ISBN: 978-3-95450-247-9 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2024-MOPS24

PYG4OMETRY UPDATE: A TOOL TO CREATE GEOMETRIES FOR GEANT4, BDSIM, G4BEAMLINE AND FLUKA

S. T. Boogert*, Cockcroft Institute, Daresbury Laboratory, Daresbury, UK
L. Nevay†, F. Stummer, F. Metzger, CERN, Geneva, Switzerland
W. Shields, Royal Holloway, University of London, Egham, UK
L. Pertoldi, M. Huber, Technical University Munich, Germany

Abstract

Studying the energy deposition in accelerator components, mechanical supports, services, ancillary equipment and shielding requires a detailed computer readable description of the component geometry. The creation of geometries is a significant bottleneck in producing complete simulation models and reducing the effort required will allow non-experts to simulate the effects of beam losses on realistic accelerators. This paper describes a flexible and easy to use Python package to create geometries usable by either Geant4 (and so BDSIM or G4Beamline) or FLUKA either from scratch or by conversion from common engineering formats, such as STEP or IGES created by industry standard CAD/CAM packages. This paper describes the updates to pyg4ometry since IPAC19. These updates include ROOT geometry loading, refactored geometry processing using CGAL, direct CAD file loading using Open Cascade, geometrical feature extraction, geometry comparison algorithms and improvements to FLUKA conversion. The paper includes small examples of the new features with explanations.

INTRODUCTION

Pyg4ometry is a Python package with the aim to make the creation of input geometry for radiation transport Monte Carlo codes quicker, easier and less error prone. In addition, it allows new workflows for the creation and conversion of geometry. The package has classes which represent common geometric concepts in Geant4 (and hence BDSIM [1], G4BEAMLINE) and FLUKA [2] as well as geometric algorithms and functions (provided by CGAL [3] and Open Cascade [4]). Internally, pyg4ometry closely matches concepts found in Geant4 and GDML. Each format supported can be loaded and viewed using a 3D visualiser based on VTK [5]. Geometries can be inspected, checked for overlaps, combined, converted, or created from scratch in a Python script.

Pyg4ometry is effective as it uses best-in-class open source software. Both CGAL and Open Cascade are very large C++ projects and pyg4ometry requires only a small subset of the available functionality. External packages based on C++ are made accessible in Python by using pybind11 [6].

Pyg4ometry was not envisioned nor designed to replace DD4Hep and other similar frameworks for the HEP community. DD4Hep deals with a significantly larger problem space of geometry and its subsequent use in simulation, reconstruction and display. Pyg4ometry does not attempt to solve these problems and it is focused on the workflows most commonly required for accelerator physicists.

Full details of the implementation can be found in the publication [7], but this paper presents the latest code developments and example use-cases.

ROOT GEOMETRY IMPORT

ROOT is a data format and analysis package that is dominant in the high-energy physics community. It also includes its own geometry description that can be used with Geant4 and other codes [8]. A loader was written in pyg4ometry for ROOT geometry, and once loaded the geometry can be visualised with the usual VTK viewer. This was used to load detector geometries and then, with the new comparison algorithms described later in this paper, to compare and validate newly prepared GDML geometry of the LHCb detector at CERN. This geometry was created independently of pyg4moetry but it was used to validate that the new geometry was equivalent across the thousands of volumes.

CGAL IMPROVEMENTS

Recently pyg4ometry has created granular bindings to both CGAL and Open Cascade. For example GDML to FLUKA conversion relies heavily on 2D convex decomposition of arbitrary polygons. A monolithic binding strategy would be to create a function that matches the pyg4ometry required functionality, so for example `decompose(polygon)` and this function return the desired result. A more granular approach would be to bind the CGAL polygon object, bind all the CGAL functions and allow the user to decompose the polygon. The more granular binding approach allows users to customise and extend the capability of pyg4ometry within Python and leveraging the full capability of CGAL and/or Open Cascade. A more granular approach is significantly more time consuming for the developers as more C++ needs to be wrapped and tested within Python, however, there is a clear benefit and this was the chosen strategy.

With this granular approaching to binding, future workflows can trivially be added with minimal extension to pyg4ometry. For example, mesh repair algorithms and tetrahedralisation of meshes are clear future additions making full use of the capability of the underlying tools to process meshes loaded from CAD.

