ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPTS054

PYG4OMETRY : A TOOL TO CREATE GEOMETRIES FOR GEANT4, BDSIM, G4BEAMLINE AND FLUKA FOR PARTICLE LOSS AND ENERGY DEPOSIT STUDIES

Stewart Boogert*, Andrey Abramov, Joshua Albrecht,
Gian Luigi D'Alessandro, Laurence Nevay, William Shields, Stuart Walker
JAI at Royal Holloway, University of London, Egham, TW20 0EX, UK

Abstract

Studying the energy deposits 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 provide the ability of non-experts to simulate the effects of beam losses on realistic accelerators. The 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. The conversion requires an intermediate conversion to STL or similar triangular or tetrahedral tessellation description. A key capability of `pyg4ometry` is to mix GDML/STEP/STL geometries and visualisation of the resulting geometry and determine if there are any geometric overlaps. An example conversion of a complex geometry used in Geant4/BDSIM is presented.

INTRODUCTION

Simulating radiation transport in an accelerator beamline requires a description of the 3D layout, structure, dimensions and material properties of the physical objects that the beam particles can interact with. This description is commonly referred to as a "geometry model" or "geometry". In general, the elements (volumes) in such a model must not overlap, ensuring that particles can be located in only one volume at a time. There is also a trade-off between the geometry detail and the simulation execution time. Because of those constraints, radiation transport geometries have traditionally been prepared by hand.

`Pyg4ometry` started as a python scripting tool to generate beam line geometries for BDSIM. BDSIM is a Geant4 application which allows a user to rapidly create a full three dimensional model of an accelerator from an optical description. A guiding principle of BDSIM is rapid simulation of accelerator models, the MADX input format for example can be converted for use in BDSIM in minutes. Rarely described in accelerator optical descriptions is the geometry of the physical material that comprises the accelerator, beam-pipe, magnets, supports, tunnel, beam instrumentation etc. The aim of `pyg4ometry` is to create a tool in which complex geometry can be created as quickly as a generic BDSIM model. A key requirement is to be able to integrate and composite geometry sources to a single file.

Particle Transport Codes

There are multiple different Monte Carlo (MC) codes to simulate the transportation and physics processes of particles though accelerators and detectors, these include Geant4 [1], MCMPX [2] and FLUKA [3]. Generally accelerator codes, like MAD8 [4], MADX [5], Transport [6] etc, are interfaced to a MC code to produce a complete simulation of beam losses. Two beam line simulation tools have been developed on the basis of Geant4; BDSIM [7–9] and G4Beamline [10].

Geometry Generation and GDML

The specification of the geometry of the material surrounding an accelerator can be an exceedingly time consuming and error prone task. Typically either the detector or accelerator infrastructure is constructed over many years and the simulation geometry can be created over similar time scales. This does not allow for rapid simulation of a system as the burden of creating the geometry is too great. An XML-based markup language, Geometry Description Markup Language (GDML) is used as the file format for geometry export in `pyg4ometry`.

SOFTWARE IMPLEMENTATION

`Pyg4ometry` is a collection of python classes that mimic closely the C++ interface of Geant4. The aim to have all of the "detector" description classes implemented in python, these include geometry, materials and optical surfaces. The `pyg4ometry` defined geometry can then quickly be written as a GDML file for loading into BDSIM or G4Beamline. This is a much quicker interface to a full C++ Geant4 application and any programmed geometry can be viewed quickly using VTK. Geometry defined using `Pyg4ometry` can be converted to a surface triangulation using primitive mesh generation of each solid in python and a constructive solid geometry (CSG) library based on Binary Space Partitioning (BSP) trees. GDML has a simple mathematical expression language so geometries can be parametrised, this is also implemented in `pyg4ometry` using ANTLR [11]. A Python interface to the geometry primitives of Geant4 allows conversion applications to be developed from FLUKA/STEP/STL descriptions to `pyg4ometry`. Finally and most importantly `pyg4ometry` provides an interface to GDML, as the python interpreter performs important syntax checking of large and complex geometries.

