Towards an automatic geometry conversion: `TGeoArbN` — a ROOT-compatible triangle mesh geometry implementation

Ben Salisbury[1,*], *Christoph* Schmidt[1], *Tobias* Stockmanns[2], and *Ulrike* Thoma[1]

[1]Helmholtz-Institut für Strahlen- und Kernphysik, University of Bonn, Germany
[2]Forschungszentrum Jülich GmbH, Germany

Abstract. `TGeoArbN` is developed as a tool to automate the process of converting Computer Aided Design (CAD) models into ROOT geometries for physics simulations using the Virtual Monte Carlo (VMC) package. ROOT's geometry package currently has a capability gap when it comes to handling arbitrary shapes. This, however, is central to automatically convert CAD models into simulatable ROOT geometries. To overcome this particular problem a ROOT-compatible geometry class called `TGeoArbN` was created to allow the use of triangle meshes in simulations using ROOT's geometry package in combination with VMC. These triangle meshes can then be used to represent complex CAD parts without any more work needed.

1 Introduction

For modern particle physics experiments the accuracy of simulations is of great importance. Simulations are omnipresent. They are relevant in the design phase, where simulations are required to study, for example, the best placing of unavoidable holding structures, or the impact acceptance issues might have on the later physics output. Simulations are also vital for the actual data taking, such as for the tuning of triggers. Furthermore, they are necessary for the development of reconstruction tools, such as leakage corrections. Finally, they enable acceptance studies, which are required to determine differential cross sections. One requirement to achieve highly accurate simulations is that the detector geometries used in simulation are as close to reality as possible. The reality itself should be represented by the Computer Aided Design (CAD) geometry. Hence, the simulation geometry should replicate CAD. However, CAD models cannot be used directly for particle physics simulations. Instead, CAD geometries have to be translated into the geometry description "language" used by the particle physics framework.

One way of setting up particle physics simulations is by using ROOT's [1] geometry package in combination with the Virtual Monte Carlo (VMC) [2] package. This combination allows to define the detector simulation application code independent of the used Monte Carlo transport engine [2], such as Geant4 [3], that is used to propagate the particles through the geometry. The ROOT geometry package offers a large set of different geometry primitive shapes, such as boxes, tubes, cones, and others. It further provides capabilities to combine these simple shapes to form more complex composite shapes (Constructive Solid Geometry) [4]. Such shapes are then stored as a chain of boolean composition operations. This way of

*e-mail: salisbury@hiskp.uni-bonn.de

representing complex shapes is substantially different to the way modern CAD applications represent shapes, which is done via a so-called boundary representation (BREP). Here any solid is represented as a set of surfaces. Tools exist to help to convert CAD models into ROOT geometries, such as the STEP-to-ROOT converter [5]. STEP, standing for "Standard for the Exchange of Product Data", is merely a file format in which CAD models are stored [6]. The converter attempts to analyze CAD shapes based on their surface numbers, surface types and angles. It maps shapes that it recognizes as primitive shapes (e.g. boxes, tubes, etc.) to the corresponding ROOT primitives. Only CAD models already designed with this conversion process in mind, i.e. all CAD shapes have to be made of primitive solids, lead to a successful conversion, otherwise this tool will not succeed to convert the whole model. This poses a large constraint on designing detector components, and it does not allow converting already existing CAD models not following this constraint. Complex CAD parts have to be translated manually into sets of ROOT primitives that approximate the given shape.

However, neither a manual conversion of complex CAD parts into sets of ROOT primitives nor forcing CAD users to only use ROOT-compatible shapes during the design phase are acceptable. Since CAD software can approximate the surface of shapes by the use of triangle meshes, also referred to as tessellation, a ROOT-compatible geometry class called `TGeoArbN` was developed to encapsulate these meshes and render them simulatable. While the tessellated representation of complex solids is present in the Geant4 framework through G4TessellatedSolid [7, 8], it is absent in ROOT. `TGeoArbN` seeks to fill that gap.

