Geometry preparation for Geant4 and Fluka

CERN. 18th Jan 2020
Stewart Boogert

Andrey Abramov, Gian Luigi D’Alessandro, Laurie Nevay, William Shields Benjamin Shellswell, Stuart Walker

https://bitbucket.org/jairhul/pyg4ometry/src/develop/
http://www.pp.rhul.ac.uk/bdsim/pyg4ometry/
RHUL group has developed BDSIM, a code to make Geant4 accelerator models


Want to insert custom components / customise models

- Geometry preparation takes a long time
- Needed to make geometry preparation as quick as possible to compliment BDSIM

Create geometry from other codes e.g. Magnetic or electromagnetic modelling

Interpreter or compiler checking of syntax (not possible with GMDL)
RHUL group has developed **BDSIM** a code to convert accelerator descriptions to Geant4 (geometry, material, fields etc.)

- Geometry preparation takes a long time
- Need to make geometry preparation as quick as possible to compliment BDSIM
- Create geometry from other codes e.g. Magnetic or electromagnetic modelling
- Interpreter or compiler checking of syntax (not possible with GDML)

- Rich language for creating geometry
- Parametrized construction (length safety)
- Lots of geometry exists in Flair/Fluka, STL and CAD
- Physics comparisons between Geant4 and Fluka (or other code)
- Facilitate geometry reuse and modification
- Started life as an internal group tool to accelerate geometric model development
- Set of python classes to aid geometry generation → **PYG4OMETRY**
CERN IR1 example

H. Lefebvre (FASER) and S. Walker (ATLAS)

LHC Tunnel Complex for accelerator background in detector
Requirements

- Load (and convert) GDML, STL, STEP, FLUKA files
- Complete support (reading/writing) of GDML
- Visualize geometry
- Check for overlaps and geometry issues
- Composite (load and place) geometry from different sources
- Rendering for data analysis
- Modify geometry (cut holes, remove material etc.)

- Leverage modern tools and programming
- Lightweight
- Open source and simple to install
- Simple to use API (think of a summer student)
- Simple to contribute to (think of a PhD student)
- Reasonable performance (I could not render ATLAS in GDML ;-)
Guiding principles and implementation

- Follow patterns of Geant4 (object interfaces, methods and internal data)
- Use GDML as a fundamental file description of geometry
- Use existing codes/libraries wherever possible
- Aim for 100% test coverage
- Create python class representation for geometric data (other data too)
Technology tools and dependencies

All dependencies are all open source and well maintained.

ANTLR
Visualisation Tool kit
FreeCAD
SymPy
CGAL

Python

ANTLR
VTK
FreeCAD
SymPy
CGAL

Open TECHNOLOGY

CASCAD
Important design features (G4/GDML)

- GDML interpreted via ANTLR grammar
- Dynamic (late) evaluation of expressions
- Complete coverage of GDML including mathematical expressions
- All different types of physical volume (placement, replica, parametrized etc)
- All solids (G4) can generate a tri/quad mesh
- Simple Boolean (union, intersection, subtraction) library based on BSP trees
- Meshes created once per LV and placed as instances in rendering pipeline (ok different for param and replica volumes)
- Scene tree created from PV-LV tree
import os as _os
import pygeometry.gdml as _gd
import pygeometry.geant4 as _g4
import pygeometry.visualization as _vi

def Test(vis = False, interactive = False):
    reg = _g4.Registry()

    # defines
    wx = _gd.Constant("wx","50",reg,True)
    wy = _gd.Constant("wy","50",reg,True)
    wz = _gd.Constant("wz","50",reg,True)
    bx = _gd.Constant("bx","10",reg,True)
    by = _gd.Constant("by","10",reg,True)
    bz = _gd.Constant("bz","10",reg,True)
    wm = _g4.MaterialPredefined("G4_Galactic")
    bm = _g4.MaterialPredefined("G4_Fe")

    # solids
    ws = _g4.solid.Box("ws",wx,wy,wz,reg,"mm")
    bs = _g4.solid.Box("bs",bx,by,bz,reg,"mm")

    # structure
    wl = _g4.LogicalVolume(ws,wm,"wl",reg)
    bl = _g4.LogicalVolume(bs,bm,"bl",reg)
    bp = _g4.PhysicalVolume([0,0,0],[0,0,0],bl,"b_pv1",wl,reg)

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

    # gdml output
    w = _gd.Writer()
    w.addDetector(reg)
    w.write(os.path.join(_os.path.dirname(__file__), "T001_Box.gdml"))
    w.writeGmadTester(os.path.join(_os.path.dirname(__file__),"T001_Box.gmad"),"T001_Box.gdml")

