Simulation-aided optimization of detector design using portable representation of 3D objects

Jan Balewski

Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA

E-mail: `balewski@MIT.EDU`

Abstract. Use of the Standard Tessellation Language (STL) for automatic transport of CAD [1] geometry into Geant is presented. The hybrid approach of combining Geant native and STL objects is preferred. The tradeoffs between the CPU cost of the simulation and the accuracy of tessellation are discussed.

1. Introduction

The design of a new complex detector for a physics experiment typically requires multiple iterations between two teams: the physicists modeling detector properties using simulation package like Geant and the engineers assembling the detector model in CAD to ensure it can be built. The transfer of information from CAD to Geant as well as the double coding of the frequently changing geometry require a sizable investment of resources.

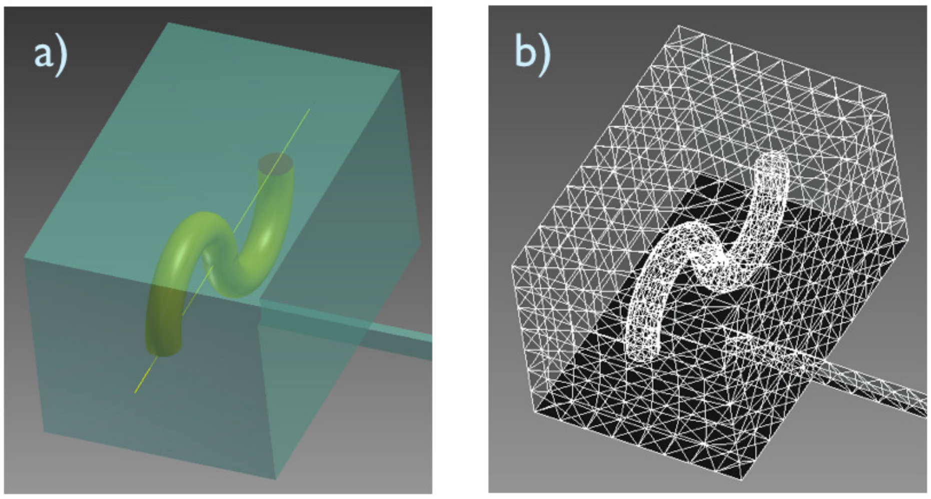

Figure 1. A geometry defined in CAD (a) can be transported to Geant as a mesh of triangles using STL format (b).

[1] CAD is understood as generic name for AutoCad, SolidWorks, or similar tool.

The advent of portable representation of 3D objects via Standard Tessellation Language (STL) file format significantly simplifies transfer of geometry information and reduces the evaluation time of the evolving detector configurations. In section 2 we will describe the basic properties of the STL format, section 3 discuses example of use of STL in Geant simulations.

2. Features of Standard Tessellation Language

The STL file format allows for a convenient way to transport a detailed geometry from a CAD program (see fig. 1a) in to Geant in the form of triangular mesh (see fig. 1b). Most of CAD-like programs have the capability to export 3D objects in to the STL format.

As a prerequisite, one needs to compile Geant with the CADMesh library [1], which in turn requires the ASSIMP library [2]. Next, it suffices to add the following 2 lines to your code to import to Geant the STL file for the object named 'mySphere'

```
CADMesh * mesh = new CADMesh("mySphere.stl", STL,...);
G4VSolid  * mySolid = mesh->TessellatedMesh();
```

From that point one can use the `mySolid` pointer as if it was created by the Geant-native:

```
G4Solid *G4Orb( name="mySphere", rMax=...);
```

Unfortunately, one can't export from CAD the information about the material used to build 'mySphere' and it needs to be re-defined again in Geant.

Although the position and orientation of an object can be passed from CAD to Geant it is advisable to not use this option. CAD uses the absolute reference frame for any object exported as STL. In Geant, as illustrated in fig. 2, we often define a nested geometry of a complex object using nested local coordinate frames. Also, one often places multiple copies of the same Geant object at different absolute locations. To preserve those features of Geant it is more convenient to export from CAD as STL only one element of each type, read it into Geant, then clone it, and re-position as needed.