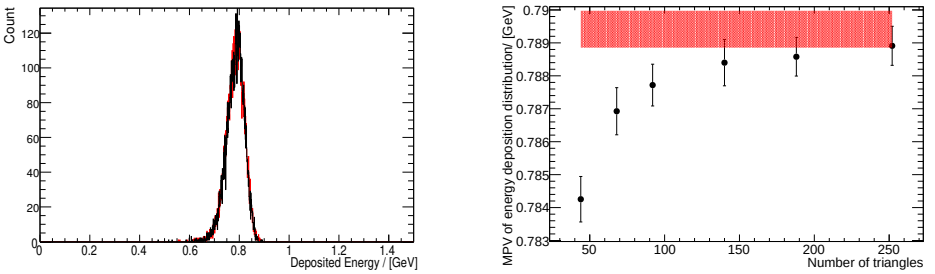
To automate the process of converting CAD models to ROOT geometries, `TGeoArbN` can then be used in combination with the STEP-to-ROOT converter. Primitive CAD shapes would be automatically converted to the appropriate ROOT primitives, while more complex CAD shapes are tessellated and represented by `TGeoArbN` in the ROOT geometry. This combination of `TGeoArbN` and the STEP-to-ROOT converter allows to further automate the process from converting CAD models to ROOT geometries. This automation then enables, for example, rapid iterative cycles of design and simulation to fine-tune a detector shape without constraining the CAD or compromising on the simulation geometry fidelity for conversion purposes.

2 The `TGeoArbN` class

`TGeoArbN` is a ROOT-compatible class representing a geometry as a triangle mesh. It implements the required navigation functions using ray casting techniques on the wrapped triangle mesh. These navigation functions consist of functionality to determine e.g. the distance from a given point along a given direction until the geometry volume boundary is reached, functionality to find the shortest distance between a given point and the geometry boundary, and whether a point is contained in the geometry volume or lies outside of it. The ray casting approach is generic and is applicable to implement these functions in combination with a triangle mesh. However, this generic approach comes at the cost of an increased simulation time. To mitigate the simulation time penalty, a partitioning structure can be used to decrease the simulation time by reducing the number of triangles needed to be tested against a ray. This will be discussed in 2.2.2.

2.1 Triangle mesh

Functionality to tessellate CAD parts into triangle meshes is widely available. Free CAD programs and some software libraries are available that allow to export surface approximations of CAD solids in form of meshes.



(a) (b)

Figure 1: Deposited energy of 1000 photons with an energy of 1 GeV shot at a lead tungstate cylinder with a lateral diameter of 2.54 cm and a longitudinal depth of 22 cm using a true cylinder and cylindrical meshes. (a): Deposited energy in the cylinder based on `TGeoTube` (red) and `TGeoArbN` (black). For the latter the cylinder surface was approximated by 252 triangles. (b): Most probable values (MPV) of the energy deposition distribution plotted against the triangle number of the cylinder triangle mesh. The red band indicates the one σ environment of the MPV for a cylinder represented by `TGeoTube`, while the data points represent results for different meshes simulated via `TGeoArbN`, with the data point indicating the MPV, and the error bars the one σ deviation.

One general problem to the approach of using triangle meshes is approximating smooth curves. Smooth curves cannot be exactly replicated using a set of triangles (or any other polygon). Therefore, a curved geometry represented by a mesh is only an approximation, but not an exact version of the desired geometry. However, the reproduced smoothness of the mesh can be modified via the tessellation parameters, such as the number and size of the triangles used. These parameters can be changed for most tessellation software, which allows for higher or lower geometric fidelity of curved shapes.

The achieved geometric fidelity will lead to differences in simulation results as shown in Fig. 1a. Fig. 1a depicts the energy deposition distribution of 1000 photons of 1 GeV energy in a lead tungstate cylinder, once represented using ROOT's `TGeoTube` primitive, and once by a cylinder mesh consisting of 252 triangles. Increasing the geometric fidelity of the triangle mesh (by increasing the number of triangles) leads to results that approach the results seen for the true cylinder represented using `TGeoTube`, as shown in Fig. 1b.

2.2 Inner workings

2.2.1 Ray casting

During the simulation, geometry objects need to be able to determine whether a given particle is currently inside or outside of their volume, or how far a particle can be propagated in a given direction before entering or leaving their volume. `TGeoArbN` uses a ray casting approach to implement these required navigation methods. That concept is also successfully used by G4TessellatedSolid [7, 8].

