CERN

Summary Student Project Report

Converting Solids to VecGeom's Surface Model

student:

Dušan Cvijetić[1],

University of Belgrade

supervisor:

Dr. Andrei Gheata,

CERN EP-SFT

Geneve,
August 26, 2022

Contents

1 Introduction

Simulating geometry is a necessary part of any detector simulation in high energy physics. It introduces constraints to a given problem, bounding physical properties to regions of space.

One of the most important tasks geometry model has is navigation of moving particles. It must provide information on where the

---
[1] dusancvijetic2000@gmail.com

particle currently is, so that interactions with material can be properly simulated, but also predict where will the particle hit boundary of the current region. This prediction is done through checking ray–boundary intersections, so as to see whether the particle's trajectory leads it to exit the current region and pass into another. [1]

Traditionally, this was done for a single particle and a single potential intersection. However, with the advent of powerful GPU technologies in the recent years, a question arises if there is possibility for parallelization, and therefore speedup, of this process. As the simulation of particle movement is one of the most resource-intensive tasks of collision simulations, its acceleration would be very valuable. VecGeom library [2] is an effort in this direction.

Old models used the concept of solids (3D volumes) to represent the elements of a detector. However, GPU parallelization over the solid structures turns out to be quite inefficient, for they tend to use many registers and exhibit a lot of divergence. A promising approach to solving these problems is decomposing the volumes into surfaces, lowering register usage and producing less divergent algorithms.

2 Bounded Surface Model

There are a few approaches to implementing surface models for geometry simulations, including triangulation or half-space decomposition. VecGeom, however, uses the bounded surface model.

To create a shape in this model, it is necessary to decompose a solid into its limiting surfaces. At first, infinite surfaces are generated. Then, *frames* are created on each infinite surface, defining the actual limits of the solid on that surface. Frames provide a masking operation delimiting the real from the virtual parts of the infinite surface, in a local reference frame. To describe the boundaries of the original solid, the frames must be properly rotated and translated to be placed in the correct positions on the infinite support surfaces. The combination of infinite unplaced surfaces and placed frames defines what is called a framed surface, which completely describes one face of the modeled solid. (figure 1). If a particle is traveling through space and hits the surface, it also has to check if it has landed within the frame. If not, then the collision wasn't real and should be ignored. If yes, then the surface was really hit and
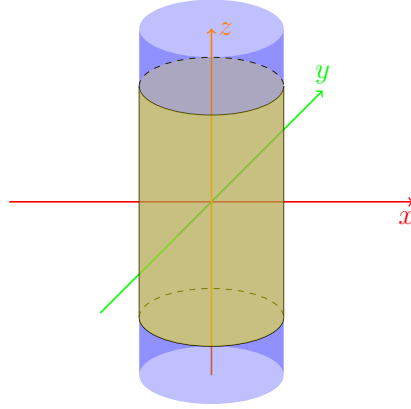
Figure 1: Frame on an infinite cylinder creates a framed cylindrical surface.

it should be checked how this affects the particle's state (i.e. what are the two regions between which a particle is crossing through a surface).

Depending on the setup, it may happen that certain support infinite surfaces are overlapping, although the corresponding object frames are different. Planes might share a common normal (figure 2), or cylinders might have the same axes and radii. In such cases, it is useful to group these smaller structures into a common surface object.
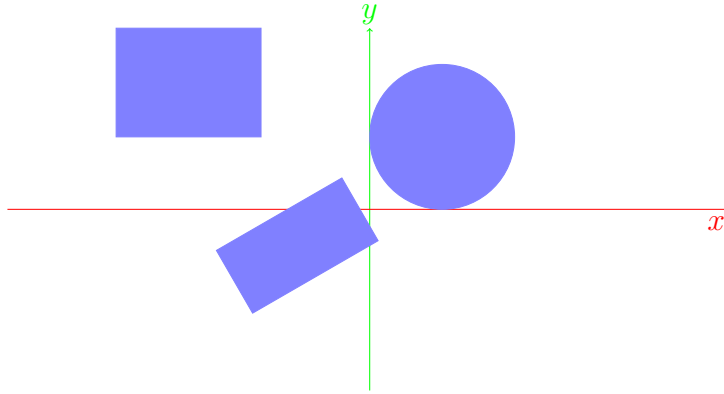

Figure 2: Masked planar surfaces that share a common normal.

A common surface is, just like before, created so that it extends to infinity. That is not very practical, and a way must be found to delimit the useful area from its virtual part. Unlike before, when

frames were used to limit the surfaces based on solid's parameters, the useful part of a common surface is determined by the placement and masks of framed surfaces that make it.

