GPU-friendly boundary representation geometry model for simulation

Eduard George Stan and Andrei Gheaţă (supervisor)
SFT simulation R&D group, CERN, Geneva, Switzerland

Abstract
The SFT simulation R&D group is working on optimizing the performance of the Geant4 particle transport simulation toolkit. This includes porting the simulation software to accelerator hardware such as graphics processing units (GPUs), which often implies major rewrites of simulation components. One of the current projects targets developing a surface-based geometry model within the VecGeom library, to mitigate the geometry performance overhead observed in GPU simulations with the current 3D-solid modeling approach. The project is in a phase where the base required functionality is implemented, but several areas are still incomplete, in particular providing surface support for the full set of supported 3D-solid primitives. This project targets extending the set of supported 3D solids to be meshed, including the unit test coverage, and integration with the GPU particle transport framework.

Keywords
Simulation; R&D; Geometry; Surface Model; GPU multi-threading.

1 Introduction

In High Energy Physics, simulation is an inherently resource-intensive process, composed of three primary components:

1. Modeling Detector Geometry.
2. Modeling the Physical Processes: This involves simulating the physical phenomena related to radiation within the detectors.
3. User Code Simulating Detector Response.

Recent estimations of the computing resources needed for the high-luminosity LHC phase designate simulation as one of the major CPU consumers, as shown in figure 1. This shows the importance of doing R&D efforts to improve the efficiency of particle transport simulation.

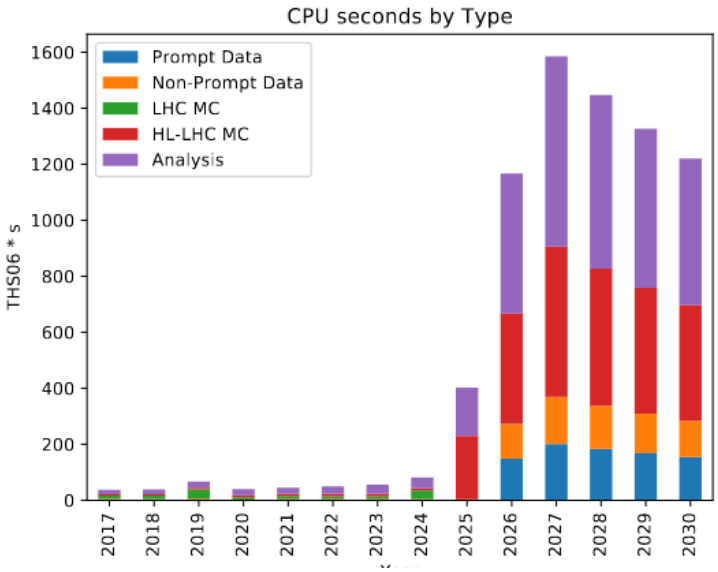

Figure 1: CMS estimated CPU resources required into the HL-LHC era, using the current computing model with parameters projected out for the next 12 years. [1]

Traditionally, high-energy physics (HEP) simulation primarily uses CPU resources, but with the growing availability of High-Performance Computing (HPC) resources, GPUs are becoming increasingly popular. There are important efforts made in the HEP community to make software stacks GPU-aware in the first phase, and then GPU-efficient. Simulation is not an exception, and there are a few ongoing R&D projects, such as AdePT [5] at CERN and CELERITAS [6] in the US. Both these projects integrate are using VecGeom [7] as a GPU-aware geometry library.

VecGeom is a standalone C++ library developed as a part of GeantV R&D initiative [3], that aims to separate the geometry component from the rest of the simulation. VecGeom 2.0.0-rc1, the latest version of VecGeom, makes use of CPU multithreading and several optimization techniques in order to boost computing time and improve memory usage.

However, because GPUs are more and more common, it becomes obvious that we should prioritize tools that take advantage of this resource. The initiative to develop efficient GPU-aware code led to a series of difficulties that require specialized solutions, different from what we have today.

Throughout this report, we will outline the current obstacles preventing us from utilizing the existing geometry model on GPUs. Our focus will be on the development of a new model capable of harnessing the full potential of this powerful resource.

2 Constraints of the current geometry model

Recent studies performed with AdePT library reveal that the current GPU implementation faces a significant bottleneck [2]. As we can see in figure 2, if we compare two geometry setups of very different levels of complexity, a simple sampling calorimeter with box layers versus the CMS setup, we observe similar compute and memory usage per streaming multiprocessor (SM), but there are many more stalled instructions in the complex case, mainly due to divergence of the threads in a warp.