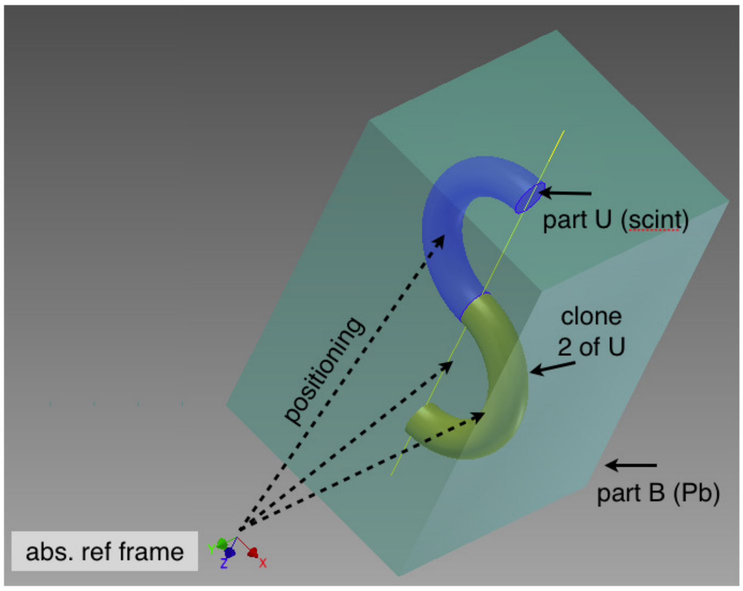

Figure 2. Building a complex object (S-shape) in Geant by replicating and translating the elementary object (U-shape).

There are few tradeoffs in using the STL format. CAD allows user to choose the granulation of the mesh. Smaller triangles will represent a curved surface with higher fidelity. But at the expense of more CPU time spent in simulations, because Geant will need to use smaller steps to resolve boundaries between the volumes. Additionally, the size of the STL file will grow for a smaller mesh step, leading to the longer initialization time of the STL-based geometry encoded in Geant. Fig. 3a shows the CAD ideal model of a half-torus, 4cm in diameter. Fig. 3b illustrates, that if we generously required the maximum deviation between the STL and the ideal shape to not exceed 60μm the STL file will have the size of 51kB and will contain about 1k facets. However, if the highest fidelity of 1μm is requested then the same half-torus would use 2.5MB of the disk space, because now it consist of 50k facets, as shown in fig. 3d.

It is up to the user to decide for which part of the geometry the high fidelity of STL representation is necessary. E.g. to simulate a response of a silicon tracker the 1μm tolerance for STL geometry may be needed, but to study the background produced by a 4m long iron magnet yoke it may suffice to be accurate up to 1mm.

A more saddle issue is a too crude STL mesh 'cuts corners' while approximating series of nested curved objects (think about modeling of an onion). If different layers of such 'onion' were designed and exported from the CAD separately they may partially overlap if re-assembled in Geant. Although Geant probes volumes overlaps using random thrown points it may not detect if two volumes overlap by just 1e-6 fraction of the total volume. But when you run a large scale simulations some particles will get to the ambiguous region and given job may get stuck, run forever. To prevent it, it is advisable to add an additional clearance space between the layers of the onion, make those gaps few times larger than the intended STL mesh accuracy selected while exporting form CAD.