* stewart.boogert@cockcroft.ac.uk
† laurie.nevay@cern.ch

ISBN: 978-3-95450-247-9 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2024-MOPS24

CAD LOADING IMPROVEMENTS

CAD descriptions of geometry differ significantly from those used by Geant4 and FLUKA, in that CAD models typically use a B-rep structure which includes a sense of topology. An example could be a simple cube, which in Geant4 and FLUKA would be presented by a single solid or body respectively. In B-rep, a cube would be a SOLID which has six FACES and each FACE would have one square WIRE which bounds the face and then the WIRE could consist of four EDGEs, each EDGE would require two VERTICES. The FACEs in this example are a simple plane and the EDGEs are lines in three dimensions.

The most recent version of pyg4ometry allows users to navigate the topological data in a CAD file by exposing the relevant Open Cascade functionality using pybind11. In general a common workflow would be conversion of a CAD solid to a mesh-like data structure and with a monolithic interface only this single function needs to be exposed. This granular interface enables feature extraction, half-space decomposition, and conversion to half-space representations that are valuable additions to the code.

GEOMETRICAL FEATURE EXTRACTION

Direct and complete conversion from CAD to Geant4/GDML is not always required. For example, a detailed beam pipe model including flanges, mounting plates, screws etc. might exist in a CAD model. However, such a detailed model is unnecessary and would be computationally prohibitive, and the original software is not always available to physicists creating radiation transport models. With feature extraction, the CAD model can be loaded and inspected to give, for example, three radii (inner and outer minor radii and major radius) of a torus. These parameters can then be used to quickly created new original geometry for FLUKA or Geant4.

GEOMETRY COMPARISON AND REGRESSION

When converting geometry either with pyg4ometry or externally through other means, it is often useful to know whether two models are consistent or different. A new set of comparison algorithms was added to allow both exact and equivalent comparison of two geometry trees. A set of tests can be defined for the comparison and certain comparisons can be ignored if known to be different. Using the available visualisation meshes in pyg4ometry, we can compare surface area and volume to allow equivalence comparison even if geometry is create from different shapes or primitives.

CONVERSION TO FLUKA

There already is an excellent tool in FLAIR [9], but there is still a need for automatic and programmatic (i.e scriptable) interface to FLUKA geometry. In particular, configurable geometry for devices such as collimators, can be easily regenerated in a Python script with simple calculations and parameters.

Conversion from Geant4 to FLUKA was in the original release of pyg4ometry, however, recent work has greatly improved the usefulness of the produced FLUKA input by more carefully considering the conversion. Because of some of the more complex shapes and the geometry hierarchy in Geant4, the resultant FLUKA input may contain a prohibitive quantity of brackets. Expanding these could result in an explosion of terms making the input unusable, and possibly poor tracking performance. This problem is particularly acute at the top level of a geometry where a structure like

```
WORLD -(OBJECT1) -(OBJECT2) -(OBJECT3) ...
```

might arise. If each object is composed of multiple regions, this is problematic and it is highly preferable that those objects are made from a single primitive such as a BOX,

Additionally, the conversion of a Boolean expression for a REGION into the disjunctive normal form (DNF), i.e the union of convex solids, was introduced. The expansion is performed symbollically using SymPy [10].

Conversion of Geant4 to FLUKA geometry requires construction of a sequence of a hierarchy of transformations (ROTDefi in FLUKA). It is possible to completely remove all transformations by including the transformation in the BODY definition. For example FLUKA has an arbitrary plane (PLA in FLUKA) defined by a normal $\mathbf{n}$ and point in the plane $\mathbf{p}_0$. Removal (or *baking in*) the transformation $\mathbf{x} = \mathbf{Mx} + \mathbf{d}$ would give new plane normal of $\mathbf{Mn}$ and plane point $\mathbf{Mp}_0 + \mathbf{d}$. This is relatively straightforward for all FLUKA bodies apart from those that are quadrics. Here, the transformation is a little more complex if the quadric is defined as $xQx^T + P \cdot x + R = 0$, then the transformed quadric is

$$\begin{aligned} Q' &= M^T Q M, \\ P' &= MP + M^T Q^T d + M^T Q d, \\ R' &= R + d^T Q d + d^T P. \end{aligned}$$

FLUKA does not allow reflections in the transform (ROTDefi) definitions, however, Geant4 physical volumes can potentially include reflections. During Geant4 to FLUKA conversion, if pyg4ometry detects a reflection (i.e $\det(\mathbf{M}) \neq 1$) then the complete transformation including reflection is compounded into the body as previously described.