Since the triangle mesh represents the boundary of the described shape, computing the distance from a point to the shape's boundary is equal to finding the shortest distance between the point and the triangles. Determining the distance from a point to the next boundary along a direction vector is also done by finding the closest triangle in that direction. It is less clear how to check whether a spatial point lies inside or outside of the meshed shape. This requires

choosing a test ray, with a freely choosable direction vector starting at the test point. It is then tested if the ray enters or leaves the described object through the first crossed triangle, or if it does not hit a triangle at all (the point is outside of the object and the chosen direction vector points away from the object). Identifying whether a ray enters or leaves the object can be achieved by comparing the normal direction of the first intersected triangle to the ray direction. If their projection onto each other are antiparallel, the ray enters the object, if they are parallel, the ray leaves the object through the triangle. The ray-triangle intersection test used at the moment is a ray-plane intersection test, followed by testing that the intersection point (if existent) lies in the triangle. This could potentially be improved upon in the future using e.g. the Möller-Trumbore intersection algorithm [9].

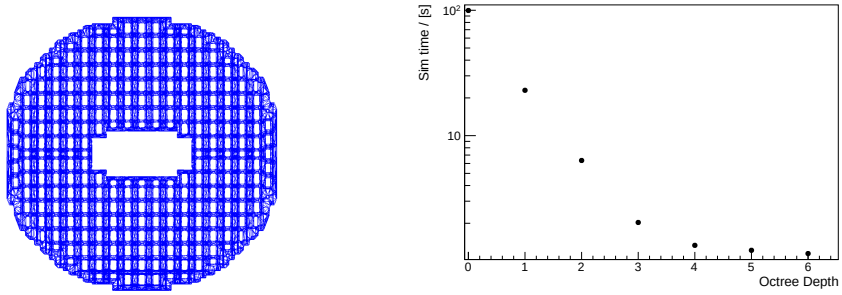
A downside of such a generic ray-casting approach is that for every navigation computation all triangles of the mesh need to be tested as it is not known without testing which triangles might intersect the test ray. Hence, the processing time will increase linearly with the increasing triangle number, since more triangles need to be tested. Indeed, meshes in the order of 100,000 triangles that cover a significant amount of the available solid angle range can quickly grow into simulation time sinks.

2.2.2 Partitioning structure

As processing resources are costly and limited, it is unfeasible to test every triangle in a mesh. However, only a small number of triangles is relevant to determine the relative location of a point or a ray. To reduce the number of triangles, which have to be tested, a partitioning structure can be used. This allows to select a subset of all triangles based on their proximity to a point or a ray. So far an Octree [10] was implemented to reduce the number of triangles that need to be considered to determine a point's or a ray's relative location. Furthermore, ROOT's geometry package has recently included a Bounding Volume Hierarchy (BVH) implementation [11]. This implementation has now also been used to implement a second option for a partitioning structure for `TGeoArbN`.

Octree

An Octree subdivides the bounding box of the triangle mesh into eight equally sized child boxes, which are referred to as octants. Each octant is itself split into eight octants, etc., depending on the Octree depth (the maximal number of recursion levels). Each octant is then assigned all triangles of the mesh that intersect the octant. When trying to determine the location of a point or ray only the triangles in the octant containing the point, or octants intersected by the ray, need to be considered. In this way the number of triangles, which need to be tested, can be reduced. This reduction of triangles comes, however, at the price of additional overhead of having to determine the relevant octants first. For meshes with low triangle numbers ($\lesssim$ 1000 triangles) this overhead can impose a significant increase in the required simulation time. However, the idea behind `TGeoArbN` is to handle more complex shapes, which goes hand in hand with a larger number of triangles. As test case, the mesh of the so-called Backplate of the Forward Endcap electromagnetic calorimeter of the $\overline{\text{P}}$ANDA experiment, depicted in Fig. 2a, would be such a high number triangle mesh. It is made up of well beyond 100.000 triangles. Depending on the chosen Octree depth, the simulation time of 10 photons of 1 GeV energy required for this geometry decreases substantially. With increasing Octree depth the simulation time decreases until it begins to level off at around an Octree depth of 4, as shown in Fig. 2b. Note that the Octree depth does not change the geometry, but merely changes the smallest octants size and the number of octants in which the triangles are grouped.



