pyg4ometry Documentation

Release 0.1

Royal Holloway

Jul 16, 2018

CONTENTS

pyg4ometry is a package to create, load, write and visualise solid geometry for particle tracking simulations.

CHAPTER

ONE

LICENCE & DISCLAIMER

CHAPTER TWO

AUTHORSHIP

The following people have contributed to pyg4ometry:

- Stewart Boogert
- Andrey Abramov
- Alistair Butcher
- Stuart Walker
- Laurie Nevay

CHAPTER

THREE

INSTALLATION

3.1 Requirements

- pyg4ometry is developed exclusively for Python 2.7.
- VTK (Visualisation toolkit)

3.2 Installation

To install pyg4ometry, simply run `make install` from the root pyg4ometry directory.:

```
cd /my/path/to/repositories/
git clone http://bitbucket.org/jairhul/pyg4ometry
cd pyg4ometry
make install
```

Or install from pypi.:

```
pip install pyg4ometry
```

or alternatively, run `make develop` from the same directory to ensure that any local changes are picked up.

CHAPTER FOUR

INTRODUCTION

4.1 Need for programatic geometry generation

- Non-expert user creation and maintainence of geometry
- Reduce time spent creating geometry
- Reproducibility
- Lower number of errors
- Parametrisation of geometry
- Visualisation of geometry
- Overlap checking
- Import from other geometry packages

4.2 Geant4 key concepts

- GMDL

4.3 Geometry key concepts

- Constructive Solid Geometry (CSG)
- Boolean operations
- Boundary representation (B-REP)
- Boundary mesh

4.4 Implementation concepts

- Registry
- Parameter
- ParameterVector
- Pycsg

CHAPTER

FIVE

TUTORIALS

5.1 Geant4 python scripting

Making use of pyg4ometry requires the following modules

```
import pyg4ometry      as pyg  # geant4
import pyg4ometry.vtk  as vtk  # vistualisation tool kit commands
import pyg4ometry.gdml as gdml # gdml io
```

The standard math and numpy modules are also very useful

```
import numpy as np
import math  as math
```

To make a simple geometery of a boolean subtraction solid (a cube with a cylinder removed in the centre)

A triangular mesh is generated from any physical `volume` by the following command

```
m = volume.pycsgmesh()
```

Given an output from `m = volume.pycsgmesh()` it can be viewed in the `vtk` viewer with the following example

```
v = pyg4ometry.vtk.Viewer()
v.addPycsgMeshList(m)
v.view();
```

To write an STL file from `m = volume.pycshmesh()`

```
vtkConverter = vtk.Convert()
vtkPD        =  vtkConverter.MeshListToPolyData(m)
r = vtk.WriteSTL("file.stl",vtkPD)
```

To write an GDML file file

```
w = _gdml.Writer()
w.addDetector(pyg.geant4.registry)
w.write('./file.gdml')
w.writeGmadTester('./file.gmad')
```

5.2 GDML input

5.3 STL input

5.4 STEP/STP input

5.5 FLUKA input

5.6 STL output

CHAPTER

SIX

MODULE CONTENTS

This documentation is automatically generated by scanning all the source code. Parts may be incomplete.

6.1 Geant4 module

Geant4 classes. The classes mainly match those of Geant4

class pyg4ometry.geant4.PhysicalVolume.**PhysicalVolume**(*rotation, position, logicalVolume, name, motherVolume, scale=[1, 1, 1], debug=False, register=True*)

Bases: object

Geant4 Physical volume class

gdmlWrite(*gw, prepend*)

imeshed = 0

pycsgmesh()

pyg4ometry.geant4.PhysicalVolume.**recursive_map_size**(*nlist*)

Recursive application of .polygonCount() and .vertexCount() to meshlist :argument: nlist

pyg4ometry.geant4.PhysicalVolume.**recursize_map_rottrans**(*nlist, trans, rot, scale=[1, 1, 1]*)