As an example, Figure 1 shows a BDSIM quadrupole, which has been exported to GDML and converted to FLUKA input. This quadrupole geometry cannot be expanded to DNF and so will not run in FLUKA, but shows the potential of pyg4ometry conversion. Therefore, further development was introduced to handle such a case.

Some of the problems with subtracting extrusions can be avoided by performing a 2D decomposition of the extrusion section, including voids. This is implemented in pyg4ometry using 2D convex decomposition in CGAL. A

ISBN: 978-3-95450-247-9 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2024-MOPS24

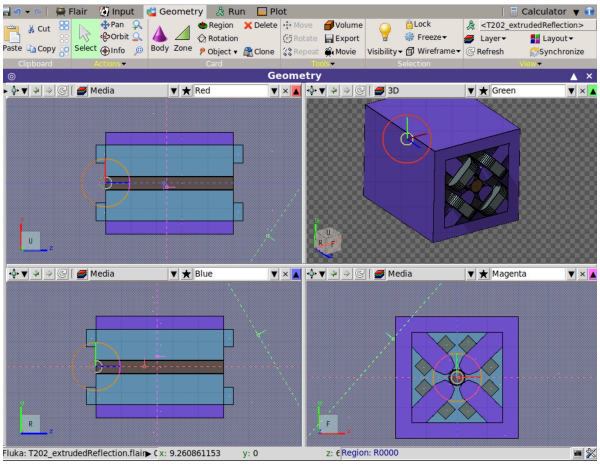

Figure 1: Example of a BDSIM (Geant4) quadrupole converted using pyg4ometry to FLUKA input and displayed in FLAIR.

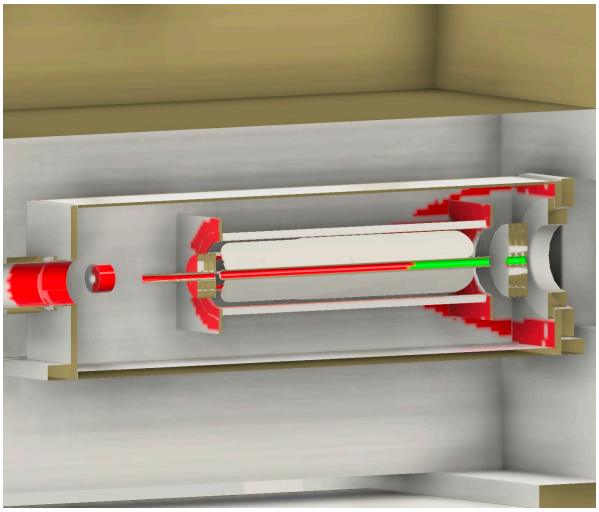

Figure 2: The Möller experiment geometry rendered in Fusion 360.

new 'extruder' class was introduced to register 2D polygons with assigned materials and to maintain a simple boundary that corresponds to a single boundary primitive.

INTERESTING USE CASES

Since publication and deployment, pyg4ometry is finding applications in high energy physics (HEP) detector and accelerator collaborations. In HEP, models for FASER, the Möller experiment, LUXE, and Legend have made use of the code. In accelerator applications, there are many BDSIM users preparing geometry with pyg4ometry.

FASER is a forward search experiment at the LHC in CERN searching for axion-like particles and measuring neutrino fluxes [11]. A detailed beamline model of the ~ 500 m of LHC accelerator and its environment were modelled combining FLUKA geometry, BDSIM-generated GDML files and new geometry created directly using pyg4ometry. Additionally, studies for FASER-2 [12] at the Forward Physics Facility [13] are making use of the code rapidly test and optimise detector configurations.

LUXE is a non-linear QED experiment proposed for the European XFEL [14]. LUXE converts its Geant4 model (the nominal source of the geometry) to a FLUKA model for neutron fluence studies, with great success.

The Möller experiment is a $\sin^2\theta_W$ experiment and has been using pyg4ometry combined with a ray tracing code to determine the number of bounces optical photons would require to reach detectors [15]. This uses the meshing of GDML geometry. Figure 2 shows experiment where the geometry is rendered in Fusion 360. The regions highlighted in red are visible to the source, whereas regions highlighted in green are visible to the source and detector.

