tessellated solids, the performance improvements compared to the previous scalar implementations for other solids using this approach, and how this is reflected in Geant4 simulations using VecGeom as geometry engine.

1 Introduction

Geometry modeling is at the core of many simulation environments, from building design and aeronautics to space technology or particle physics. Geometry introduces boundaries and constraints to a given problem, associating specific physical properties to objects with given shapes and extents. Some important geometry tasks in simulations, in particular for particle transport in physics experiments, is the classification of points and computation of ray-object intersections. In high-energy physics we typically refer to detector models composed of 3D primitives such as surfaces or solids, described by hierarchical data structures.

Simulating a particle traveling through such detectors involves a so-called "stepping" procedure that allows propagating the particle in small steps. All physics processes registered

*e-mail: mihaela.gheata@cern.ch

for the particle have stochastic nature and will propose random step lengths according their cross sections. The selected process having the minimum step length will be validated only if the step is not crossing a geometry volume boundary. The exit point of the particle trajectory from the current volume, as well as the identity of the next volume being entered have to be therefore accurately computed. These calculations represent the "navigation" functionality which is the most computationally-expensive feature of geometry modelers.

This functionality is implemented in scalar mode (estimation for a single track/ray per query) by the geometry packages of Geant4 and ROOT [5] in a similar manner. This motivated the development of a common navigation module, the USolids [6] unified solids geometry library. USolids' development started in 2010 in the framework of the AIDA [7] project.

In the early 2012, the GeantV R&D started exploring the idea of grouping many particles for the same processing phase - such as geometry, field propagation or physics models - to increase cache locality and create the conditions for SIMD vectorization on loops over these particles. This idea required the development and availability of vector APIs performing geometry calculation for multiple tracks in an efficient SIMD manner. This was the main idea that triggered the development of a new vector-aware geometry package, called VecGeom. In turn, having additional vector APIs next to the scalar ones in VecGeom has driven the development of the VecCore library as an abstraction layer in order to express algorithms that compile for scalar and vector types. The VecGeom project eventually merged with the USolids activity and integrated the algorithmic modernization started therein, further improving it and extending the development to modern C++ template paradigms while adding multi-track capabilities using vectors.

2 Vectorization

The main goal in the development of the VecGeom library was to provide optimized algorithms for navigation, both for 3D solids and global navigation queries, seeking improved performance compared to other existing implementations. Several vectorization techniques were explored, in particular the abstraction of the data types to allow computational kernels operating on both scalar and vector data in an architecture-independent way.

This approach allowed for important SIMD speed-up factors of the VecGeom multi-particle mode compared to the single particle mode, as reported in a previous publication [8]. The performance improvement in this approach decreases with the algorithm complexity in terms of branching and/or memory operations.

We decided to explore further vectorization opportunities to take advantage of the complex but repeated sub-structures of solids composed by planar facets. This mode can speed-up single particle queries by vectorizing the internal loop over the solid's faces, instead of parallelizing over multiple particle (see figure 1), extending the potential for speedup to classical scalar simulation clients, including Geant4. In this work, we describe the application of this idea to the tessellated solid, as well as multi-faceted solids and multi-unions.

3 Tessellated solids

The detailed simulation of complex surfaces is typical of certain applications, including: medical applications, applications sensitive to material budget or space applications for shielding studies. Such surfaces are usually designed from CAD applications and can be represented as closed meshes of connected facets of very simple types (typically triangles or quadrilaterals, see figure 2). There are several simple procedures for tessellating a given

Figure 1: VecGeom is exploring different vectorization paths. The throughput can be increased by operating on multiple input data with SIMD operations (left) or by executing faster single queries vectorizing on internal loops over faces (right).

surface, giving the main advantage of being able to tune the precision of the representation to the appropriate level required by a given application. This technique is frequently used in computer graphics and is very suitable for massive parallelism.

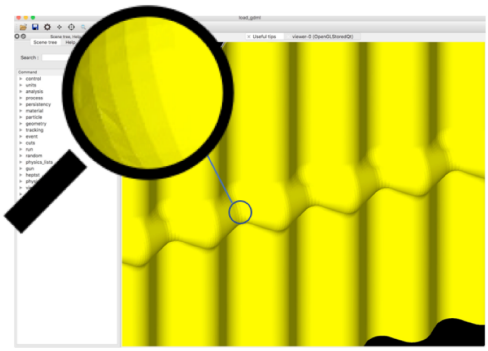


Figure 2. The LHCb RF-foil [9] loaded from GDML as tessellated solid (164k facets) and rendered in Geant4 using VecGeom.