    # test _repr_
    str(ws)

    # test extent of physical volume
    extentBB = wl.extent(includeBoundingSolid=True)
    extent = wl.extent(includeBoundingSolid=False)

    # visualisation
    v = None
    if vis :
        v = _vi.VtkViewer()
        v.addLogicalVolume(reg.getWorldVolume())
        v.addAxes(_vi.axesFromExtents(extentBB)(0))
        v.view(interactive=interactive)

    return {
        "testStatus": True,
        "logicalVolume":wl,
        "vtkViewer":v"
Geant4 (advanced python) example

- Generic python vacuum chamber builder (CF). Arbitrary sphere with arbitrary number of ports (flanges, beam pipes, spherical chamber)
- Pyg4ometry code : 229 lines
- GDML code : 385 lines

Pyg4ometry : Vtk

Geant4 Ray tracer
Geant4 (GDML) example

- Load of the GDML examples distributed with Geant4
- Take a more complex example

```python
[In [1]: import pyg4ometry
FreeCAD 0.17, Libs: 0.17RUnknown

[In [2]: r = pyg4ometry.gdml.Reader("./lht_fixed.gdml")

[In [3]: l = r.getRegistry().getWorldVolume()

[In [4]: v = pyg4ometry.visualisation.VtkViewer()

[In [5]: v.addLogicalVolume(l)

[In [6]: v.view()
```

- Whole file is loaded and can be manipulated in the python terminal
- Dimensions can be changed, holes cut etc.
Important design features (Fluka → Geant4-GDML)

- For each Fluka body create a G4/GDML solid
- Create large but finite G4/GDML solids instead of Fluka infinite solids
- Modify the sizes of bodies to create a length safety between solids
- Create CSG tree from Fluka regions
- Determine if CSG tree creates disjoint solids
- Shrink large solids once extent of Fluka region is determined
- Algorithm uses approximate meshes to perform the calculation and a general solution is not possible
In a very similar way to GDML, classes are created to represent Fluka concepts.

```python
import pyg4ometry.convert as convert
import pyg4ometry.visualisation as vi
from pyg4ometry.fluka import RPP, Region, Zone, FlukaRegistry

def Test(vis=False, interactive=False):
    freg = FlukaRegistry()

    rpp = RPP("RPP_BODY", 0, 10, 0, 10, 0, 10, flukaregistry=freg)
    z = Zone()
    z.addIntersection(rpp)
    region = Region("RPP_REG", material="COPPER")
    region.addZone(z)
    freg.addRegion(region)

    greg = convert.fluka2Geant4(freg)

    greg.getWorldVolume().clipSolid()

    v = None
    if vis:
        v = vi.VtkViewer()
        v.addAxes(length=20)
        v.addLogicalVolume(greg.getWorldVolume())
        v.view(interactive=interactive)

    return {"testStatus": True, "logicalVolume": greg.getWorldVolume(), "vtkViewer":v}
```
Fluka to Geant4-GDML example (Small)

- Magnet created by CERN-RHUL PhD student (Gian Luigi D’Alessandro EN-EA-LE / KLEVER)

```python
In [1]: import pyg4ometry
FreeCAD 0.17, Libs: 0.17RUnknown

In [2]: r = pyg4ometry.fluka.Reader("./QFS_magnet_v8.inp")

In [3]: greg = pyg4ometry.convert.fluka2Geant4(r, flukaregistry)

In [4]: wl = greg.getWorldVolume()

In [5]: v = pyg4ometry.visualisation.VtkViewer()

In [6]: v.addLogicalVolume(wl)

In [7]: v.setOpacity(1)

In [8]: v.setRandomColours()

In [9]: v.view()
```
Fluka to Geant4-GDML example (Medium)

- Shielding created by Maarten van Dijk
- Typically cubical Bodies/solids (so relatively simple test case)
- Large number of simple bodies is relatively easy and saves a lot of time constructing geometry
STL (tessellated solid) example

- Standard Tessellation Language
- Many 3D authoring programmes will produce STL
- Difficult format for GDML

```
[In [1]: import pyg4ometry
FreeCAD 0.17, Libs: 0.17RUnknown
[In [2]: reg = pyg4ometry.geant4.Registry()
[In [3]: r = pyg4ometry.stl.Reader("/utahtapot.stl")
[In [4]: l = r.logicalVolume("test","G4_Cu",reg)
[In [5]: v = pyg4ometry.visualisation.VtkViewer()
[In [6]: v.addLogicalVolume(l)
[In [7]: v.view()
```

- Incorporate MeshCAD or other tools? Currently use CGAL
Load STEP file using FreeCAD-OpenCascade

Still need to simplify CAD file

Bodies and Parts map well to LV and PVs respectively. Convert bodies to triangulated mesh and place