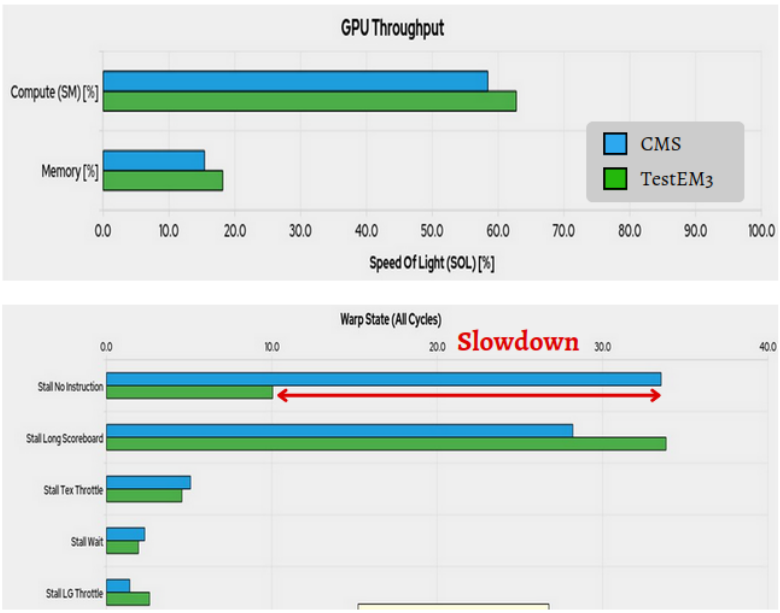

Figure 2: Estimated computing time on GPU for two geometry setups of very different levels of complexity. TestEM3 = sampling calorimeter 50 layers; CMS = full CMS 2018 geometry. [2]

Measurements demonstrate that, as the complexity of the geometry increases, the computing time grows rapidly. This discrepancy arises because, during multi-threading, simpler shapes are held back, waiting for more complex ones to complete the computation. A visual representation is shown in figure 3.

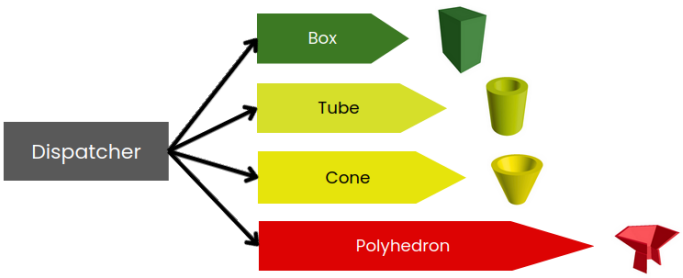

Figure 3: Computing time per volume in the current geometry model.

This bottleneck issue is what motivated the development of the surface model.

3 Surface model

A surface model was created to mitigate VecGeom divergence on GPU. It works by representing objects as a boolean operation of half-spaces, and it uses frames to represent each surface of the solid.

When it comes to multi-threading, surfaces with comparable complexities and similar computing time allow a shorter waiting time for synchronizing threads in GPU warps. A visual representation is shown in figure 4.

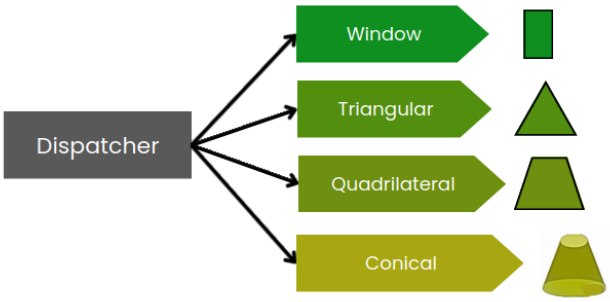

Figure 4: Computing time per surface in the surface model.

Three of the essential objectives of the surface model are:

- Transparent conversion from solids to surfaces.
- Transparent navigation through the surface model.
- Data structures fitting memory for complex cases.

The first objective will be further discussed in section 4.

4 Transparent conversion from solids to surfaces

The surface model aims to represent most of the 3D shapes present in LHC detector setups as Boolean operations between local surfaces, as shown in figure 5.

A local surface consists of three components:

1. Unplaced surface - specifies the type of surface and represents it in the simplest local system of reference. The unplaced surface can be planar, cylindrical, conical, etc.
2. Frame - specifies if the shape is a quadrilateral, a ring, or a triangle, and defines its vertices.
3. Transformation - is in fact a matrix of rotations and translations. The role of the transformation is to translate and rotate the surface from its original position, defined by the unplaced surface, to its appropriate position on the solid.