(a) (b)

Figure 2: (a): Mesh forming the backplate of the $\overline{\text{P}}$ANDA EMC Forward Endcap. The mesh consists of 277240 triangles. (b): Simulation time of just 10 1 GeV photons shot at the backplate assuming lead tungstate (to avoid changing the medium between the cylinder and Backplate simulation times)

Bounding Volume Hierarchy (BVH)

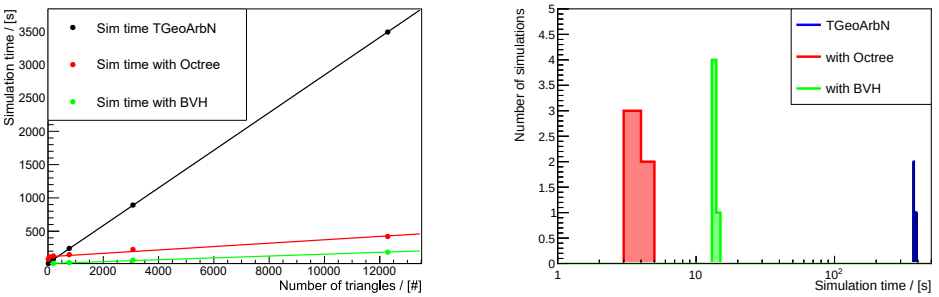
Unlike the Octree, a BVH does not partition the volume of the bounding box of the mesh, but groups triangles based on their proximity to each other. It acts like a binary tree. First, the overall box of the triangle mesh is used as root node. This node is then split into two nodes, for which two sub-boxes of the overall box are found that split the triangles in the parent box into two groups, again based on the proximity of the triangles to each other. This procedure is repeated until the leaf nodes only contain a single triangle each. Finding intersections of rays with triangles can then be done efficiently by top-down searches using the BVH. This allows to compute the distance in a given direction, or to determine if a point is inside the mesh, more efficiently than without a BVH.

2.2.3 Resource usage

Having these three options for using `TGeoArbN` (without any partitioning structure, with the Octree or the BVH) the question of how their resource consumption compares should be posed. For this evaluation, both simulation time and memory usage are considered.

Simulation times

The first simulation time measurement is done by using different meshes of a box-like lead tungstate crystal of 2.54 cm × 2.54 cm × 11 cm size. 1000 events of a photon of 1 GeV energy, shot at the center of the crystal's front face (2.54 cm × 2.54 cm) from the outside, are simulated using Geant4 as tracking engine. To investigate the dependence of the simulation time on the triangle count, the crystal's box mesh is artificially increased by splitting each triangle up into 4 smaller triangles. This splitting is done iteratively to gain meshes up to a size of 12288 triangles. The simulation times are then plotted against the triangle count in Fig. 3a. As can be seen, both partitioning structures yield substantial benefits for meshes beyond 1000 triangles. However, the BVH seems to outperform the Octree. Changing the mesh to represent a more complex shape, such as the Backplate, the Octree starts to outperform the BVH. This is illustrated in Fig. 3b, where the Octree decreases the simulation time down to $\sim \frac{1}{3}$ of the required simulation time of the BVH. It is reasonable that the Octree performs worse for the box-like mesh. The Safety distance, the distance from a point to the closest point of the



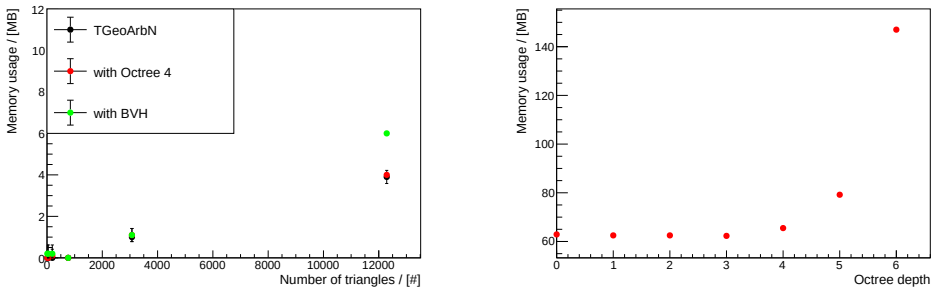
(a) (b)

