CERN

Summer Student Project Report

Creating Tubes in VecGeom's Surface Model

student:
Dušan Cvijetić[1],
University of Belgrade

supervisor:
Dr. Andrei Gheata,
CERN EP-SFT

Geneve,
August 24, 2022

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1] dusancvijetic2000@gmail.com

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 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.

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 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 decom-

position. 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. Creating a useful structure from infinite surfaces is then done by using frames, which delimit the area that really exist from the virtual one (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 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).

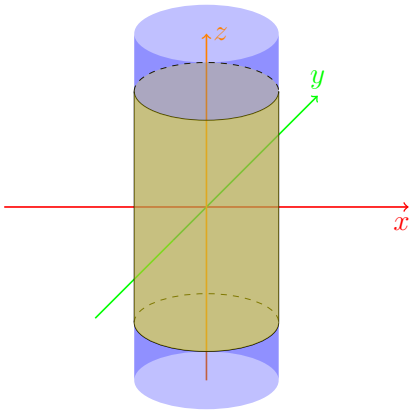

Figure 1: Frame on an infinite cylinder creates a framed cylindrical surface.

Depending on the setup, it may happen that certain surfaces lie in each other's extensions. 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.

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 common surfaces, 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

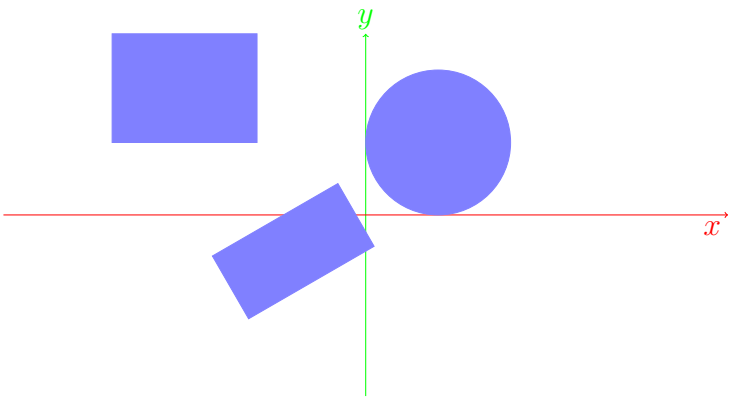

Figure 2: Masked planar surfaces that share a common normal.

geometrical structures with smallest possible area that covers all the frames. (Figure 3)

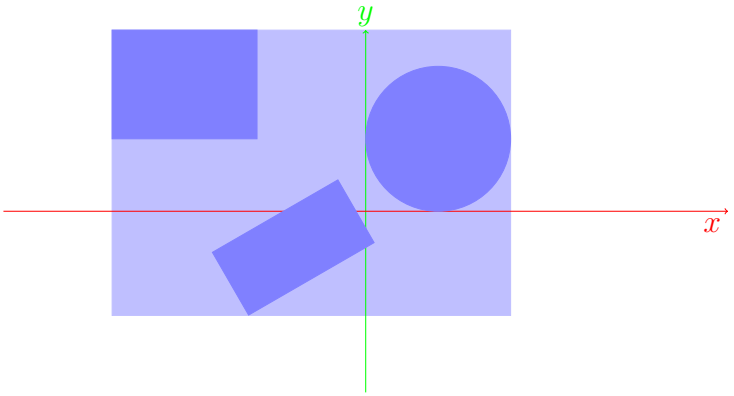

Figure 3: Rectangular extent on a planar common surface. At first, all that the particle sees is an extent. Frames are checked only if the extent is hit.

The purpose of an extent is to cut down the number of frames that have to be checked. When a particle is traversing the space and can potentially hit a certain common surface, the first step is to check whether its trajectory intersects the common surface within its extent. If not, then there is no possibility that the particle will hit any of the framed surfaces, and their frames don't have to be checked separately. Since it is much more common 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 navigation algorithm.

3 Implementation

The implementation of described structure in a way that can run on both CPU and GPU presents many constraints. For one, all structures and functions must be templated, so that appropriate data types can be used depending on architecture the code is running on. 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.

The initial architecture of Surface Model library was in its prototypal phase. There was some confusion with structure names, the delegation of responsibility was in many places inappropriate, and overall scalability was quite bad. The restructuring had to be done to allow further development.

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

//TODO: UML diagram of old architecture.

Range 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 Ranges 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 in 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 with this some shapes couldn't fit into four real numbers however ingenious the conjured solution was. This structure 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 data 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 Extents with UnplacedSurfaces. Sides and CommonSurfaces were used for grouping of FramedSurfaces, as described before.