Function to apply transformation (rotation then translatoin) to nested list of meshes (nlist)

class pyg4ometry.geant4.LogicalVolume.**LogicalVolume**(*solid, material, name, debug=False, register=True*)

Bases: object

add(*physicalVolume*)

gdmlWrite(*gw, prepend*)

getSize()

imeshed = 0

pycsgmesh()

setCentre(*centre*)

setClip(*centre=True, tolerance=None*)

setSize(*size*)

pyg4ometry.geant4.LogicalVolume.**mesh_extent**(*nlist*)

Function to determine extent of an tree of meshes pyg4ometry.geant4.LogicalVolume.**pycsg_overlap**(*meshTree*, *worldVolumeIncluded=True*)

Function to determine if there overlaps of meshes. If the mesh list is generated by recursively meshing the world volume, the first mesh in the list (the world box) is ignored as it overlaps with everything.

class pyg4ometry.geant4.Registry.**Registry**

addDefine(*define*)

addDefinition(*definition*)

addLogicalVolume(*volume*)

addMaterial(*material*)

addParameter(*parameter*)

addParameterisedVolume(*volume*)

addPhysicalVolume(*volume*)

addReplicaVolume(*volume*)

addSolid(*solid*)

clear()

Empty all internal structures

orderLogicalVolumes(*lvName*)

Need to have a ordered list from most basic (solid) object upto physical/logical volumes for writing to GDML. GDML needs to have the solids/booleans/volumes defined in order

setWorld(*worldName*)

solidTree(*solidName*)

Not sure what this method is used for

volumeTree(*lvName*)

Not sure what this method is used for

class pyg4ometry.geant4.Parameter.**Parameter**(*name*, *value*, *addRegistry=True*)

Bases: object

str()

class pyg4ometry.geant4.ParameterVector.**ParameterVector**(*name*, *vlist=[]*, *addRegistry=True*)

Bases: list

str()

6.2 Geant4 solids

class pyg4ometry.geant4.solid.Plane.**Plane**(*name*, *normal*, *dist*, *zlength=10000*)

Constructs a *infinite* plane. Should not be used to construct geant4 geometry.

Parameters

- **name** (*str*) – of object in registry
- **normal** (*tuple*) – normal [x,y,z]
- **dist** (*float*) – distance from origin to plane
- **zlength** (*float*) – large transverse box size to emulate infinite plane

class pyg4ometry.geant4.solid.Wedge.**Wedge**(*name*, *pRMax=1000*, *pSPhi=0*, *pDPhi=1.5*, *halfzlength=10000*)

Constructs a *infinite* wedge. Should not be used to construct geant4 geometry.

Parameters

- **name** (*str*) – of object in registry
- **normal** (*tuple*) – normal [x,y,z]
- **dist** (*float*) – distance from origin to plane
- **zlength** (*float*) – large transverse box size to emulate infinite plane

class pyg4ometry.geant4.solid.Box.**Box**(*name=", pX=0.0, pY=0.0, pZ=0.0, register=True*)

Constructs a box.

Parameters

- **name** (*float*) – of object in registry
- **pX** – half-length along x
- **pY** – half-length along y
- **pZ** – half-length along z

class pyg4ometry.geant4.solid.Tubs.**Tubs**(*name, pRMin, pRMax, pDz, pSPhi, pDPhi, nslice=16, register=True*)

Constructs a cylindrical section.

Parameters

- **name** (*str*) – of object in registry
- **pRMin** (*float*) – inner radius
- **pRMax** (*float*) – outer radius
- **pDz** (*float*) – half-length along z
- **pSPhi** (*float*) – starting phi angle
- **pDPhi** (*float*) – angle of segment in phi

class pyg4ometry.geant4.solid.CutTubs.**CutTubs**(*name, pRMin, pRMax, pDz, pSPhi, pDPhi, pLowNorm, pHighNorm, register=True*)

Constructs a cylindrical section with cuts.