A typical geometry modelling problem is finding if a 3D solid surface is hit by a ray (P, v) and at what distance from a starting point P along the direction vector v. For a solid tessellated by triangular tiles, the problem can be naïvely solved by checking the hit of each individual triangle and taking the minimum of the hit distances. If the computing time for each of the N triangles is t_0 then the trivial algorithm will scale like $N{\cdot}t_0$ while a vectorized version on a machine having the SIMD vector width W will scale like $N{\cdot}t_0/W$.

Using appropriate tree data structures, the problem can typically be reduced to a complexity $O(log(N))$. In order to benefit in addition from SIMD, the data need to be formatted as structures of arrays, easier to gather into the vector registers. Previously, we gained good experience with balanced bounding volume hierarchies (BVH) [10]. Such structures are also used by industry ray-tracing kernels (i.e. Intel Embree library [11]) or in other fields of computer graphics, and consist of wrapping geometry objects in bounding volumes (boxes) that form the leaf nodes of a tree. These nodes are then grouped as small sets (clusters) and enclosed within larger bounding volumes, in a recursive fashion, until ending up with a single top bounding volume enclosing the entire structure.

So far, we implemented a two-level hierarchy of such BVH, which is shown in the next section to work well for up to $O(10^5)$ components, but we do see scaling issues that call for a deeper hierarchy implementation in case of more detailed tessellations. In terms of data structure, the bounding box data is gathered in groups of size W_s (SIMD width in single precision). In the leaves of this bounding volume tree, we group triangular facets in clusters having the size W_d (SIMD width in double precision). The search algorithm is presented in

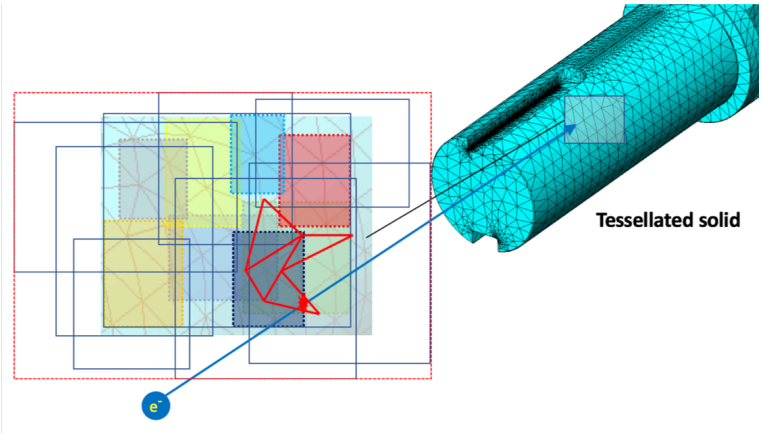


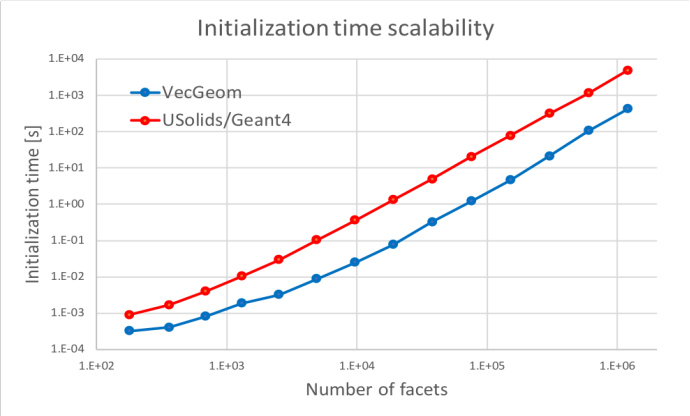
Figure 3: Algorithm for vectorizing the track-solid collision problem for tessellated solids in VecGeom. Top-level (red) bounding boxes are sorted by increasing track-box hit distance. Second-level (blue) hit bounding boxes are also sorted, then the contained clusters of triangles are queried in sequence. The algorithm stops at the first triangle being hit. All distance calculations are vectorized, using single precision for bounding box sorting and double precision for triangle selection.

figure 3 and performs in the following way: A single scalar collision query of a track with the entire BVH structure will first compute a sorted list (by distance) of the top-level bounding boxes being hit. This computation can be done in single precision because it is used only for selection purpose and have no impact on the final precision. For each box in this list, the query is propagated to the next level of boxes and produce a new sorted list, containing the selected leaf clusters of triangles. The distance computation to individual triangles is then vectorized in double precision for every cluster, in increasing distance order. The algorithm stops when the first triangle is being hit, since the sorting insures this will be also the closest triangle along the track. This algorithm highly reduces the number of single triangle distance calculations needed, while vectorizing all involved floating point operations with an optimized choice of precision.

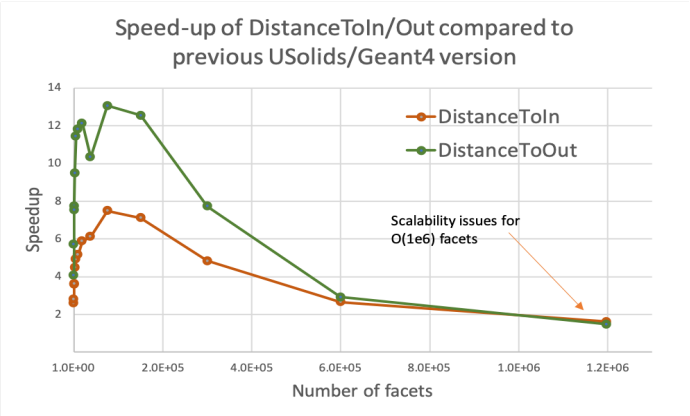
The performance of this implementation compared to the previous version in USolids is shown in figure 4. We observe a speed-up factor of up to one order of magnitude for both initialization - dominated by the clustering time - as well as for ray-solid collision detection. This performance improvement is consistent up to about 100K facets, covering most use cases for HEP geometry structures. Note that currently, for very large number of triangles, the scaling is worse in the two-level only BVH approach compared to the Geant4 optimization based on voxelization. However, we plan to use improved data structures in this regime in VecGeom as well. A generalization to arbitrarily deep BVH is, in fact, straightforward. Also, improvements are planned for the initialization time by using a clustering algorithm with better scaling.

4 Extensions

In a next step, we extended the tessellated solid optimizations to other solids that have multiple facets, specifically polyhedra, extruded solids and trapezoids. As an additional feature, these solids have one or more “sections” along the Z axis delimited by planes parallel to



(a) Initialization time.



(b) Speed-up for ray-solid collision detection from inside/outside points.

Figure 4: Performance of the tessellated solid implementation depending on the number of facets on Intel Ivy Bridge (AVX), compared to USolids/Geant4 implementation.

(XY), allowing for collision searches in a known, constrained range. We implemented a helper structure of a sequence of clusters of triangles, vectorizing on the cluster data structure as for the tessellated solid, but using sections with Z-limits rather than BVH optimizations.

We observed that for most simple solids (boxes, parallelepipeds, simple trapezoids) and for solids that were already vectorized (e.g. by the loops over facets in the previous implementation), there was no speedup (or only marginal) by using tessellated sections. On the other hand, the new implementation was considerably better for solids that were not previously vectorized, such as general trapezoids or extruded solids (figure 5).

Vectorizable BVH optimizations were applied also to multi-union solids. Unions are often used in constructive solid geometry (CSG) to represent complex assemblies, but also to describe cavities of holes made in compact bodies. These are classically represented as binary tree structures and exhibit complexity as $O(n)$ for point classification and collision problems. The USolids package implemented a "multi-union" construct, flattening the binary tree representation to a single level of nodes and optimizing the searches using voxelization. We improved on this approach by replacing voxelization with the same BVH approach used

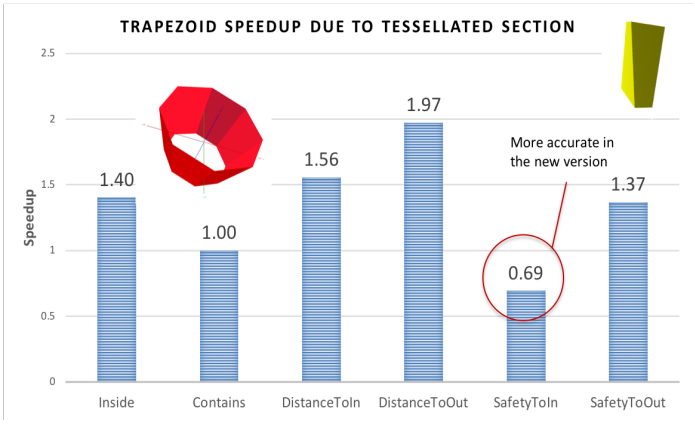


Figure 5: Performance of trapezoid navigation functions using the tessellated section helper compared to previous implementation on Intel Ivy Bridge (AVX). The slowdown in the new implementation for SafetyToIn is due to the more accurate computation of the isotropic safety value compared to the old implementation.

in tessellated solids, and benchmarked the scalability using an artificial setups composed of randomly positioned boxes.

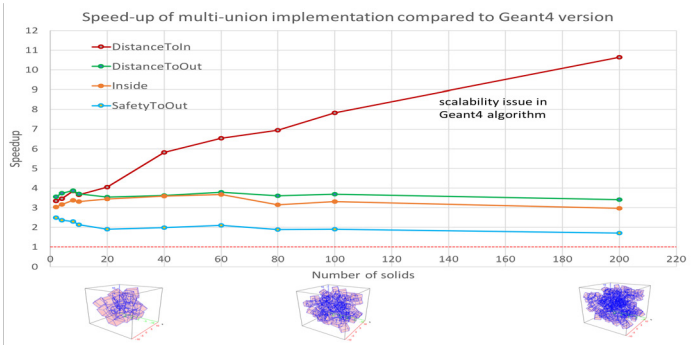
As shown in figure 6, we measured a speed-up between 2x and 4x for the new multi-union solid implementation compared to the USolids/Geant4 version, identifying also a scalability issue in the original implementation for the DistanceToIn function.

Trying to understand the benefits of migrating traditional CSG union solids to the new multi-union approach, we implemented a method to automatically flatten a CSG union into a multi-union. In this approach, performance benefits are becoming visible starting with a minimum of 4-5 components, when the BVH structure starts reducing the number of checked components.

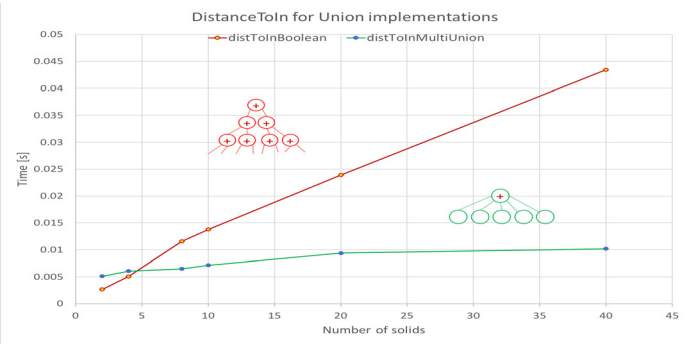
5 Conclusions

We investigated new vectorization opportunities for volume types not previously implemented in the VecGeom project. As VecGeom is used already in production as geometry backend for the Geant4 particle transport toolkit, and its use will increase, the single particle geometry optimizations are extremely relevant for the overall performance of the package. We developed an algorithm for handling multi-facet tessellations providing vectorization on two levels: a lower level cluster of triangles dispatching VecCore-driven SIMD instructions, and a higher-level BVH optimization structure minimizing the number of checked clusters. This algorithm shows clear benefits compared to the previous voxelization-based implementation in Geant4 up to $O(10^5)$ triangles. The scalability of the algorithm can be improved by using a deeper BVH structure. In this context, we are investigating to directly interfacing with other libraries such as Intel Embree.

We extended this vectorization approach to other multi-faceted solids, observing performance gains for the cases not vectorized in the previous implementations. We also applied the vectorized BVH approach to multi-union solids where we improved by a large factor the existing Geant4 algorithm, showing clear benefits when using the multi-union approach in place of the classical CSG union solids.



(a) Speed-up for different navigation functions compared to the corresponding implementation in USolids/Geant4.



(b) Performance increase after flattening traditional CSG union solids into VecGeom multi-union.

Figure 6: Performance of vectorized BVH-based implementation of multi-union solids on Intel Ivy Bridge (AVX).

References

[1] J. Apostolakis et al., *J.Phys.:Conf.Ser.* **608** (2015) 012023
[2] G. Amadio, P. Canal, D. Piparo and S.Wenzel, *J.Phys.:Conf.Ser.* **1085** (2018) 032034
[3] J. Apostolakis, R. Brun, F. Carminati and A. Gheata *J.Phys:Conf.Ser.*, **396** (2012) 022014
[4] J. Allison et al., *NIM* **A 835** (2016), 186-225
[5] I. Antcheva et al., *Comp. Phys. Comm.* Volume **180** Issue 12 (2009) 2499-2512
[6] M. Gayer et al., *J.Phys.: Conf.Ser.*, **396** (2012) 052029
[7] AIDA 2020 project http://cern.ch/aida2020
[8] S. Wenzel and Y. Zhang, *J.Phys.:Conf.Ser.* **898** (2017) 072032
[9] T. Bird, *arXiv*:**1410.0812** [physics.ins-det]
[10] T. Larsson and T. Akenine-Möller, *Computing & Graphics* **30**, Issue: 3 (2006) 450-459
[11] C. Benthin et al., *ACM Transactions on Graphics* **33** Issue 4 (2014) 143