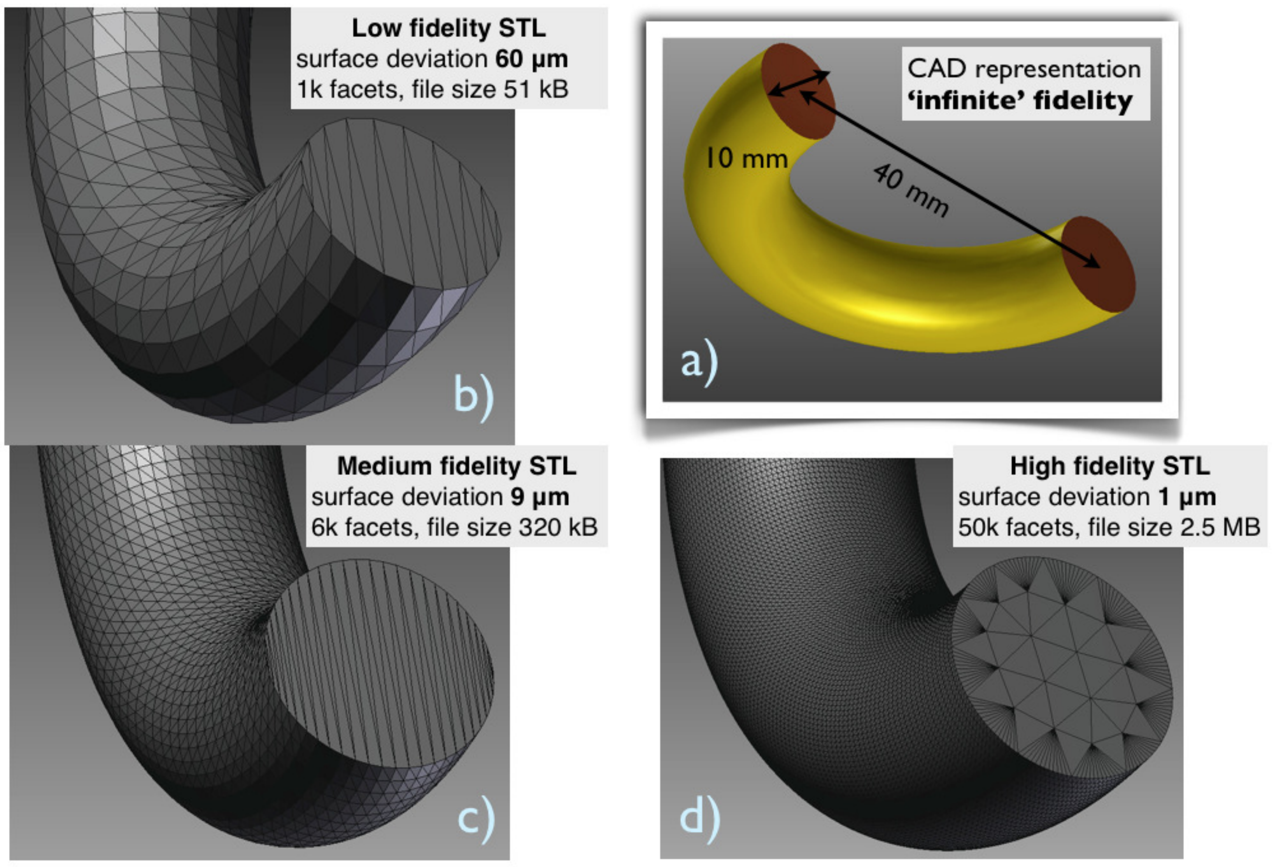

Figure 3. The STL file size and number of facets strongly depend on the required fidelity. (a) is ideal CAD geometry, (b)-(d) show consequences of decreasing maximal deviation of STL from 60μ to 1μm.

2.1. Tessellation precision vs. CPU cost
To study the CPU cost of the Geant simulations we have used a simplified implementation of the electromagnetic calorimeter consisting of many S-shaped scintillating fibers embedded in a block of lead, as shown fig. 4a. In the simulation this calorimeter was exposed to 400 positrons, we measured the total CPU time. The graph in fig. 4b shows how the CPU usage depends on the accuracy of the STL representation, and on the energy of the positron. We concluded, that for the fixed positron energy the CPU time increases by 60% if we replace the ideal, Geant native, torus by its crudest STL model. Another factor of 2 in CPU cost is acquired if we require the highest fidelity matching of geometry. As seen from the graph, the pattern is energy independent [2].

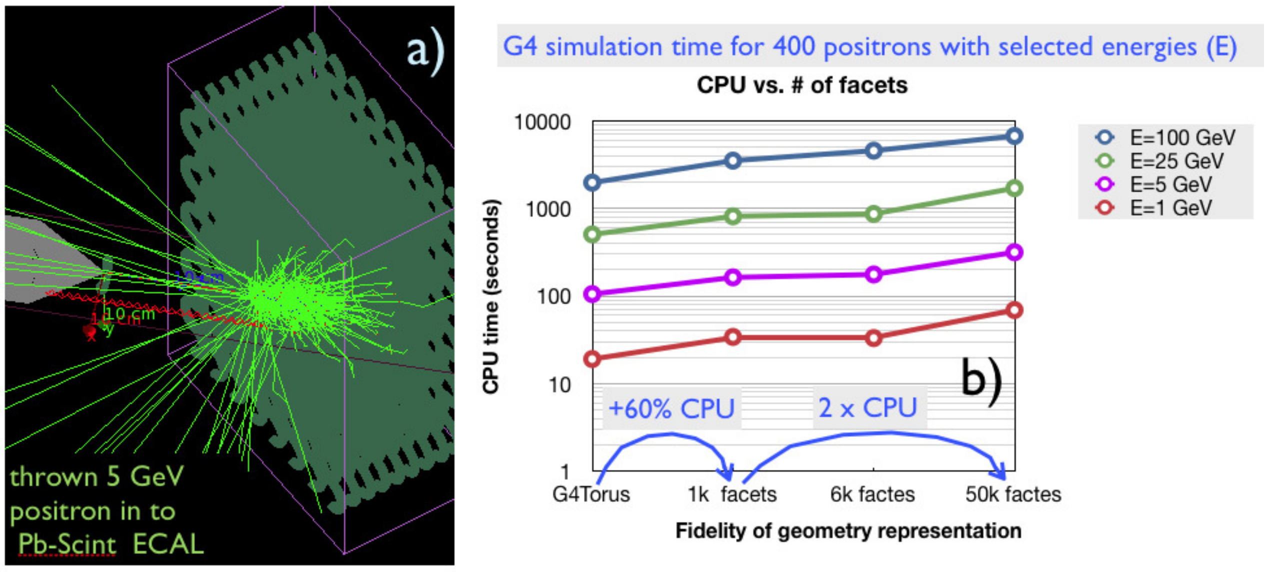

Figure 4. (a) The Geant model of a plastic-lead EM calorimeter used for (b) evaluation of the CPU cost of simulations. Different STL resolutions are labeled on the x-axis, results for different energies of the incident particle are shown with different colors.

3. Hybrid geometry model in Geant
The most practical approach of parallel building of the Geant and CAD models for a large detector is to use the hybrid approach.

Fig. 5 shows an example of the CAD detector model consisting of a multi-disc detector embedded in a phi-symmetric magnet. The parts labeled in blue have simple shapes and it is trivial to implement them in Geant using its native volumes. The more complex parts are labeled in pink. It took a lot of engineering time for the CAD design to adjust the angles, dimensions, make the whole detector assembly fit together. For such case, it is more practical to export the complex parts from CAD using STL, because they will fit again in Geant instantly. The precise location, orientation angles can be read from CAD projections and used while re-assembling the STL volumes in Geant - this saves time too.

[2] One can assume that number of secondary particles generated and tracked in the simulation of the EM cascade remains approximately constant for the fixed energy of the incident particle and does not depend on the granulation of STL mesh.