* stewart.boogert@rhul.ac.uk

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPTS054

EXAMPLES

Python

A complete code example for creation of a simple geometry consisting of three Iron solids is shown below. The python interfaces are so similar to the Geant4 C++ interface and the user can refer to the Geant4 documentation.

```
import pyg4ometry.gdml as gd
import pyg4ometry.geant4 as g4
import pyg4ometry.visualisation as vi
import numpy as _np

# create empty data storage structure
reg = g4.Registry()

# 1) expressions
wx = gd.Constant("wx","100",reg)
wy = gd.Constant("wy","100",reg)
wz = gd.Constant("wz","100",reg)
bx = gd.Constant("bx","10",reg)
by = gd.Constant("by","10",reg)
bz = gd.Constant("bz","10",reg)

# 2) materials
wm = g4.MaterialPredefined("G4_Galactic",reg)
m  = g4.MaterialPredefined("G4_Fe",reg)
# 3) solids
wb = g4.solid.Box("wb",wx,wy,wz,reg)
b  = g4.solid.Box("b",bx,by,bz,reg)
s  = g4.solid.Orb("o",bx/2,reg)
t  = g4.solid.Tubs("t",0,bx/2,bz,0,2*_np.pi,reg)

# 4) structure
wl = g4.LogicalVolume(wb, wm, "wl", reg)
bl = g4.LogicalVolume(b, m, "b", reg)
sl = g4.LogicalVolume(s, m, "s", reg)
tl = g4.LogicalVolume(t, m, "t", reg)
bp = g4.PhysicalVolume([0,0,0.0],[0,0,0],
                       bl, "b_pv", wl,reg)
sp = g4.PhysicalVolume([0,0,0],[-2*bx,0,0],
                       sl, "s_pv", wl,reg)
tp = g4.PhysicalVolume([0,0.5,0],[2*bx,0,0],
                       tl, "t_pv", wl,reg)

# set world volume
reg.setWorld("wl")

# 5) gdml output
w = gd.Writer()
w.addDetector(reg)
w.write("simple.gdml")

# 6) visualisation
v = vi.VtkViewer()
v.addLogicalVolume(wl)
v.addAxes(40)
v.view()
```

The code is divided in 6 blocks of definitions, reusable parameters, materials, solids, structure and placement, GDML IO and finally visualisation. A key difference between pyg4ometry and Geant4 is a dedicated object known as the Registry to store all geometry definitions to be stored in the output file. The VTK output from this example is shown in Figure 1, where three solids, a cube, sphere and cylinder, are placed with translations and rotations within a cubical world volume.

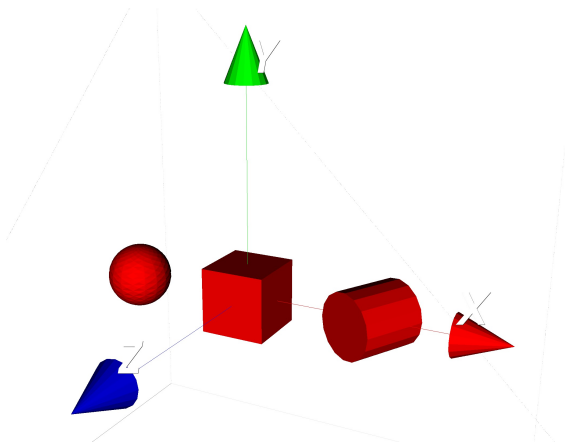

Figure 1: Example of three primitive Geant4 solids rendered in VTK.

Figure 2 shows a more complex geometry example of a cavity beam position monitor and short sections of beam pipe.

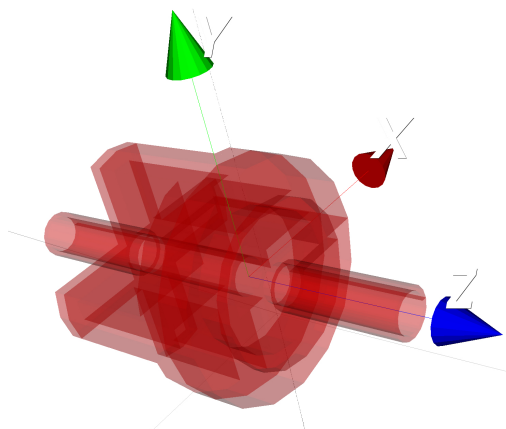

Figure 2: Example of a more complex geometry example, a cavity beam position monitor.

Standard Tessellation Language (STL)

STL file format describes a raw unstructured triangulated surface, where the surface normal is defined by the ordering of the vertices. Pyg4ometry supports the loading STL files and conversion to G4TessellatedSolid. A classic STL example loaded in pg4ometry and displayed in VTK is shown in Figure 3

```
import pyg4ometry
r = pyg4ometry.stl.Reader("example.stl")
reg = r.getRegistry()
```

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPTS054

Figure 3: STL surface mesh rendered in VTK using pyg4ometry.

Computer Aided Design/Manufacturing (CAD/-CAM)

Arbitrary conversion of CAD/CAM files to `pyg4ometry` is a very challenging task. Typically users convert the CAD description into an intermediate surface triangulation format (such as STL described previously) and load the geometry as a G4TessellatedSolid. An interface for loading STEP/STP files was creating using FreeCAD/OpenCASCADE. A single solid in general corresponds to a *part* and multiple placements of a solid to a *part assembly*. A example of a conversion of a large STEP file to pyg4ometry/GDML is shown in Figure 4.

```
import pyg4ometry
r = pyg4ometry.freecad.Reader("example.step")
reg = r.getRegistry()
```