Based on STL loading

Very advanced proof of principle (working in our workflow)

Need to account for material

(what about CadMesh, DagMC, McCAD)
Decompose G4-GDML primitive solids to Fluka bodies, then zones and join into regions

Cut daughter volumes from mother to create flat hierarchy

Working through G4 solids

Need to implement union, intersection and subtraction G4 solids

Least developed area of pyg4ometry but making rapid progress

Scales an issue [-1,1,1] as does not exist in Fluka
Possible to convert larger models at the scale of a small experimental region

Complex solids, CSG trees, booleans

Issue with large CSG trees in G4 converting to Fluka

Fluka geometry in form of disjunctive normal form (DNF) and can see large blow (computation time and memory) up when converting from Geant4
Geant\textsubscript{4}-GDML to Paraview

- Cedric Hernalsteens (CERN), Robin Tesse (ULB)
- Example of proton therapy system from Ion Beam Applications (IBA)
- Another potential target for 3D data is Paraview (built on VTK)
- “Industry standard” for visualisation of 3D data
- Use geometry data from pyg4ometry and output from Geant\textsubscript{4}/Fluka
Have 3D mesh description for Geant4 and Fluka geometry

Have possibility to create Augmented and Virtual reality models of beamlines and detectors

Great potential public engagement and outreach potential

Overlay particles, energy deposits etc. on AR/VR world

Tools are being much more available (Unity, Unreal, USDZ file format)

GL-D’A is also thinking about this
Compositing example (FASER)

- A powerful workflow could be to take geometry from multiple different sources and composite in a GDML file.
- Convert LogicalVolumes to Assembly volumes (excellent for placement).
- Intelligently merge two GDML files (dealing with name clashes).
- Example where Fluka/Geant4/GDML is used.

- Converted tunnel complex (fluka -> geant4).
- BDSIM generic tunnel.
- FASER.
- Shielding blocks.
- BDSIM beam line placed inside all tunnel pieces.
- Custom tunnel prepared using pyg4ometry.
Compositing example (Injector)

- Example using all geometries (CAD, STL, Fluka, Python and GDML)
- Imaginary photo-injector with different components
- Convert LogicalVolumes to Assembly volumes (excellent for placement)
Overlap detection example

Three classes in general

- **Protrusion**
- **Daughter overlap**
- **Coplanar faces (can be a problem)**

First two easily dealt with CSG operations (intersection)

Coplanar faces needs a dedicated algorithm

Search strategy

- Between daughters of LV
- Between daughters and LV solid
Work-flow discussion

- As models get larger a complete conversion of all geometry gets harder
- Need to adapt geometry creation workflow to easily create reusable, maintainable \( G_4 / \) Fluka geometry
  - Capability of pyg4ometry is excellent at the accelerator component level (examples in this talk)
  - How objects are *composited* into a complete model
- Easy to create small applications for use cases
  - ShieldingBuilder
  - LineBuilder (place objects in curvilinear coordinates)
  - MagnetBuilder
  - VacuumChamberBuilder
“Good” house keeping

Testing

- Over 400 unit tests
- 86% test coverage

Manual

- Unit tests are an excellent documentation for features
- Sphinx documentation
- All examples in this presentation (apart from the complex Fluka and CAD) are in Git repository
Potential projects and future directions

- **Pyg4ometry developments**
  - C++ output for compiled code
  - Add “loop” GDML tags
  - GUI for controlling geometry creation
  - Output mode for Unity/Unreal engines for outreach activities
  - Build model hierarchy (LV,PV) from Fluka input
  - Dynamic scene tree update
  - Performance testing (physics comparisons) between Geant4 and Fluka
  - Symbolic expression simplification (GDML equations, Fluka regions?)

- **Geant4 developments**
  - Create VTK based visualizer for Geant4 (might help with solids which cannot be displayed in OpenGL visualization)
  - Contribute to Fluka/Geant4 interoperability projects
  - Update GDML to use different file formats (STL, CAD)
  - Multiple GDML files per Geant4 application (currently only one world volume supported and lots of name collisions)

- **Flair developments**
  - GDML loader for Flair?
Conclusions

• Have developed a relatively powerful geometry manipulation tool (kit)
• Uses most up to date software packages and modern programming language
• Generic conversion back and forth between different tools is quite possible but probably not how a tool like this will be used (area of discussion)
• Need testing and refinement on larger and more complex models to home the algorithms and tests
• Potential to save a lot of user’s time with generating geometry
• Programmatic interface to geometry creation and manipulation, lots of potential for different use cases we have not thought about
• Materials between different codes still needs some work (grey pyg4ometry images in this talk are because materials are not set)
• Need to test more models, run timing tests in Geant4/Fluka