Inputs: name: string, name of the volume pRMin: float, inner radius pRMax: float, outer radius pDz: float, half-length along z pSPhi: float, starting phi angle pDPhi: float, angle of segment in radians pLowNorm: list, normal vector of the cut plane at -pDz pHighNorm: list, normal vector of the cut plane at +pDz

class pyg4ometry.geant4.solid.Sphere.**Sphere**(*name, pRmin, pRmax, pSPhi, pDPhi, pSTheta, pDTheta, nslice=10, nstack=10, register=True*)

Constructs a section of a spherical shell.

Parameters

- **name** (*str*) – of object in registry
- **pRmin** (*float*) – inner radius of the shell
- **pRmax** (*float*) – outer radius of the shell
- **pSPhi** (*float*) – starting phi angle in radians
- **pSTheta** (*float*) – starting theta angle in radians
- **pDPhi** (*float*) – delta phi angle in radians
- **pDTheta** (*float*) – delta theta angle in radians

class pyg4ometry.geant4.solid.Cons.**Cons**(*name*, *pRmin1*, *pRmax1*, *pRmin2*, *pRmax2*, *pDz*, *pSPhi*, *pDPhi*, *register=True*)

Constructs a conical section.

Inputs: name: string, name of the volume pRMin1: float, inner radius at -pDz pRMax1: float, outer radius at -pDz pRMin2: float, inner radius at +pDZ pRMax2: float, outer radius at +pDz pDz: float, half-length along z pSPhi: float, starting phi angle pDPhi: float, angle of segment in radians

class pyg4ometry.geant4.solid.Trd.**Trd**(*name*, *pDx1*, *pDx2*, *pDy1*, *pDy2*, *pDz*, *register=True*)

Constructs a trapezoid.

Inputs: name: string, name of the volume pDx1: float, half-length along x at the surface positioned at -dz pDx2: float, half-length along x at the surface postitioned at +dz pDy1: float, half-length along y at the surface positioned at -dz pDy2: float, half-length along y at the surface positioned at +dz dz: float, half-length along the z axis

class pyg4ometry.geant4.solid.Union.**Union**(*name*, *obj1*, *obj2*, *tra2*, *register=True*)

name = name obj1 = unrotated, untranslated solid obj2 = solid rotated and translated according to tra2 tra2 = [rot,tra] = [[a,b,g],[dx,dy,dz]]

class pyg4ometry.geant4.solid.Intersection.**Intersection**(*name*, *obj1*, *obj2*, *tra2*, *register=True*)

name = name obj1 = unrotated, untranslated solid obj2 = solid rotated and translated according to tra2 tra2 = [rot,tra] = [[a,b,g],[dx,dy,dz]]

class pyg4ometry.geant4.solid.Subtraction.**Subtraction**(*name*, *obj1*, *obj2*, *tra2*, *register=True*)

output = obj1 - obj2 name = name obj1 = unrotated, untranslated solid obj2 = solid rotated and translated according to tra2 tra2 = [rot,tra] = [[a,b,g],[dx,dy,dz]]

class pyg4ometry.geant4.solid.Para.**Para**(*name*, *pDx*, *pDy*, *pDz*, *pAlpha*, *pTheta*, *pPhi*, *register=True*)

Constructs a parallelepiped.

Inputs: name: string, name of the volume pX: float, half-length along x pY: float, half-length along y pZ: float, half-length along z pAlpha: float, angle formed by the y axis and the plane joining the centres of the faces parallel tothe z-x plane at -dy and +dy pTheta: float, polar angle of the line joining the centres of the faces at -dz and +dz in z pPhi: float, azimuthal angle of the line joining the centres of the faces at -dx and +dz in z