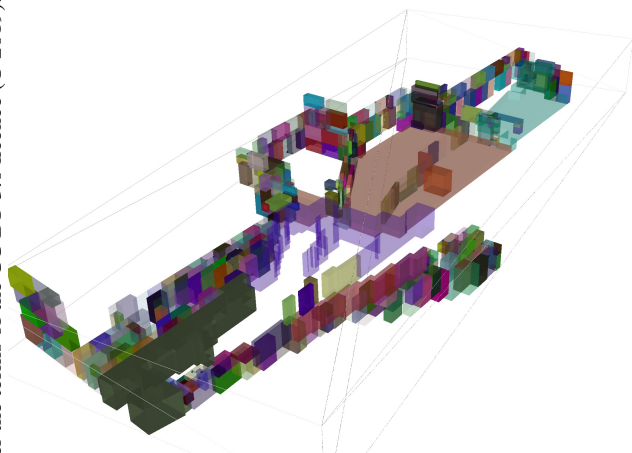

Figure 4: Example a conversion from STEP file description of the shielding of the CERN East Area T8 and T11 test beam lines to GDML. Each part is given a randomly assigned colour to clearly indicate the structure of the shielding.

BDSIM

Geometry created by `pyg4ometry` can be directly loaded into Geant4 based. Figure 5 shows the cavity BPM example loaded into BDSIM.

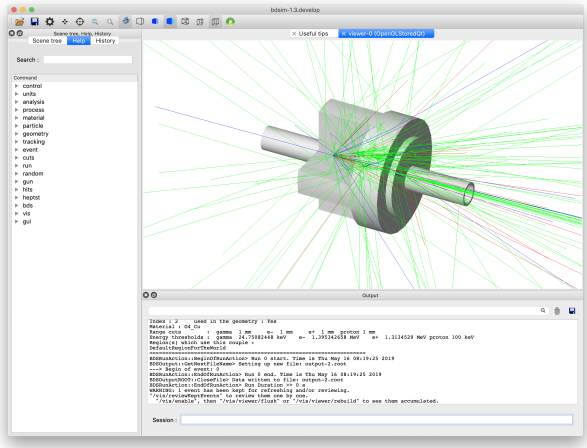

Figure 5: Example of geometry in Geant4/BDSIM. The example shows a 50 GeV proton interacting with the geometry material.

SUMMARY AND FUTURE DEVELOPMENTS

There any many potential enhancements that are being considered for `pyg4ometry`. Given the interface matches so closely that of Geant4, an alternative C++ output writer can be quickly written to allow `pyg4ometry` to generate C++ code which can be directly compiled into a Geant4 application. Although not described in this publication, a FLUKA geometry loader has been written and can load large pre-existing geometries into `Pyg4ometry` and subsequently write GDML files. This is currently being refactored and will appear in the next public release of `pyg4ometry`. In addition to the FLUKA to GDML conversion, conversion of Geant4/GDML geometry to FLUKA is yet unstarted but a relatively straight forward task. The requirement that volumes do not overlap causes a problem for the CAD/CAM (STEP) conversion to Geant4/GDML as there is no requirement that the parts and assemblies are not overlapping. Finally a graphical user interface (GUI) can be developed so allow users without programming experience to generate geometry.

`Pyg4ometry` is a powerful package to rapidly develop material geometries for accelerator beam line simulations. Currently existing geometry can be loaded from GDML, STEP, STL and FLUKA files. We introduce a new python interface to create geometry and adapt existing geometry. `pyg4ometry` can composite geometry from multiple sources to create a complete model which can be exported directly to GDML and then can be loaded into a Geant4 based application. Although based on GDML and Geant4 `pyg4ometry` can be easily extended to other MC simulation geometry formats. `Pyg4ometry` is available via git and published under a GPL licence [12].

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPTS054

REFERENCES

[1] Recent developments in Geant4, NIMA 835, pages 186-225, 2016

[2] MCNPX user manual, https://laws.lanl.gov/vhosts/mcnp.lanl.gov/pdf_files/la-ur-02-2607.pdf

[3] A. Ferrari, *et al.*, "FLUKA: a multi-particle transport code," *CERN Report CERN-2005-10*, 2005.

[4] MAD8 documentation, http://mad8.web.cern.ch/mad8/

[5] MAD-X documentation, http://madx.web.cern.ch/madx/

[6] PSI Graphic Transport Framework by U. Rohrer, based on a CERN-SLAC-FERMILAB version by K.L. Brown et al.

[7] L. Nevay, *et al.*, "Bdsim: An accelerator tracking code with particle-matter interactions," , arXiv:1808.10745, 2018.

[8] BDSIM manual, http://www.pp.rhul.ac.uk/bdsim

[9] L. J. Nevay *et al.*, "BDSIM: Recent Developments and New Features Beyond V1.0", presented at the 10th Int. Particle Accelerator Conf. (IPAC'19), Melbourne, Australia, May 2019, paper WEPTS058, this conference.

[10] G4Beamline documentation, http://www.muonsinternal.com/muons3/G4beamline

[11] ANTLR documentation, https://www.antlr.org

[12] Pyg4ometry project, https://bitbucket.org/jairhul/pyg4ometry/src/master/