It is important to mention that a solid can be represented as Boolean operations between local surfaces that contain only an unplaced surface and a transformation. The reason frame is added is that it helps the optimization process for the navigator.

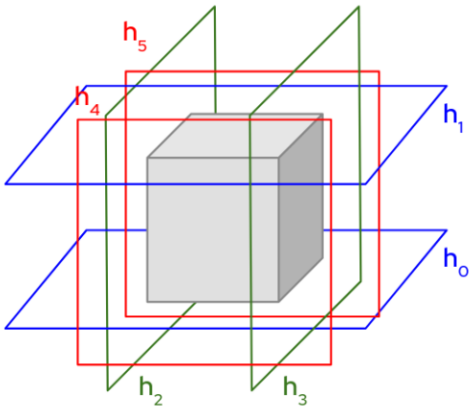

Figure 5: Solid represented as boolean operations between local surfaces. [2]

The current milestone regarding conversion from solids to surfaces is to simulate CMS 2018 geometry in AdePT for a realistic example. In order to reach this milestone, we need to successfully convert all shapes included in the mentioned setup.

4.1 Conversion of parallelepiped solid

If we want to represent a parallelepiped in the surface model, we need to compute its six faces as local surfaces and then execute boolean operations between them. Let's take this process step by step.

First of all, as mentioned in section 4, to create a local surface we need three components: unplaced surface, frame, and transformation.

Let's consider the surface at +x, represented in figure 6. In this case, the unplaced surface will be planar.

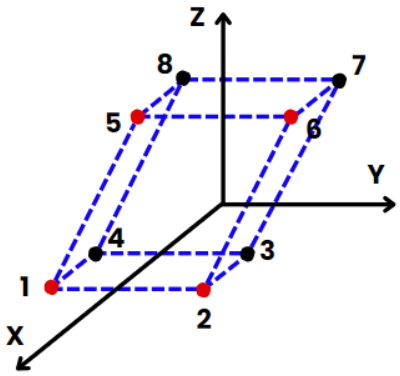

Figure 6: Parallelepiped in cartesian coordinates.

Next, the frame is a quadrilateral. Frame requires the vertices of the quadrilateral to be represented in 2D coordinates so that the equation of the shape will be shorter and easier to compute. In addition, the order in which the vertices are given as parameters for the quadrilateral frame creator defines the direction of the normal vector of the surface. By convention, the normal vector should point outside the solid. This is a useful feature for navigation to find out whether a particle is inside the solid or not.

Finally, we define the transformation, which will translate and rotate the shape from 2D, as it was defined, to 3D. A visual representation is shown in figure 7.

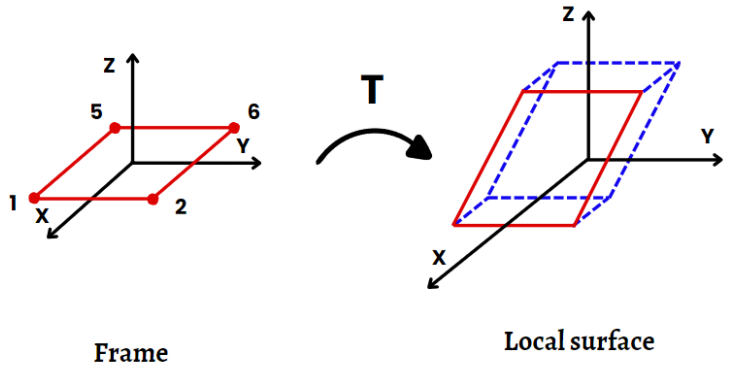

Figure 7: Transformation applied to the front face of parallelepiped.

After all 6 local surfaces are created using the same method, the only thing remaining is to compute the Boolean operations between them. In this case, we will use AND operators between each local surface: $0 \,\&\, 1 \,\&\, 2 \,\&\, 3 \,\&\, 4 \,\&\, 5$.

4.2 Conversion of simple extruded solid

For a more complex case, we consider a simple extruded solid. As we can see in figure 8, this particular solid can be divided into N quadrilaterals and 2 concave polygons, for the top and bottom surfaces.

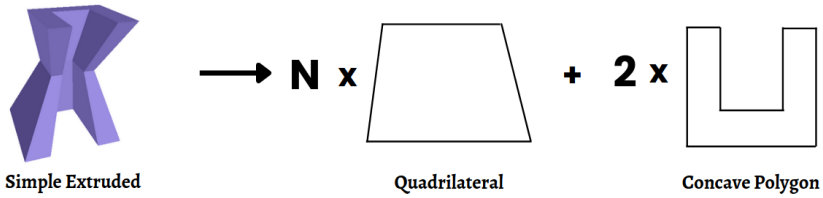

Figure 8: Simple extruded solid divided into its correspondent surfaces.