6.3 VTK module

class pyg4ometry.vtk.Viewer.**Viewer**

addPycsgMesh(*m*, *refine=True*)

addPycsgMeshList(*meshes*, *refine=True*)

setAxes()

view()

pyg4ometry.vtk.Viewer.**mkVtkIdList**(*it*)

class pyg4ometry.vtk.Convert.**Convert**

MeshListToPolyData(*meshes*)

pyg4ometry.vtk.Convert.**VerticesAndPolygonsToPolyData**(*m*)

pyg4ometry.vtk.Convert.**mkVtkIdList**(*it*) `pyg4ometry.vtk.Writer.`**`WriteSTL`**(*fileName*, *meshes*)

meshes : list of triFilters

6.4 Freecad module

6.5 STL module

`class` `pyg4ometry.stl.Reader.`**`Reader`**(*filename*, *solidname='tess'*, *visualise=True*, *writeGDML=False*, *scale=1*)

Bases: `object`

`load`(*solidname='tess'*, *visualise=False*, *writeGDML=False*)

6.6 GDML module

`class` `pyg4ometry.gdml.Reader.`**`Reader`**(*filename*)

Bases: `object`

`load`()

`parseDefines`(*xmldoc*)

`parseMaterials`(*xmldoc*)

`parseSolids`(*xmldoc*)

`parseStructure`(*xmldoc*)

`stringAlgebraicSplit`(*string*)

`class` `pyg4ometry.gdml.Writer.`**`Writer`**(*prepend='PREPEND'*)

Bases: `object`

`addDetector`(*registry*)

`checkDefineName`(*defineName*)

`checkLogicalVolumeName`(*logicalVolumeName*)

`checkMaterialName`(*materialName*)

`checkPhysicalVolumeName`(*physicalVolumeName*)

`checkSolidName`(*solidName*)

`createPosition`(*name*, *x*, *y*, *z*)

`createSection`(*zOrder*, *zPosition*, *xOffset*, *yOffset*, *scalingFactor*)

`createTriangularFacet`(*vertex1*, *vertex2*, *vertex3*)

`createTwoDimVertex`(*x*, *y*)

`createrzPoint`(*r*, *z*)

`createzPlane`(*rInner*, *rOuter*, *zplane*)

`extractDefinesFromTesselatedSolids`(*registry*)

`write`(*filename*)

`writeBox`(*instance*)

`writeCons`(*instance*)

`writeCutTubs`(*instance*)

writeDefaultLattice(*filename='lattice.gmad'*)

writeEllipsoid(*instance*)

writeEllipticalCone(*instance*)

writeEllipticalTube(*instance*)

writeExtrudedSolid(*instance*)

writeGenericPolycone(*instance*)

writeGmadTester(*filenameGmad*, *writeDefaultLattice=False*, *zLength=None*)

writeHype(*instance*)

writeIntersection(*instance*)

writeMaterial(*material*)

writeOpticalSurface(*instance*)

writeOrb(*instance*)

writePara(*instance*)

writeParaboloid(*instance*)

writeParameter(*param*)

writePolycone(*instance*)

writePolyhedra(*instance*)

writeSolid(*solid*)

Dispatch to correct member function based on type string in SolidBase.

writeSphere(*instance*)

writeSubtraction(*instance*)

writeTesselatedSolid(*instance*)

writeTet(*instance*)

writeTorus(*instance*)

writeTrap(*instance*)

writeTrd(*instance*)

writeTubs(*instance*)

writeTwistedBox(*instance*)

writeTwistedTrap(*instance*)

writeTwistedTrd(*instance*)

writeUnion(*instance*)

6.7 Fluka module

6.8 Test module

CHAPTER

SEVEN

TODO

7.1 2018 / 06 / 04

- start sphinx documentation (with simple examples)
- Simple boolean geometry
- STL mesh load
- GDML load
- check for name collisions in registry
- throw error if same name appears on add
- proper gmad element length
- integrate pyfluka
- materials
- optical surfaces
- vtk overlap
- vtk execute overlap test
- check g4 tests (mainly for timing)
- use gmsh for mesh optimisation

CHAPTER EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX