EPJ Web of Conferences **295**, 03039 (2024) https://doi.org/10.1051/epjconf/202429503039
CHEP 2023

Surface-based GPU-friendly geometry modeling for detector simulation

John Apostolakis[1], *Dusan* Cvijetic[2], *Gabriele* Cosmo[1], *Andrei* Gheata[1,*], *Jonas* Hahnfeld[1], and *Eduard-George* Stan[3,4]

[1]CERN, Geneva
[2]EPFL, Lausanne
[3]IFIN-HH, Bucharest
[4]University of Bucharest

Abstract. In a context where the high-energy physics community strives to enhance software to handle increased data throughput, detector simulation is evolving to take advantage of new performance opportunities. Given the intricacy of particle transport simulation, recent advancements, such as adapting to accelerator hardware, require a significant research and development effort.
The feasibility of porting complex particle transport codes to GPUs has been already demonstrated by parallelizing the processing of tracks undergoing electromagnetic interactions. However, the GPU workflow presents distinct performance bottlenecks compared to the CPU workflow, owing to differences in parallelism models and hardware capabilities. Notably, the geometry component in this workflow has emerged as the primary bottleneck that needs addressing to enhance GPU-based simulations. The current CUDA-aware geometry is a 3D-solid modeler featuring substantial warp divergence, primarily due to the varying complexity of the supported 3D primitive solid shapes.
We present the outcomes of a one-year endeavor to create a bounded surface model that can seamlessly map the navigation capabilities of the existing solid-based geometry, with a focus on improving GPU efficiency. Our implementation aims to simplify low-level algorithms to reduce warp divergence, while simultaneously eliminating other sources of GPU inefficiency, such as recursive and virtual function calls, and ensuring code portability.

1 Introduction and motivation

Geometric modeling is one of the performance-critical components of particle transport simulation. The geometry API enables the transport engine to advance simulated particles in incremental steps while the material properties are constant and determine their new location as they traverse volume boundaries within the detector model. The geometry complexity highly increases the computational cost for geometry queries during this stepping, so the evolution of geometry modeling tools is a major concern in the context of future experiments embedding high-granularity detectors exposed to higher interaction rates [1].

Amongst the most popular high-energy physics simulation engines, Geant3 [2] and then Geant4 [3] developed and then improved a custom geometry modeler that demonstrated its

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

efficiency over a few generations of HEP experiments. The models are built from a large number of primitive 3D solid classes among which are elementary ones such as boxes and tubes, but also more complex polyhedra or extrusions, supporting Boolean operations. These solids provide basic geometry functionality to qualify inside versus outside and various distance computation algorithms. These components are grouped into more abstract and comprehensive entities referred to as "volumes," which also encompass associated material properties. Volumes are structured hierarchically, following strict containment and non-overlapping rules, which allows for optimizing global geometry queries such as locating a particle in the detector or finding distances.

Due to the evolution of computing technology, HEP computing needs to be able to exploit more and more heterogeneous resources. In this context, GPU-based solutions have been developed and put to work successfully in areas such as online processing and reconstruction. Due to the simulation's high complexity and stochastic nature, offloading particle transport on GPUs is still an R&D topic. Projects such as AdePT [4] and Celeritas [5] are now working on refactoring simulation workflow components that are in some cases sub-optimal on the GPU. Both projects are currently using the VecGeom [6] library as a GPU-aware geometry tool for complex detector modeling, observing independently that the largest fraction of the GPU simulation time is spent on geometry calculations. The current GPU port of VecGeom provides a CUDA implementation reusing the same C++ code as on the CPU, not optimized for the GPU use case. The code features virtual functions and recursive algorithms hard to optimize on the device. However, the main performance bottleneck is due to stalled instructions at warp level induced by thread divergence. The reason stands in the very diverse complexity of low-level geometry algorithms at the 3D solid level, which is an inherent characteristic of 3D solid modeling.

To mitigate the observed geometry bottleneck, the agreed solution was to develop a surface-based model targeting GPU geometry offloading. The core concept is reducing the complexity of low-level geometry algorithms while removing all virtual function calls and revisiting/rewriting the most performance-critical recursive algorithms. Adopting a boundary representation (BREP) model [7] allows decomposing navigation tasks into simple surface queries. After careful evaluation of the GPU-aware solutions available in the community, such as *ORANGE* [8] and *detray* [9], we decided to develop a custom surface model adapted to the Geant topological constraints and using a bounded surface representation. The model supports first and second-order half-spaces bounded by frames delimiting the faces of the 3D solids.

2 Design considerations

The design of the surface support implementation has been mainly driven by GPU-efficiency and portability considerations. However, the main pre-condition was to have an implementation transparent to user code, making the surface model an alternative view of the existing solid-based model. The users do not need to change their code, the surface model is created automatically from the transient solid model and then deployed on the device. A custom surface navigator implements the same interfaces for locating particles and computing distances as in the solid model case, dispatching the queries to the new model behind the scenes and hiding the implementation.

To allow for easier portability, the surface library was designed to be header-only, using C++ features supported by all GPU compilers. The defined types are simple structures accessible via indexing from a common data storage. The host and device hold different globally accessible copies of the surface storage. The surface model code is back-end independent

and free of virtual calls, so portability mainly implies copying the constant model data to the device in a back-end-dependent way.

To give the compiler more insight into the code, we were striving to remove all run-time recursions, discussed in detail in section 4. Another major GPU optimization target was to improve work balancing per geometry query while minimizing the number of stalls due to tails. This was the key factor in choosing a bounded versus unbounded surface model, for keeping the geometry query reduce problem less dependent on the original 3D solid complexity.

Another major consideration affecting the design was the ability to make use of the Geant model feature of non-overlapping containment. This reduces the number of visible surfaces from any given point inside the geometry setup to a single hierarchic level, compared to a flat surface model approach where all surfaces are visible from virtually any point in space. The touchable volume hierarchic state is cached by the surface representation and used during initialization for creating surface candidate lists for any state. Volumes can have joint surfaces, allowing the surface model to pre-calculate at construction time all possible transitions when crossing volume boundaries and simplify the volume relocation at run-time.

Last but not least, due to the performance impact of the floating point representation for many GPU architectures, we wanted to provide support for single-precision by using a precision parameter as a template for all surface model types. Several of the above requirements or performance considerations discarded the direct reuse of existing surface code, being that mesh representations or more domain-specific code such as ORANGE or *detray*. However, many of the features were inspired by the implementations of these models.

3 Model description

A common approach used in graphics is mesh modeling, approximating surfaces using points connected by lines creating planar polygons, often triangles. One of the advantages of this approach is that the ray-object distance calculation only needs to consider the triangles possible to be hit. Constructive solid geometry (CSG) surface models are also popular because 3D objects are often described as Boolean operations between simple half-spaces, having accurate mathematical representations. In this approach, geometrical calculations per 3D solid can be decomposed into a sequence of simple half-space equations. The boundary conditions are resolved by evaluating the complete Boolean expression, scaling with the object complexity (number of surfaces).

We have decided to take an approach that combines the strong points of meshing and CSG models, namely using half-spaces for surface representation to have an accurate description, and using frames having specific shapes to describe the limits of the solid faces on these half-spaces. Figure 1 shows that in this approach a cylindrical bar is decomposed into two planar half-spaces having ring frames, and one cylindrical half-space having a frame limiting the range on the z-axis. The ray-surface collision problem is solved in two steps: finding the crossing point with the half-space, and checking if the mask condition describing the inside part of the potentially hit frame is true.

The model currently supports planar, cylindrical, conical, and spherical half-spaces. All half-space types are "unplaced surfaces" in the sense that they are defined in a local reference frame where their equation has the simplest form. All the placed instances of the same surface type can be obtained by rotations and translations. For example, we use the (xOy) plane having the normal along the z-axis to represent the unplaced planar half-space, as shown in figure 2. The unplaced cylindrical and conical half-spaces have the z-axis as the axis of symmetry but take as additional parameters the different radii. In this approach, the plane half-space equation becomes as simple as $z < 0$, while the cylindrical half-space is defined

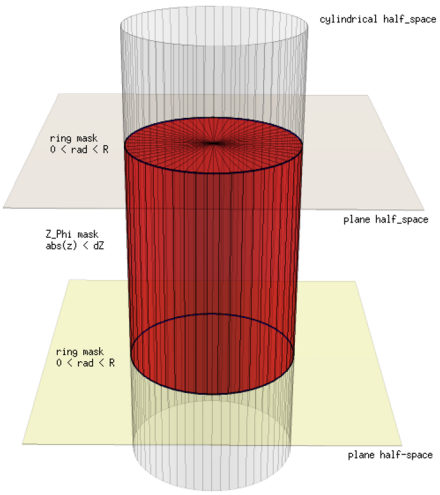

Figure 1. Half-space decomposition for a cylindrical bar.

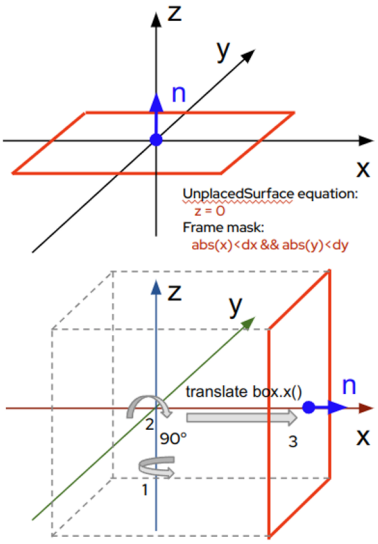

Figure 2. Sequence of transformations to convert from the local frame of a framed surface to one of the faces of a box.

as $x^2 + y^2 < R^2$. Unplaced surfaces can qualify space points as inside or outside, but can also compute the hit distance from a given point/direction. Queries involving placed surfaces are solved by first converting the input parameters (e.g. particle position and direction) to the unplaced surface local frame before using directly the unplaced surface functionality.

The frame types currently implemented are: windows, rings, triangles, quadrilaterals, and ranges in cylindrical coordinates. The main functionality provided by frames is point-on-face classification by masking the inside region, which allows fast exclusion of virtual intersections of rays with half-spaces outside the actual solid range. The frame data for a given geometry are stored in separate arrays per frame type, and referred to by index.

Associating an unplaced surface with a frame and a transformation matrix allows us to describe the faces of a solid in an arbitrary reference frame. In addition, we attach to this construct the index of the touchable volume in the local volume hierarchy, which uniquely identifies the 3D object. Framed surfaces are the basic building blocks of the model, as they provide the complete functionality required by navigation, in the same way as triangles can be used for ray-tracing in 3D graphics.

Extending a feature of the ORANGE model, we implemented the concept of a common surface, holding a set of framed surfaces on each side, on a joint unplaced surface support. Common surfaces are detected during model initialization, using a de-duplication algorithm based on hashing the surface placement transformation. A particle exiting a volume will have to hit one of that volume frame surfaces on the corresponding side, and one of the neighbor volume frames on the other side, as shown in figure 3. Locating the hit frame having the highest hierarchical depth for the crossing point defines the track relocation procedure for the surface model. As shown in section 4, this algorithm is much more efficient on GPU than the current one.

The model fully supports arbitrary Boolean solid composition. Boolean solids generate framed surfaces that can contribute to common surfaces, but they are flagged to trigger the

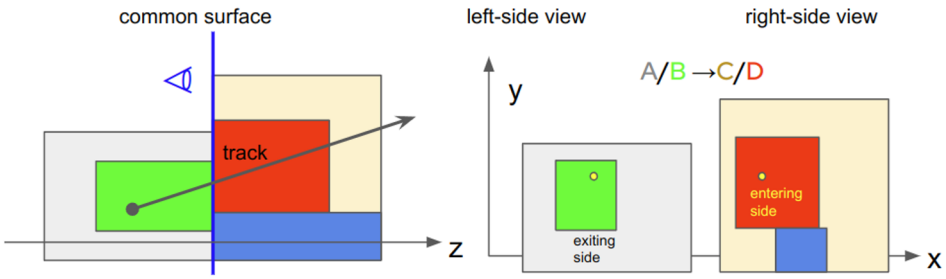

Figure 3. A common surface delimiting neighboring volumes in the new model.

evaluation of the complete CSG Boolean expression of the solid when such a surface is evaluated. The reason is that in this case, some of the framed surfaces may be masked out by the CSG expression, so only the complete expression evaluation represents the correct solution. We provide a GPU-friendly implementation for resolving CSG expressions by linearizing the search in a non-recursive way and allowing for Boolean expression short-circuiting by using infix evaluation for the surface predicates (distance/safety/location).

To make the model implementation transparent, the surface model is built automatically based on the transient 3D model representation. The model data is used by a custom navigator utility that provides the same geometry functionality as the solid-based model.

4 Performance evaluation

The main target of the project is to reduce the geometry overhead for the GPU simulation use case. This required making simpler and more balanced algorithms and giving the compiler more optimization opportunities. We adopted a header-based implementation free of virtual function calls and recursions for the performance-critical code.

4.1 Relocation after boundary crossing

One of the main bottlenecks to address was the geometry relocation after crossing volume boundaries. In the solid modeling approach, the algorithm has to visit recursively the volume tree, querying the corresponding 3D solid at each level. Once the topmost volume containing the particle position is found, the search has to be continued downwards the tree. The search scales with the volume content complexity, and stops once the deepest containing volume is found.

The new approach relies on common surfaces, transforming the tree search into a non-recursive linear loop over neighbor volume framed surfaces. This reduces the complexity from 3D to 2D, and from global volume content to just touching neighbors. The relocation performance becomes quasi-independent of the volume complexity, as shown in figure 4, which takes the example of a simple layered calorimeter with an increasing number of layers. BVH navigation optimizations have been disabled in this test to have a comparison on equal footing.

4.2 Boolean solids in the surface approach

The detector geometry description in the Geant approach allows for arbitrary CSG composition between supported 3D primitives, involving union, intersection, and subtraction op-

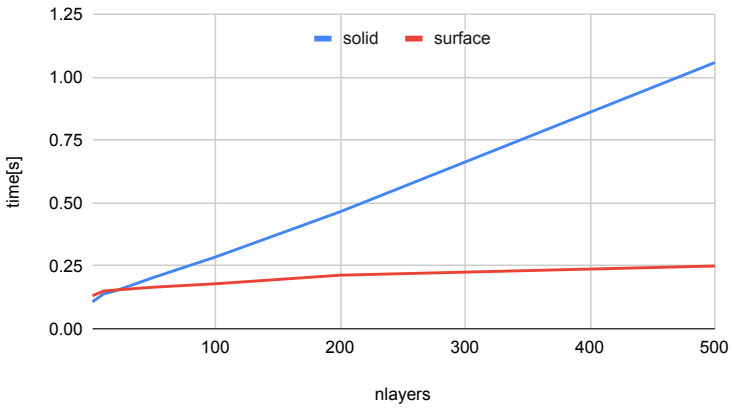

Figure 4. Time for distance computation followed by relocation in a multi-layered sampling calorimeter. The surface model is compared to the current solid model approach.

erations. This has typically a binary tree representation, and the navigation functionality is implemented by pushing the query from the top Boolean node to the final leaves. This allows short-circuiting the Boolean evaluation of some branches of the tree, but it is recursive and has a footprint on the stack that scales with the CSG expression complexity, which limits the generality and the achievable performance on the GPU.

In the new surface modeling approach, each solid is represented by a logic expression describing it as the result of a Boolean operation between half-spaces. For example, a cut tube segment can be described as a Boolean operation between the following surfaces: two planar surfaces p_1 and p_2 limiting the length of the tube along z-axis, two cylindrical surfaces c_1 and c_2 limiting the inner and outer radii, and two planar surfaces p_5 and p_6 containing the z-axis of the tube, limiting the ϕ range of the segment. The Boolean expression giving the corresponding 3D solid for an obtuse ϕ angle is: $p_1\&p_2\&c_3\&c_4\&(p_5|p_6)$. The framed surfaces are replaced with their actual indices in the surface storage. The solid-level CSG composition is then implemented by replacing the solids with their actual scoped surface expressions. The full expression is then simplified using De Morgan's rules, changing the order of operands such that the left-hand operand has the least number of Boolean operations. This optimizes the expression parsing whenever short-circuiting is possible.

In practice, the CSG expression is parsed from left to right, calling the predicate query function for each framed surface index. The results of evaluated expressions in outer scopes are pushed to a temporary stack and then combined with the results on inner scopes. Depending on the Boolean operation and left-hand operand value, the right-hand expression evaluation may be just skipped in case the operation would not change the result (short-circuiting). Even for very complex CSG expressions, a shallow stack is sufficient to handle the operations.

Figure 5 shows the scaling of a complex Boolean tower made as a union of an arbitrary number of layers, each layer being a subtraction between a box and a tube. Although the first implementation is not yet optimized to date, we observe the benefits of simplifying and linearizing the expression evaluation. Besides the better scaling, the new implementation is also stack-friendly. While the maximum complexity reachable for the solid case due to GPU

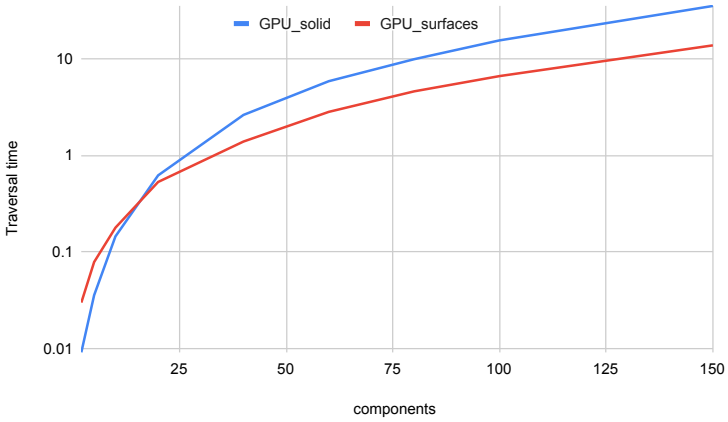

Figure 5. Time for boundary-to-boundary traversal of a single complex tower solid represented as a complex Boolean CSG operation between an arbitrary number of volume components. The surface model is compared to the current solid model approach.

stack size limitations was about 150, the surface implementation can handle much higher complexity with minimal impact on the stack size.

4.3 Integration in AdePT GPU prototype

The first important project milestone was to be able to integrate the new surface model development with one of the simple examples provided in the AdePT prototype, simulating EM showers in a simple layered sampling calorimeter. This was possible after implementing the required navigation functionality for a subset of solids. The change to be made in the example was limited to triggering the conversion to the surface model and replacing the standard navigator with the provided surface model navigator. The change produces compatible results, as shown in figure 6, with similar or slightly better performance compared to the solid-based approach.

The observed differences are due to changes in the random number sequence compared to the solid model. These may occur due to improper relocation in the mother volume while crossing calorimeter layers, in the case of the solid model. This is one of the undesired effects of numerical rounding that affects the current solid traversal approach and is mitigated by the new surface model implementation.

5 Next steps

The next major project milestone is the capability to use the surface model implementation in the case of realistic LHC geometry setups. This requires mitigating the extra memory requirement of the surface model, which is in the current approach estimated to be 1kB per touchable volume. In addition, we plan to use an optimization strategy for the surface model navigation, similar to the bounding volume hierarchy (BVH) approach currently used by the solid model implementation. Both memory and candidate search mitigation are presently under development.

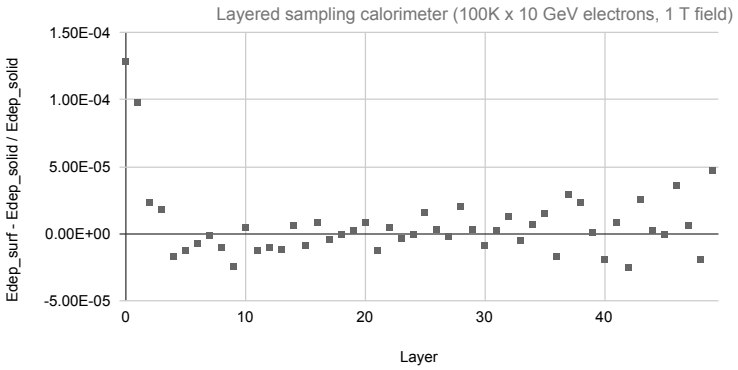

Figure 6. Relative difference of energy deposited per layer in the surface vs. solid geometry for the TestEM3 example in AdePT.

Another important work site is extending the set of 3D solids, eventually encompassing the full set supported by Geant4. We are currently supporting boxes, trapezoids, parallelepipeds, tubes, cones, and polyhedra, while the conversion of other solids such as polycones and simple extrusions is in progress. The extension will allow testing different setups and measuring the impact on GPU simulation performance in both AdePT and Celeritas projects.

References

[1] J. Apostolakis, M. Bandieramonte, S. Banerjee, N. Bartosik, G. Corti, G. Cosmo, V.D. Elvira, T. Evans, A. Gheata, S. Pagan Griso et al., Frontiers in Physics **10** (2022)

[2] R. Brun, F. Bruyant, M. Maire, A.C. McPherson, P. Zanarini, *GEANT 3: user's guide Geant 3.10, Geant 3.11; rev. version* (CERN, Geneva, 1987), `https://cds.cern.ch/record/1119728`

[3] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)

[4] G. Amadio, J. Apostolakis, P. Buncic, G. Cosmo, D. Dosaru, A. Gheata, S. Hageboeck, J. Hahnfeld, M. Hodgkinson, B. Morgan et al., Journal of Physics: Conference Series **2438**, 012055 (2023)

[5] S. Johnson, S. C. Tognini, P. Canal, T. Evans, S. Jun, J.G. Lima, A. Lund, V. Pascuzzi, EPJ Web of Conferences **251**, 03030 (2021)

[6] J. Apostolakis et al., J. of Phys.: Conf. Ser. **608**, 012023 (2015)

[7] I. Stroud, *Boundary representation modelling techniques* (Springer Science & Business Media, 2006)

[8] *ORANGE code repository*, `https://github.com/celeritas-project/celeritas` (2023)

[9] A. Salzburger, J. Niermann, B. Yeo, A. Krasznahorkay, Journal of Physics: Conference Series **2438**, 012026 (2023)