The problem here is that the surface model does not provide specific frames for concave polygons. We try to avoid creating very specialized frames but rather break complex ones into more simple representations, to have the least code divergence.

A good solution to this issue is dividing the polygon into simpler frames, such as triangles. The algorithm we decided to use is polygon triangulation by ear clipping [8]. An ear of a polygon is defined as three consecutive vertices V_{i-1}, V_i, and V_{i+1} that form a triangle, where V_i is a convex vertex. An essential property of the ear is that the triangle does not contain any other vertices of the polygon, except the ones creating the triangle.

After the algorithm detects an ear, it stores the constituent vertices of the triangle and then deletes vertex V_i from the list of vertices. It follows the same steps until there will be 3 vertices remaining, as illustrated in figure 9. The stored triangles will be used to create triangular frames, which will then be used for local surfaces.

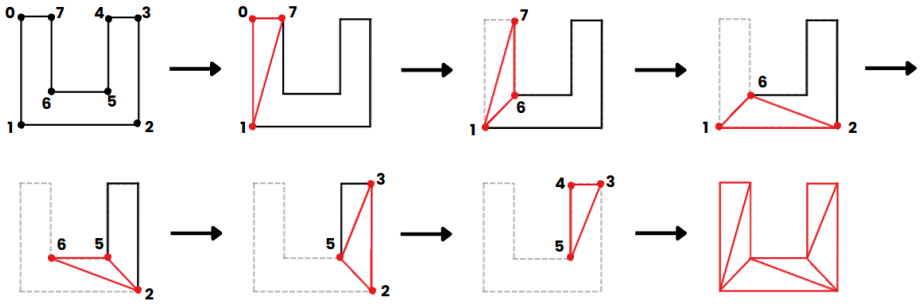

Figure 9: Separation of concave polygon into triangles.

5 Validation strategy

After building the solid converters to the surface model, we need a way to check their correctness. The testing method that we have chosen is comparing numerically the response of the solid model with the current 3D-solid one, for the same sample of input particles. A diagram for the validation workflow is shown in figure 10.

The validation program reads the geometry model data from a GDML file, which is a common Geant geometry representation format. The model is stored transiently using 3D solid primitives, making

volumes that are aggregated hierarchically to model the detector. A converter utility visits this hierarchy and generates the surface representation for each volume.

Subsequently, an arbitrary number of particles positions and directions are sampled inside the setup. These are used as input by different navigation utilities working on top of both the solid and surface models, locating the particles in the volume hierarchy, and computing the distance to the next hit volume. This navigation functionality is one of the computing-intensive parts of particle transport simulation, and it's the main optimization target of the surface approach.

A first validation can be done by enabling surface model verbosity, which prints specific data about the conversions, such as the transformations, surface types, and frames for each generated surface, as well as other model information useful for debugging. In addition, we display the run time for the conversion and navigation, useful for understanding performance features.

The last check is a numerical comparison between the solid and surface models for different navigation queries. We evaluate the distances between the crossing points where the particle hits the solid surfaces and validate only if they are numerically matching.

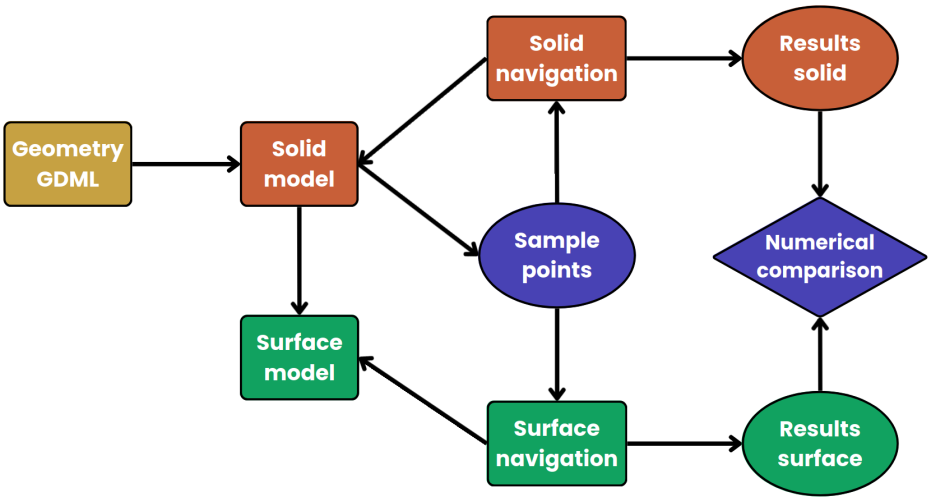

Figure 10: Validation strategy for surface model conversion.