To conclude, use of Standard Tessellation Language for automatic transport of CAD geometry to Geant is a very advantageous. However, a careful evaluation of costs and benefits of using the STL format for each detector part is advisable. Finally, note it takes some practice to streamline the process of updates of STL geometry.

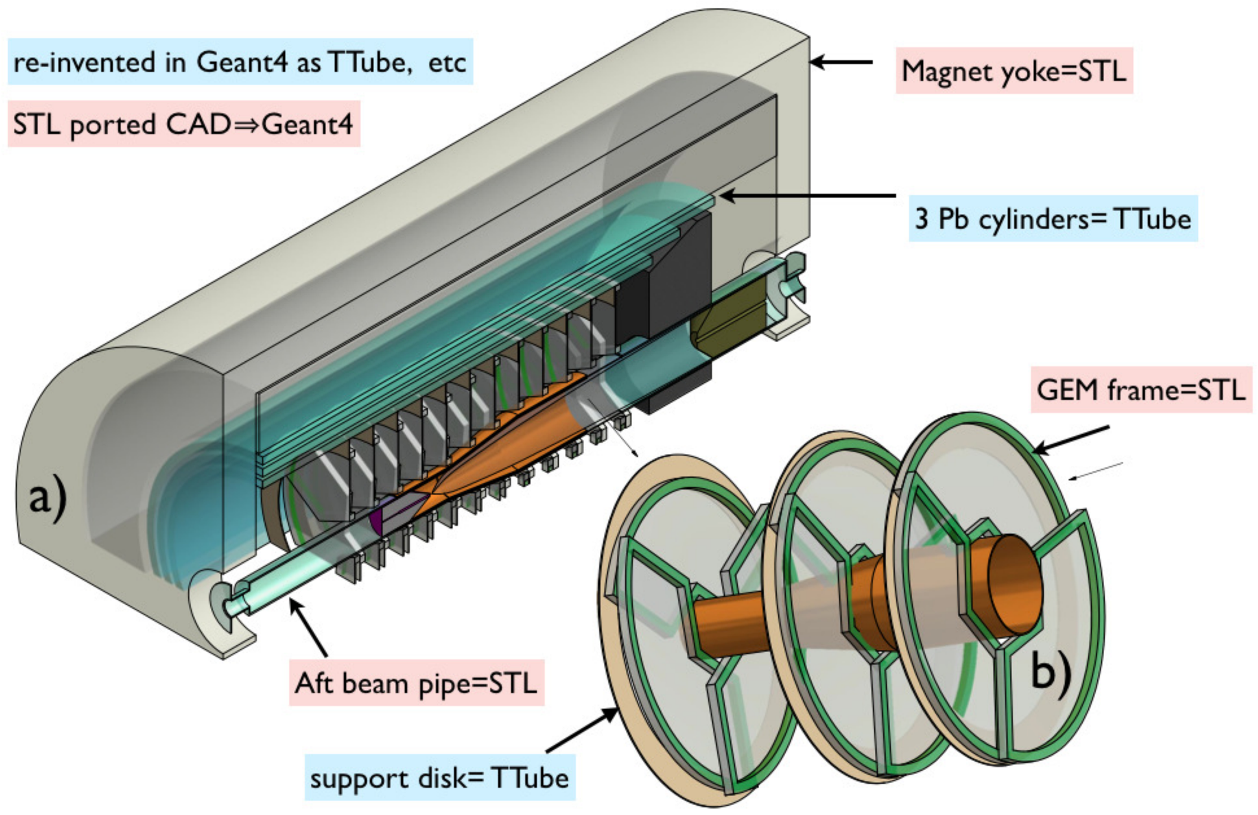

Figure 5. Illustration of a complex geometry of a detector (a). Only more complex parts (b) are exported from CAD to Geant using STL file format.

References

[1] CADMesh library: http://cadmesh.googlecode.com/files/cadmesh-v0.9.tar.bz2
[2] ASSIMP library: http://sourceforge.net/projects/assimp/files/assimp-3.0