The authors of this proceeding are also preparing a new Python package for constructing beamlines in FLUKA heavily based on pyg4ometry, called pyflubl. This allows the creation of FLUKA models of complete beamlines whilst allowing the user to define the source of geometry for each component, be it from a FLUKA input file for that single component, or a user-provided Python class using pyg4ometry to create the geometry newly. Such a package is only possible with a class library package such as pyg4ometry.

CONCLUSIONS

Pyg4ometry is a mature and stable code used by the HEP and accelerator community alike. It allows users a programatic interface to common geometry tasks and the ability to automate and parameterise these tasks. Significant improvements have been made to the CAD and CGAL interfaces and conversion to FLUKA. Given the developments described in this paper two important future develops will be 1) using the Open Cascade-CAD interface to allow conversion to half-space representations of solids and 2) a notebook (Jupyter) interface to pyg4ometry. The first improvement will allow a direct path between CAD models and FLUKA. The second improvement will allow users to work in a familiar analysis environment, and allow the integration of MC output with geometry for creation of visualisations and plots.

REFERENCES

[1] L. J. Nevay *et al.*, "BDSIM: An accelerator tracking code with particle–matter interactions", *Comput. Phys. Commun.*, vol. 252, p. 107200, Jul. 2020. doi:10.1016/j.cpc.2020.107200

[2] T. T. Böhlen *et al.*, "The FLUKA Code: Developments and Challenges for High Energy and Medical Applications", *Nucl. Data Sheets*, vol. 120, pp. 211–214, Jun. 2014. doi:10.1016/j.nds.2014.07.049

[3] M. Botsch, D. Sieger, P. Moeller, A. Fabri, "CGAL User and Reference Manual, 5.0.3 Edition", CGAL Editorial Board, 2020. https://doc.cgal.org/5.0.3/Manual/packages.html#PkgSurfaceMesh

[4] Open Cascade SAS, Open Cascade, https://www.opencascade.com, 2024 (cited 15th May 2024).

ISBN: 978-3-95450-247-9 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2024-MOPS24

[5] W. Schroeder, K. Martin, B. Lorensen, *The Visualization Toolkit–an Object-Oriented Approach to 3D Graphics*, 4th edition, Kitware, Inc., 2006.

[6] pybind11: Seamless operability between C++11 and Python, https://github.com/pybind/pybind11/.

[7] S. D. Walker, A. Abramov, L. J. Nevay, W. Shields, and S. T. Boogert, "Pyg4ometry: A Python library for the creation of Monte Carlo radiation transport physical geometries", *Comput. Phys. Commun.*, vol. 272, p. 108228, Mar. 2022. doi:10.1016/j.cpc.2021.108228

[8] R. Brun and F. Rademakers, "ROOT-An object oriented data analysis framework", *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 389, no. 1–2, pp. 81–86, Apr. 1997. doi:10.1016/s0168-9002(97)00048-x

[9] V. Vlachoudis, "FLAIR: A Powerful But User Friendly Graphical Interface For FLUKA", in *Proc. Int. Conf. on Mathematics, Computational Methods & Reactor Physics (M&C 2009)*, Saratoga Springs, New York, 2009.

[10] SymPy, https://www.sympy.org/en/index.html/.

[11] FASER Collaboration, "Technical Proposal for FASER: ForwArd Search ExpeRiment at the LHC", 2018. doi:10.48550/arXiv.1812.09139

[12] FASER-2 Collaboration, O. Salin *et al.*, "Development of tracking software and detector design studies for the proposed FASER-2 experiment at the Large Hadron Collider", in *Proc. LHCP2023*, Belgrade, Serbia, May 2023, p. 241. doi:10.22323/1.450.0241

[13] J. L. Feng *et al.*, "The Forward Physics Facility at the High-Luminosity LHC", *J. Phys. G: Nucl. Part. Phys.*, vol. 50, no. 3, p. 030501, Jan. 2023. doi:10.1088/1361-6471/ac865e

[14] R. Jacobs, "LUXE: A new experiment to study non-perturbative QED in electron-LASER and photon-LASER collisions", 2022. doi:10.48550/arXiv.2205.06096.

[15] L. A. Barrett, J. Mott, and K. S. Kumar, "Program to Identify Secondary Background Sources in the MOLLER Experiment", presented at APS DNP & JPS Fall Meeting 2023, https://www.lucbarrett.info/Poster.pdf/.