Figure 3: Simulation times for different meshes and `TGeoArbN` usage modes. (a): Simulation times for 1000 1 GeV photons shot at a triangle mesh forming a box, acting as lead tungstate crystal. The triangle number of the mesh was artificially increased to cover a large range to see the simulation time behavior of `TGeoArbN`, using the BVH, Octree and without partitioning structure. (b): Simulation times for 1000 1 GeV photons shot a triangle mesh forming the backplate, using aluminum as material. The simulation times for `TGeoArbN`, as well as `TGeoArbN` with an Octree of depth 4 and the BVH are shown, using Geant4 as tracking engines. Each usage mode is simulated 5 times, yielding the 3 distinct distributions.

geometry surface, requires first to find the distance to a triangle which can be used as minimal search radius. All octants intersecting or inside of a sphere around the test point have to be found. The distance towards all triangles within those octants have then to be computed. As the photons are shot at the center of the 2.54 cm × 2.54 cm front face of the box-like crystal, the majority of steps happens along the center line of the box-like crystal. Hence, often the search radius will have to be 1.27 cm (half of 2.54 cm), for which then a large part of octants is contained in the search sphere. Therefore, a large number of the triangles still has to be tested, resulting in high simulation times. In case of the Backplate, however, one axis of the mesh is only 3 cm long, which is significantly shorter than the other two (with half-lengths of roughly 105 cm). Hence, the search radius will be 3 cm at most. In this case, this approach reduces the number of octants and triangles lying within the search sphere significantly. So, while both partitioning structures tend to be beneficial for larger meshes, tests so far indicate that it does depend on the mesh which partitioning structure is better.

Memory usage

The cost of using the partitioning structures is an increased memory footprint for `TGeoArbN`. This is depicted in Fig. 4. Fig. 4a shows the increase of the memory footprint of an interactive ROOT session after instantiating `TGeoArbN` and populating it with the different box-like crystal meshes, with the number of triangles given on the x-axis. One finds that, with increasing triangle number, the BVH, as it is currently used, requires the largest amount of memory. The Octree is used here with a depth of 4 layers, which hardly impacts the memory footprint of `TGeoArbN` compared to no partitioning structure at all. However, the Octree depth in general does change the footprint quite significantly for higher Octree depths as shown in Fig. 4b, now using the Backplate mesh as an example. With increasing Octree depth, especially for depths greater than 4, the increase in memory is significant. Hence, the default Octree depth is chosen as 4, which tends to be a good trade-off between simulation speed and memory usage (compare Fig. 2b and Fig. 4b).



(a) (b)

Figure 4: (a): Memory usage increases due to box meshes with a different number of triangles and different `TGeoArbN` usage modes. (b): Memory usage increases due to `TGeoArbN` using Octrees of different depths for the Backplate mesh.

3 Python STEP-to-ROOT converter

`TGeoArbN` was originally intended to be used in combination with the STEP-to-ROOT converter presented in [5]. Due to difficulties accessing and installing the required library dependencies, it was decided to implement a Python version of the STEP-to-ROOT converter instead. The Python implementation uses the pythonOCC package [12] to deal with the reading and tessellation of CAD parts and includes `TGeoArbN` as a fallback option whenever the mapping of [5] to ROOT primitives fails. Information on the mapping process from CAD solid to ROOT primitives is given in [5, 13]. The result of the conversion process is a ROOT script defining the geometry, as well as a set of meshes in the form of ASCII Stereolithography (STL) files, which were automatically created by the STEP-to-ROOT converter for the more complex CAD parts. The combination of `TGeoArbN` and the STEP-to-ROOT converter was successfully used to create a ROOT geometry for the $\overline{\text{P}}$ANDA Forward Endcap electromagnetic calorimeter. The Endcap is displayed in Figure 5, containing the sensitive parts, detailed passive material, such as the mechanical holding structure of the crystals, as well as the cooling pipes and insulation.

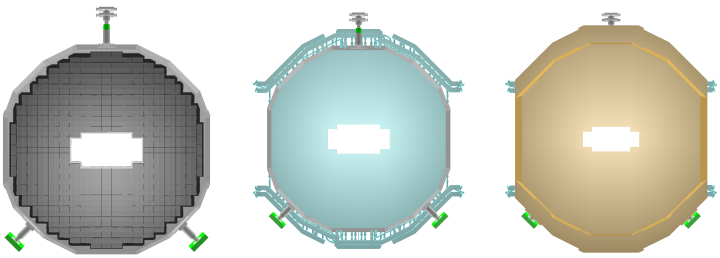


Figure 5: New ROOT geometry of the Forward Endcap EMC of the $\overline{\text{P}}$ANDA experiment, created using `TGeoArbN` in combination with the STEP-to-ROOT converter.

4 Summary

In general the strength of `TGeoArbN` as a ROOT-compatible tessellated geometry class is that it allows to further automate the process of translating CAD drawings into ROOT geometries without any manual adjusting of complex CAD solids. This can be helpful to quickly iterate

between CAD and simulation for e.g. acceptance studies during the design phase. It can also be used to produce geometries of high fidelity as baseline to test geometric approximations and their impact on simulation results. However, this highly flexible geometry representation comes with a higher processing cost during simulation. In order to minimize the simulation time, `TGeoArbN` can make use of an Octree or Bounding Volume Hierarchy. Both can significantly decrease the simulation time of objects represented by a triangle mesh with triangle counts greater than approximately 1000. `TGeoArbN` in combination with a STEP-to-ROOT converter has been successfully used in the $\overline{\text{P}}$ANDA experiment to transform the electromagnetic calorimeter Forward Endcap CAD drawings containing the full cooling and mechanical passive materials into a simulatable ROOT geometry.

This work was supported by the Federal Ministry of Education and Research (BMBF, Germany) and the programme "Netzwerke 2021", an initiative of the Ministry of Culture and Science of the State of Northrhine Westphalia (project "NRW-FAIR", ID: NW21-024-C).

References

[1] R. Brun, F. Rademakers, ROOT - An Object Oriented Data Analysis Framework, in *AIHENP'96 Workshop, Lausane* (1996), Vol. 389, pp. 81–86

[2] I. Hrivnacova, D. Adamova, V. Berejnoi, R. Brun, F. Carminati, A. Fasso, E. Futo, A. Gheata, I. Gonzalez Caballero, A. Morsch (ALICE), The Virtual Monte Carlo, eConf **C0303241**, THJT006 (2003), cs/0306005.

[3] S. Agostinelli et al., Geant4—a simulation toolkit, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003). https://doi.org/10.1016/S0168-9002(03)01368-8

[4] CERN, ROOT geometry package, https://root.cern.ch/root/html534/guides/users-guide/Geometry.html

[5] T. Stockmanns, STEP-to-ROOT – from CAD to monte carlo simulation, Journal of Physics: Conference Series **396**, 022050 (2012). 10.1088/1742-6596/396/2/022050

[6] ISO 10303-1:2024, https://www.iso.org/standard/83105.html

[7] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso, E. Bagli, A. Bagulya, S. Banerjee, G. Barrand et al., Recent developments in geant4, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **835**, 186 (2016). https://doi.org/10.1016/j.nima.2016.06.125

[8] P. Truscott, M. Gayer, https://github.com/Geant4/geant4/blob/master/source/geometry/solids/specific/include/G4TessellatedSolid.hh

[9] T. Möller, B.T. and, Fast, minimum storage ray-triangle intersection, Journal of Graphics Tools **2**, 21 (1997), https://doi.org/10.1080/10867651.1997.10487468. 10.1080/10867651.1997.10487468

[10] H. Samet, An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures, in *Theoretical Foundations of Computer Graphics and CAD*, edited by R.A. Earnshaw (Springer Berlin Heidelberg, Berlin, Heidelberg, 1988), pp. 51–68, ISBN 978-3-642-83539-1

[11] S. Wenzel, https://github.com/root-project/root/pull/16442

[12] T. Paviot, pythonOCC, Python package for 3D CAD/BIM/PLM/CAM (2022), https://doi.org/10.5281/zenodo.7471333

[13] T. Stockmanns, A STEP to ROOT converter for the FairRoot framework (2008-04-28), https://indico.cern.ch/event/30789/contributions/1676761/attachments/587383/808438/CADConverter.pdf