The section "The detector geometry description in the BM@N experiment" starts (lines 90-116) with the general description of the GDML language, then continues (lines 117-132) with the discussion of approaches used by other experiment to convert the geometry for the VR engines and concludes with the statement that in the presented work no format converters will be used a new 'transfer' will be implemented. The title of that section simply does not correspond to the content of it, which has basically nothing to do with the detector geometry in the BM@N experiment (which, as stated before, is implemented in ROOT and it was not even mentioned in this section).

Section 4 is meant to describe the actual methodology of going from a GDML file to the Unity-based VR visualisation which is the core part of this work. My understanding is that what is presented is effectively a GDML 'reader' that allows to parse a GDML file and to create a 'in memory' representation of the geometry using the Unity structures and allowing its 'VR' visualisation. The authors are using the word 'geometry transfer' for it, but I find it a bit confusing and unclear. In the case of Geant4 and ROOT, the word 'GDML reader' is used for the code that allows to read a GDML file and create an in memory representation of the corresponding Geant4 or ROOT geometry. If this is the functionality that is described here for Unity, it would be clearer to use the same terminology (and for a comparison, refer to the GDML to ROOT reader which allows to visualise any GDML geometry using ROOT).

From what I understood, the key part of the algorithm of going from GDML to Unity is to create the meshes using the mesh generators. The authors don't give any details about that code. Which language is it implemented in? How do the algorithms look for different solids? Is it possible to add examples of the code to illustrate that?

The most confusing part of that section is the subsection 4.5 where the geometry hierarchy is supposed to be discussed, but it contains only a brief description of what the 'hierarchy' in the GDML file is (already described in Section 3 and repeated here). There is no information here how that hierarchy is actually interpreted inside Unity and what is done to recreate 3D positions of the volumes (I believe there is some calculation done to go from local to global coordinates system?).

My recommendation, therefore, would be to do a major revision and extension of the Sections 3,4 and 5, including among others, a clearer discussion of Figure 2 explaining what is represented by each block (a file, a piece of code, a memory content), the description of the algorithms and some examples of the code used to create the meshes, discussion of how the 3D scene is created from the geometrical hierarchy, and some description of the availability of that software (open source?). It would be extremely useful if simple example (for a geometry consisting of a view simple volumes) could be added to illustrate the complete process of the geometry data conversion from GDML to Unity.

10: What do you mean by 'and requirements'? What requirements? Are the requirements decreasing?

118: what do you mean by 'sets of HEP experiments'? You mean the geometry exchange between different application (like exporting it from Geant4 to load into ROOT)?

118: it is not entirely clear what 'custom transfer development' is, please explain more in details what you mean here

147: it is not clear what 'applying the world reference' means? How to you apply it?

Figure 2: please explain more in detail what is represented by each block (file, code, memory, algorithm, etc)

152: We were discussing GDML and not we go back to ROOT? A bit confusing.

153-156: what does it mean 'analogues of materials representing initial materials at an appropriate level of detail'? What level of detail? I guess, the are simply different colours?

172: as I mention in the general part, please describe in more in details (with code snippets) the implementation of mesh generators.

184: it's not clear what are the 'temporary geometric shapes'

195-199: What does this subsection explain? It only repeats what are positions and rotations in GDML. There is nothing about Unity here.

200-210: It describes what is in GDML, but how is it interpreted in Unity? How is that hierarchy used when creating a Unity scene?

216: how can it be saved? In what format?

218: in the given run

235: if this can be used for other high-energy experiments, please provide the source from which the software can be obtained

Reviewer #2: The paper presents a method to convert GDML files exported from ROOT to a proprietry Filmbox (FBX) file format commonly used by the game development system and engine called Unity. The actual format is not clear from the manuscript. This is a useful and important development, viewing geometry is key for understanding monte carlo simulations and experimental data. There is also the added benefit for VR/AR collaboration/engineering/digital twins and also public engagement and outreach

Other issues with the methodogy

-------------------------------------

L83 Filmbox (FBX) is a file format that does not have an freely available description. There are excellent open file formats which can be simply read into Unity and Unreal. So examples of this are Graphics Language Transmission Format (gltf) and its binary equivalent (glb) from the Chronos Group and Universal Scene Description (USD) and its compressed version (USDZ) from Pixar/Apple etc.

L140 In general does the package parse GDML or only the GDML output from ROOT. These are two very different things, for example mathematical syntax allowed in constants. I think this should be clarified. As line-by-line parsing only really works if all numerical parameters are defined on that "line".

General inaccuracies or misconceptions in the manuscript

--------------------------------------------------------

L84 : there is a package that can convert GDML to common 3D formats (pyg4ometry, freeCAD GDML workbench)

L85 : there is no Unity format per se.

L159 : In essence the shader returns the pixel value (R,G,B,A) or (H,S,V,A). There are also non pixel output shaders now available.

L163 : A texture is more often not a high resolution image, more often than not, it is programatically generated and fills the texture memory on disk.

L170 : I don't think this is entirely true. I do think that quad subdivision meshes and similar will be supported in Unreal/Unity. In general at the final rendering pipeline is on a graphics card, it will indeed be using only triangles

L172 : In general I dont think every solid needs a dedicated mesh generator is generally true. As I would use the same generator for Sphere and Orb for example

L197 : Positions and rotations do not have to have names when defined inline with the solid or structure

Areas where is explanation is insufficient

L177 There is a lot of description missing here

1) Is the mesh water tight

2) How is the texture UV unrapping done. i.e creation of texture coordinates

3) Normals need to be generated taking into account of large dihedral angles (so not to smooth sharp edges)

How is the mesh stored?

1) In a half-edge data structure?

L209 It is not clear how the instances are done in this direct conversion. So many logical volumes can be replicated as physical volumes saving the mesh memory requirements. How is this handled?

L185 3D mesh Boolean operations is a complicated problem (to do robustly). I don't think this description is sufficient.

Minor issues

--------------

L98 : Would select a verbatim fixed width font for XML/GDML to differnetiate from normal text.

Figure 1: Not clear where this 3D view comes from, Unity? or some CAD package?