To delimit multiple placed frames on the same common surface, a structure called extent is used. Extents are akin to frames, only they represent "bounding boxes" for all framed surfaces on a common surface. They are the predefined geometrical structures with the smallest possible area that covers all the frames. (Figure 3)
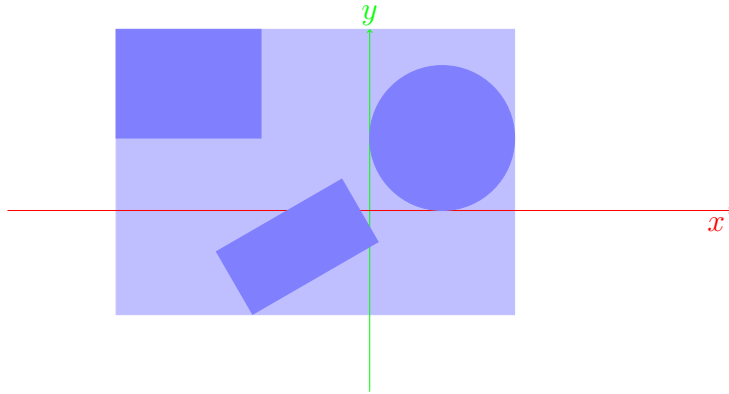
Figure 3: Rectangular extent on a planar common surface. At first, all that the particle sees is an extent. Frames are checked only if the extent is hit.

The purpose of an extent is to avoid expensive individual frame crossing checks for particles traversing a common surface outside the extent area. Since it is much more likely for a particle not to hit within an extent than to hit it, this optimization provides substantial speedup to the navigating algorithm.

Two framed surfaces can also lie on a same common surface and have opposite orientations (i.e. two planes with parallel normals that point in opposite directions). Such cases are treated by decomposing a common surface into two sides. The extents, then, are computed for each side, and not for the whole common surface, and which side should be checked is determined by the navigation algorithm.

3   Implementation

The implementation of described structure in a way that can run on both CPU and GPU presents many constraints. The model must be implemented in header files, to avoid CPU-GPU portability issues. All data structures are templated on a type representing the floating point precision, to allow for on-demand single-precision support. Dynamic memory allocation cannot be used in algorithms meant to run on GPU. Inheritance mechanism also cannot be utilized, and so virtual functions can't be created.

3.1   The old architecture

The initial architecture of the Surface Model library is in its prototype phase. Before I started contributing, there was some confusion with structure names, the delegation of responsibility was in many places inappropriate, and the overall scalability while extending the model to all supported solid types was quite bad. The restructuring had to be done for further development to be possible.

All the data needed for geometry was stored within SurfData structure. This allows greater control over memory allocation and better GPU compatibility. On top of that, various structures were implemented to represent the elements of the surface model: RangeMask, Extent, UnplacedSurface, Frame, FramedSurface, Side and CommonSurface.

RangeMask structure had two real numbers as its members, and was meant for data storage. It implemented a few basic operations, like getting and setting its members. Transformation was a class for moving around the produced surfaces into their places.

Extent data structure was comprised of two RangeMasks and was meant to represent various types of both frames and extents. This introduced confusion about its purpose, and produced messy and repetitive code (for example, the same checks had to be made for both Frame and CommonSurface structures). The limitation of using only two Range objects to represent any possible frame or extent resulted in impossibility to specialize structures based on their purpose. To some extent, it was possible to bypass this limitation through specializing generation algorithms, but even when such hacks were utilized, some shapes still couldn't fit into four real numbers however ingenious the conjured solution was. This struc-
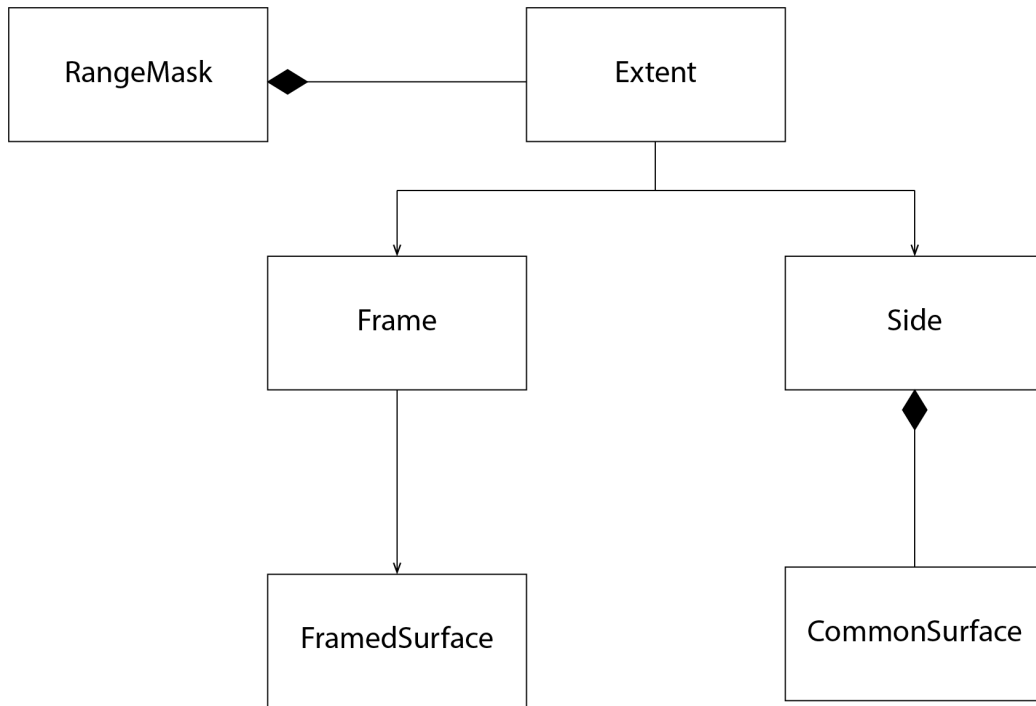
Figure 4: Snippet of architecture diagram before the redesign. There is inconsistency in the way Extents interact with higher level structures, and specialization is not possible.

ture was the main culprit of many problems that arose throughout development.

UnplacedSurface represented infinite surfaces before they were framed.

Frame provided more general interface for connecting the UnplacedSurfaces with Extent objects. It kept the info about the Extent type and the Extent itself as its members. The algorithm for checking if the point lies within the extent was implemented inside the Frame in two variants: one was meant for framed surfaces, and the other, static one, for checking extents on common surfaces.

FramedSurfaces were used to connect Frames with UnplacedSurfaces. Sides and CommonSurfaces were used for grouping of FramedSurfaces, as told before. Sides used Extents for delimiting the useful area.

The diagram of the old architecture is given in figure 4.

3.2 The new architecture

The new architecture used for implementation of surface model had to be scalable, as new types of objects should eventually be supported by the library. This was provided through better object decomposition and delegation of responsibility.

Range structure was done away with, and replaced by more general Vec2D structure, with some new operations implemented (such as vector subtraction or quasi cross product for z-component calculation). Depending on the use case, Vec2D was aliased as Range, Point2D, or AngleVector.

The major restructuring was done on frames and extents. Firstly, the new structures, called Masks, were introduced. Masks serve as low-level implementations of various possible shapes that frames and extents can take, allowing for specialization of different types. All mask structures, apart from constructor, have an `Inside()` member function, whose purpose is to check if a given point lies within mask's shape. Currently, there are only five masks implemented:

WindowMask represents a rectangular mask on a planar surface;

RingMask represents a surface extending between two circular segments, one with radius r_{min} and the other r_{max}, starting at an angle φ_{min} and extending to φ_{max};

ZPhiMask represents a mask delimiting a cylindrical surface segment along z-axis, starting from z_{min} and extending to z_{max}, and spanning between angles φ_{min} and φ_{max};

TriangularMask[2] represents a triangular mask on a planar surface and

QuadrilateralMask represents a convex quadrilateral mask on a planar surfacel,

but the number of mask types should still grow.

The Frame remains as an interface between Mask and Surface. Since the usage of virtual mechanism was not allowed, selecting the right `Inside()` function for applied mask is done through a switch statement in Frame structure's own `Inside()` function. Case is

---

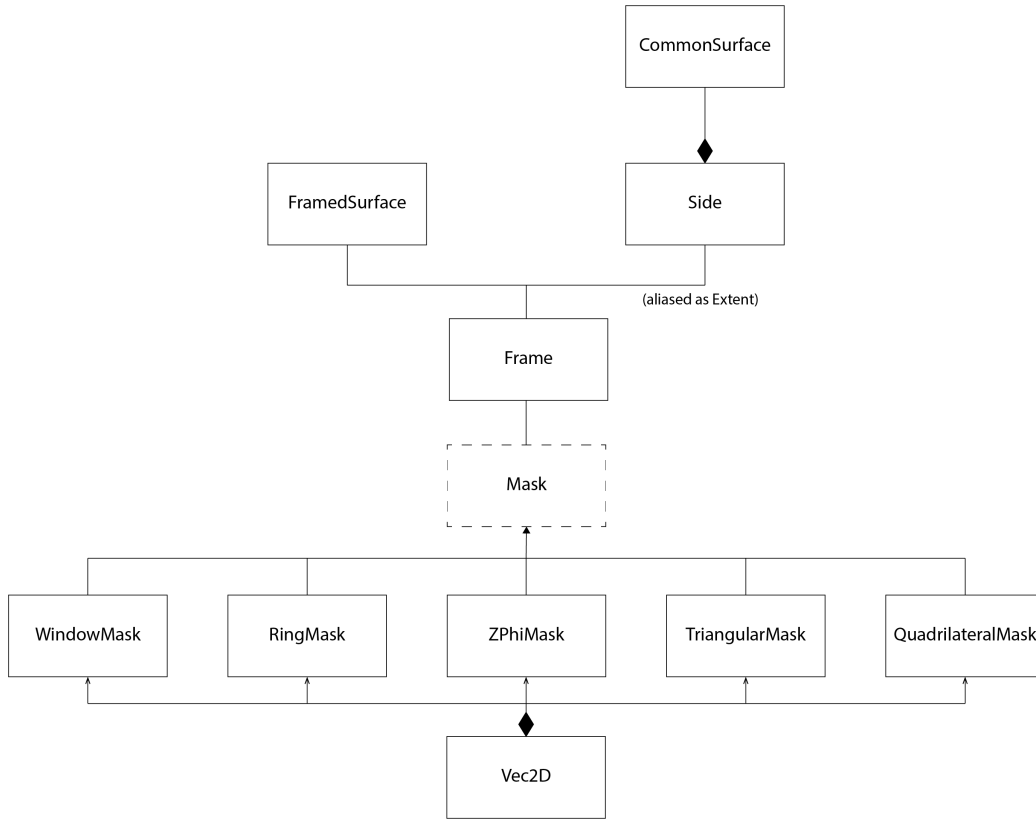[2]These have not been tested yet.

7

Figure 5: Architecture after redesign. Mask structure doesn't really exist, because virtual mechanism cannot be utilized. It is just an illustration of similarity between different mask structures, and not a real parent.

selected based on a mask type, a parameter that is stored as data member.

On Sides, extents are now not implemented as a special class, but rather as an aliased Frame objects. That way, there is much less code duplication, and logically the same mechanisms are kept as a single piece of code. This also allows Masks to be used on both UnplacedSurfaces (through Frames) and Sides. The type of a mask on an extent is enforced upon extent's creation, so that the planar sides will always be delimited by WindowMasks, and Cylinders by ZPhiMasks.

The diagram of the new structure is shown in figure 5.

With these modifications, the model's scalability was increased, and it has become more straightforward to implement new masks

and solid-surface conversions.

4   Implementing shapes

4.1   Tube

The only solid-to-surface conversion that was already done at the start of this project was for box solids. Implementing tube surfaces posed a few new challenges.

In general, tubes occupy the region delimited by two coaxial cylinders with different radii, between non-equal starting and ending angles, and within a finite length along the Z axis. If the angles are equal, the tube's cross-section is a full circle. If the inner radius is zero, the tube is not hollow.

Firstly, an algorithm was written for calculating the distance between a point in space and cylindrical surface along a given direction. As only the planes have been implemented earlier, this was the first second-order system of equations that needed to be solved, and the simple and fast quadratic equation solver was implemented.

RingMasks and ZPhiMasks were introduced for framing surfaces that constitute tube. In general, there were six of those: outer and inner cylindrical surfaces with ZPhiMasks, top and bottom cap with RingMasks and two planar sides for covering the phi-cut of the cylinder with WindowMasks.

4.2   Trapezoidal prism

Another new solid whose conversion has been implemented is the trapezoidal prism. It consists of top and bottom planar surfaces with rectangular masks of (in general) different sizes, and four planar side surfaces with quadrilateral masks. An optimization was introduced so that WindowMasks are created instead of QuadrilateralMasks whenever possible, since running the `Inside()` check then becomes simpler.

4.3   Testing

Validation and benchmarking was performed by simulating a calorimeter with varying number of layers, each consisting of a gap and an absorber.

For testing tubes, a tubular non-hollow calorimeter extending along z axis and spanning full $360°$ was used, filled with tubular layers with same outer radius as calorimeter, but which had both inner radius and a cut in phi.

Testing trapezoidal prisms was done using a calorimeter with trapezoidal cross-section in yz plane and extending along x axis. It was filled with layers with the same cross-section, but made shorter in x direction, along which they were stacked.

The particles were created in a region inside and around the calorimeter, with random positions and velocity directions. The surface model was checked against the old solid-based model to see if the same result is produced. Time was also measured for simulations using both the solid-based and bounded surface models, and surfaces always performed at least as good as solids, often even outperforming them.
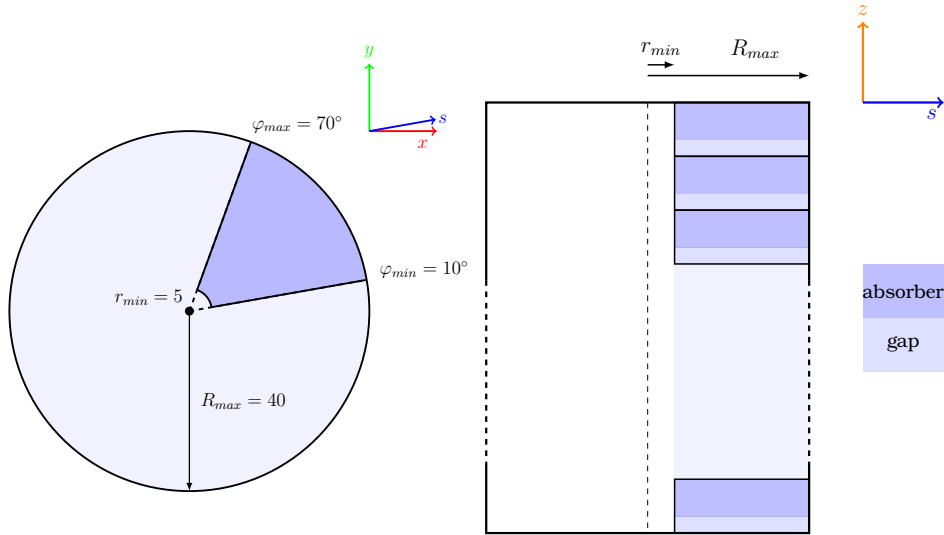

Figure 6: Example of a setup for tube testing.

5   Conclusions

In this report, an overview of the bounded surface model was given. Its implementation was still just an early prototype at the beginning of this summer, but changes that were introduced provided better scalability and more understandable code. There is, of course, still space left for further restructuring. Many header files are still too

big and need to be recomposed into multiple smaller headers, which is a nontrivial task, thanks to omnipresent circular dependence of various components.

Four new mask types have been implemented. Apart from rectangles, it is now possible to mask rings, triangles, quadrilaterals and cylinders. Of course, this is just a beginning and at least as many more masks are needed for the library to become robust enough for real applications. A particular challenge lies with composite masks, and careful planning must be done before their implementation is started.

It is now also possible to create tubes and trapezoidal prisms, raising number of solids that can be converted into surface model to three. Though just a modest number, hopefully the changes to the old architecture will allow it to grow more rapidly now.

VecGeom's bounded surfaces library is a project still in its early development phase. Results so far have been promising, though a lot of work still remains to be done. Hopefully, in a few years we will have a robust and fast library for geometry simulations able to run on the GPU.

Thanks

It has been an honor and a joy to be a part of CERN's EP-SFT group and feel what real software development is like. I owe many thanks to my supervisor, Andrei Gheata, for his never ending patience and dedication to working with me during these two months. It has been a lovely experience.

References

[1] J. Apostolakis, G. Cosmo, A. Gheata, M. Gheata, R. Sehgal, and S. Wenzel, "A vectorization approach for multifaceted solids in vecgeom," *EPJ Web of Conferences*, vol. 214, p. 02025, 01 2019.

[2] J. Apostolakis, M. Bandieramonte, G. Bitzes, R. Brun, P. Canal, F. Carminati, G. Cosmo, J. C. D. F. Licht, L. Duhem, V. D. Elvira, A. Gheata, S. Y. Jun, G. Lima, T. Nikitina, M. Novak, R. Sehgal, O. Shadura, and S. Wenzel, "Towards a high performance

geometry library for particle-detector simulations," *Journal of Physics: Conference Series*, vol. 608, p. 012023, may 2015.