6 Results

Figure 11 presents the solids that are already validated and the ones that are still in development.

First, the solids represented with red (Trd, Box, Composite Boolean, Tube, and Polyhedron) were added before CERN Summer School 2023 by SFT simulation and R&D group.

The solids represented with blue (Parallelepiped, Trapezoid, and Cone) were created and tested during CERN Summer School 2023 and they represent the main results of this report.

In addition, the solids represented with yellow (Simple Extruded, and Polycone) are in plain development. They also represent a great achievement for this report, because most of the implementation code was written, but some bugs still need to be solved.

Finally, another outcome of this report involves providing documentation for the solids present in figure 11. This documentation is intended to streamline the development process for project developers.

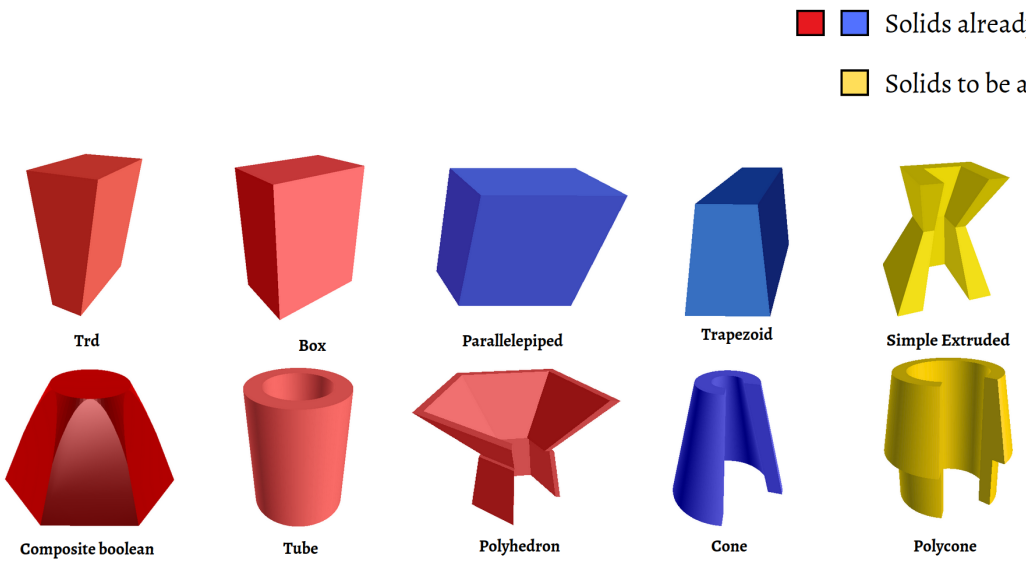

Figure 11: Solids in surface model.

Bibliography

[1] A Roadmap for HEP Software and Computing R&D for the 2020s: HSF-CWP-2017-01, December 15, 2017.
https://arxiv.org/pdf/1712.06982.pdf.

[2] Surface-based GPU-friendly geometry modeling for detector simulation, Andrei Gheata: 26th INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY & NUCLEAR PHYSICS (CHEP2023) - Norfolk, May 8-12, 2023.
https://docs.google.com/presentation/d/1ZGtAkT9pzMHfyDTde22mgPkC_HCPVMt7mLcAdE01JaY/edit#slide=id.p.

[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] The HEP Software Foundation, A Roadmap for HEP Software and Computing R&D for the 2020s, Computing and Software for Big Science, Mar 20, 2019. https://doi.org/10.1007/s41781-018-0018-8

[5] G Amadio and J Apostolakis and P Buncic and G Cosmo and D Dosaru and A Gheata and S Hageboeck and J Hahnfeld and M Hodgkinson and B Morgan and M Novak and A A Petre and W Pokorski and A Ribon and G A Stewart and P M Vila, Journal of Physics: Conference Series, 2438, 012055 (2023)

[6] Johnson, Seth and C. Tognini, Stefano and Canal, Philippe and Evans, Thomas and Jun, Soon and Lima, Jose Guilherme and Lund, Amanda and Pascuzzi, Vincent, EPJ Web of Conferences, 251, 03030 (2021)

[7] Apostolakis, John and others, Towards a high performance geometry library for particle-detector simulations, J. of Phys.: Conf. Ser., 608, 012023 (2015)

[8] David Eberly, Geometric Tools, Redmond WA 98052, Triangulation by Ear Clipping, Nov 18 (2002) https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf