

A VC IMPLEMENTATION FOR TGEOPCON

JUAN VALLÉS MARTÍN

INTRODUCTION

This report summarises the work in the Geant Vector Prototype (Geant-V) [1] project in the PH-SFT group at CERN, as part of the Summer Student Programme.

The Project. Geant-V is a particle simulation package that makes use of all performance dimensions on modern computing architectures. In particular, the code is inherently parallel in the sense that many particles go through the same operations, so the project aims to obtain performance gains by processing vectors of particles instead of single particles. This way the code can benefit from SIMD instructions and less cache misses.

The task of this Summer Student project was to analyse the `TGeoPcone` class and try to vectorize some of its functions. The approach chosen for this was the use of the library `Vc` [2], and one of the goals of this project was to evaluate the convenience of this approach in comparison to, for example, the use of auto-vectorization or CILK [3].

The Polycone Geometry. The `TGeoPcone` class defines a geometry consistent of a concatenation of tubes and cones, which can be hollow. The Polycone does not need to be fully symmetric around its z-axis - but can have wedges cut out. The parent class of the Polycone is `TGeoBBox`, which means that it has a bounding box.

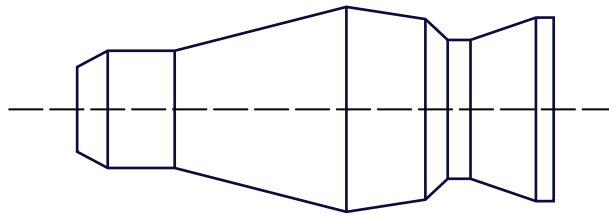


FIGURE 1. A Polycone volume

The functions vectorized during this project were `Contains` and `DistFromOutside`. The former checks if a particle is inside or outside the volume and the latter returns the distance between the particle and the volume (provided the particle is outside of it) in the direction the particle is following.

The Vc Library. Vc is a library that allows to define vectors and operate with them in C++, taking care of the low-level actions to make use of the SIMD instructions. This way the vectorized code is easy to read and write. Masks can also be defined so only some elements in a vector are affected by an operation. Operations between scalar variables and vectors are allowed as well, and the library includes several mathematical functions for vectors. More information about the library can be found at [the documentation page](#).

DEVELOPMENT OF THE FUNCTIONS

This section describes the strategies followed to implement the vectorized functions of the TGeoPcon class. The convention used in the code is adding “_v” at the end of the name of a vector and “_m” at the end of the name of a mask. The results shown for the speedups given by the code were measured comparing the Polycone ROOT serial function and the new vectorized one for a set of 8192 randomly generated particles. The code was compiled (if nothing different is said) with CMake using GCC 4.7, the AVX instruction set (with vector size 4), ROOT 5.34 and Vc 0.7. The work done can be found in the Git repository of Geant-V [4].

The Contains Method. The TGeoPcon_v::Contains_v function was the first one to be implemented. The original function checked if a single particle was in the z -range, r -range and ϕ -range of the volume, returning `false` as soon as one of these conditions was not fulfilled and `true` if they were all satisfied. This kind of early return is not possible when working with a vector of particles, so a mask was created for checking the different conditions, and only when the mask is filled with `false` values the early return happens. Therefore, the following code:

```
|| // check total z range
|| if ((point[2]<fZ[0]) || (point[2]>fZ[fNz-1])) return kFALSE;
```

becomes:

```
|| // check if the particles are inside in the z-direction
|| vdm particleInside_m = (z_v > fZ[0]) & (z_v < fZ[fNz-1]);
|| // if all the particles are outside
|| if( particleInside_m.isEmpty() ) {
||     for(unsigned int j = 0; j < Vc::double_v::Size; ++j) {
||         isin[i+j] = false;
||     }
||     continue;
|| }
```

Since the masks can’t be stored in an array of booleans, a sequential assignment is needed. There is another sequential assignment in the code when the z -segment in which each particle is evaluated (there is no function to do a binary search in parallel). For the following conditions, a vectorial “&” operation is applied to the mask:

```
|| // check if the particles are radially in the volume
|| particleInside_m &= (r2_v > rmin_v*rmin_v) & (r2_v < rmax_v*rmax_v);
```

The function was benchmarked with an 11-segment Polycone present in the CMS ROOT file. The speedup obtained with this function was **1.44**.

The DistFromOutside Method. After implementing `Contains`, the functions `Safety` (which returns the minimum distance between the particle and the volume), `DistFromInside` and `DistFromOutside` (which return the distance to the volume taking into account the direction of the particle) were analysed. The serial implementation of these functions is recursive (if the distance to the closest segment is too big the functions call themselves starting in the next segment), so a different approach needed to be taken in order to vectorize them. As the most important function, `DistFromOutside` was chosen to be implemented next.

The vectorized function `TGeoPcon_v::DistFromOutside_v` uses a brute-force strategy. First, it evaluates some conditions in order to check if it can do an early return (if the particles are outside the z -range and moving away, if they reach the bounding box and if they reach at least the maximum radius of the volume). Then, for each set of particles, it computes the distance between them and each segment, storing the minimum. In order to do that, the nature of the segment (tube or cone) is checked and a vectorized static function to compute the distance is called. An alternative version of the function was implemented where, instead of looping over the vectors of particles and then over the segments, the loop is done first over the segments and the distance of all the particles to each one of them is computed.

Tube and Cone implementations of DistFromOutside. These functions, called `DistFromOutsideS_v` needed to be implemented in order to fully vectorize the Polycone function. They get the parameters that define a vector of particles as arguments, as well as the parameters that define the geometry of the tube or the cone, and they return a vector with the distance between each particle and the volume. Both are similar although the cone version has to do more operations to check the different conditions due to its more complex geometry.

First the distances to the top or the bottom surface are computed and then if they are not valid the distances to the inner (in case that there is one) and outer surfaces are computed. As in the `Contains` method, some of the distances are ready to be returned before others, so a mask `done_m` avoids the valid distances to be modified and allows to check for early returns. Both geometries call the functions `DistToTube_v` and `DistToCone_v`, which needed to be implemented as well. They return two parameters to compute the distance between a particle and a cylindrical or conical surface. At the beginning of the function there are two early return conditions. Checking if the particles are outside the z -range and moving away produces speedup gains in both geometries, while checking if the particles reach the bounding box only does for the cone geometry.

Both functions were benchmarked, and the speedup gain was **4.85** for the tube and **2.62** for the cone. Due to their similarity, the functions `DistFromInside_v` for the tube and the cone were implemented as well, and the speedups are, respectively, **1.99** and **1.92**.

Discussion. In order to evaluate the speedup of the `TGeoPcon_v::DistFromOutside_v` function all the Polycone shapes in the CMS detector were taken, and the speedup was computed and averaged over the Polycones with the same number of segments. The following figure shows the average speedup as a function of the number of segments for two different instruction sets, SSE4 and AVX (with vectors of 2 and 4 elements,

respectively) and for the two alternative nestings. The blue bars represent the number of Polycones in CMS with that number of particles (right axis).

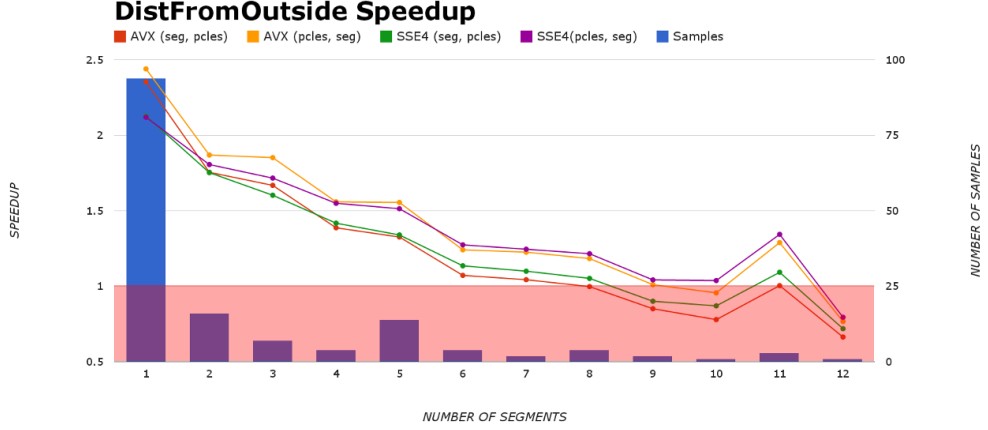


FIGURE 2. Speedup as a function of the number of segments for TGeoPcon_v::DistFromOutside_v

The figure shows that computing the minimum distance for each set of particles is faster than computing the distance between each segment and all the particles. It also shows that AVX has a better performance for Polycones with less than 6 segments, and that this approach produces speedup gains in Polycones with less than 9 segments. Fortunately, the figure shows that there are more Polycones with a few segments than with a large number of them, although it has not been tested which ones are used the most. The most common number of particles is, however, 1, which means that they are cones or tubes, and their specific `DistFromOutside` functions are faster than the Polycone implementation.

The number of segments could be checked at the beginning of the function, and depending on it the serial function or the vectorized one could be called, but a different approach for the large Polycones was implemented.

Vectorizing over Segments. Even though the strategy of the project is to compute vectors of particles, the function `TGeoPcon_v::DistFromOutsideSegs_v` takes one particle and computes the distances to the segments next to it in the z -direction. This way all the segments behind the particle are not computed. After some initial early checks, the z -position and the z -direction of the particle are checked and several lists with vectors of parameters are built.

```

|| vd * rminl_v; // inner radii of the lower planes
|| vd * rmaxl_v; // outer radii of the lower planes
|| vd * rminh_v; // inner radii of the higher planes
|| vd * rmaxh_v; // outer radii of the higher planes
|| vd * dz_v; // half-heights of the segments
|| vd * zloc_v; // relative z-positions of the particle
|| vi * seg_v; // segments to which the parameters belong

```

```
|| Bool_t * istube; // nature of the segments
```

Iterating over this lists the distances to the segments can be computed (static functions for the tube and the cone were implemented as well). The first segment reached gives a cap for the distance, and there can't be any segment of the same nature after it with a smaller distance. If the next set of vectors represents a segment of the other nature, however, it can contain a segment that is closer than the one reached.

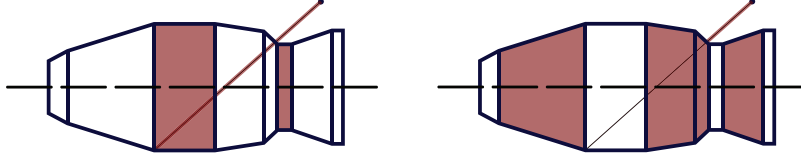


FIGURE 3. Example of the segment vectorization algorithm

The approach gave a speedup of **0.72** for the Polycone (the 11-segment one used to test `Contains_v`, and of **5.41** and **2.21** for the tube and the cone, respectively (the benchmark for them generates 8192 tubes or cones and computes the distance between a particle and each one of them). This function is just a prototype and there are still many improvements that can be done in it.

FURTHER WORK

There are still many improvements that can be done in the code, those I could think of are commented in capital letters in the code, but here there is a list with some of them.

- In `TGeoPcon_v::DistFromOutsideSegs_v` it is not very effective to build a list for each particle, so all the possible lists could be built at compilation time.
- In the same function we are computing the distance to the bounding box for an early check, so we can use geometry to see at which z -position the particle crosses it and use it to compute the first segment in the list.
- The vectorized version of `TGeoBBox::DistFromOutside` takes vectors filled with the same scalar value in every position as arguments when the `Vc` library allows operations between vectors and scalars.
- The original code of `DistFromOutside` for the tube and the cone checks if the particle is inside the volume within machine precision, which has not been included in the vectorized version.
- There is no version of the `DistFromOutside` functions (for either the tube, the cone or the Polycone) that include the non-full- ϕ case.
- There are many functions that can be inlined such as `TGeoShape::Big()`.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Federico Carminati and Sandro Wenzel for allowing me to participate in this project and for helping me during my work, and the Summer Student team for giving students like me the opportunity be part of such amazing programme. My stay here, in which I have learned a lot and I have met

incredible people from all over the world, has been very enriching in both the academic and the personal field. Thank you.

REFERENCES

- [1] Geant-V official website: <http://geant.cern.ch/>
- [2] Vc documentation page: <http://code.compeng.uni-frankfurt.de/docs/Vc-0.7/>.
- [3] CILK official website: <http://software.intel.com/en-us/intel-cilk-plus>
- [4] The Git commit number of this work is [2a053e1b5d1e62692f409cd64376c79e5bc51897](#)