PROGRAMMATIC GEOMETRY PREPARATION FOR MONTE CARLO RADIATION TRANSPORT IN BEAMLINES

THE JAI FESTIVAL 2020

STUART WALKER 11/12/2020

ROYAL HOLLOWAY UNIVERSITY OF LONDON

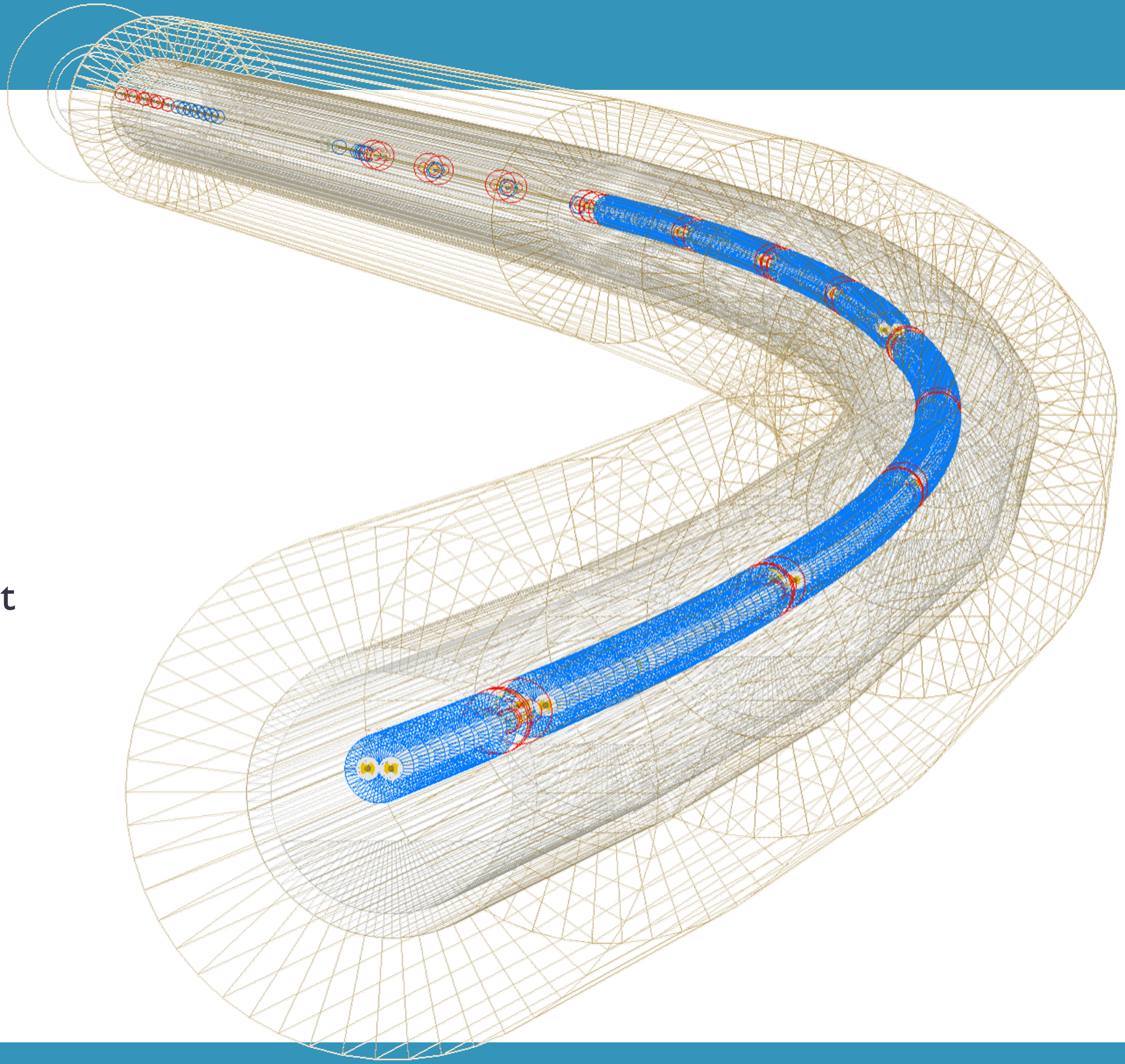
JAI John Adams Institute for Accelerator Science

BDSIM Beam Delivery Simulation

GEANT4 A SIMULATION TOOLKIT

INTRODUCTION

- Beam losses in particle accelerators.
- Beam Delivery Simulation and building Monte Carlo models to study beam losses.
- Pyg4ometry: programmatic radiation transport geometry building using Python
- Applications and future work.

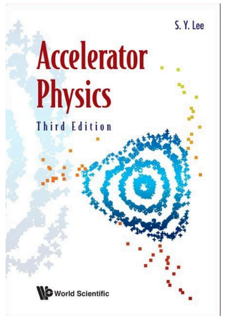


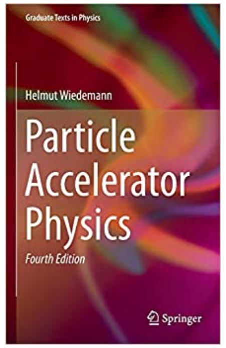
MOTIVATION

- Beam losses in any beamline are unavoidable.
- Need radiation transport codes to understand beam losses:
  - Minimize heating of superconducting elements.
  - Minimize backgrounds in experiments.
  - Optimizing shielding and dosage calculations (e.g., for a patient!).
- Typically, one will use FLUKA or Geant4 in HEP applications.
  - Others include MCNP, MARS and PENELOPE. Different codes, different capabilities.
- Specific applications of Geant4 to particle accelerators include BDSIM (developed at RHUL :)) and G4Beamline.

FLUKA

mcnp

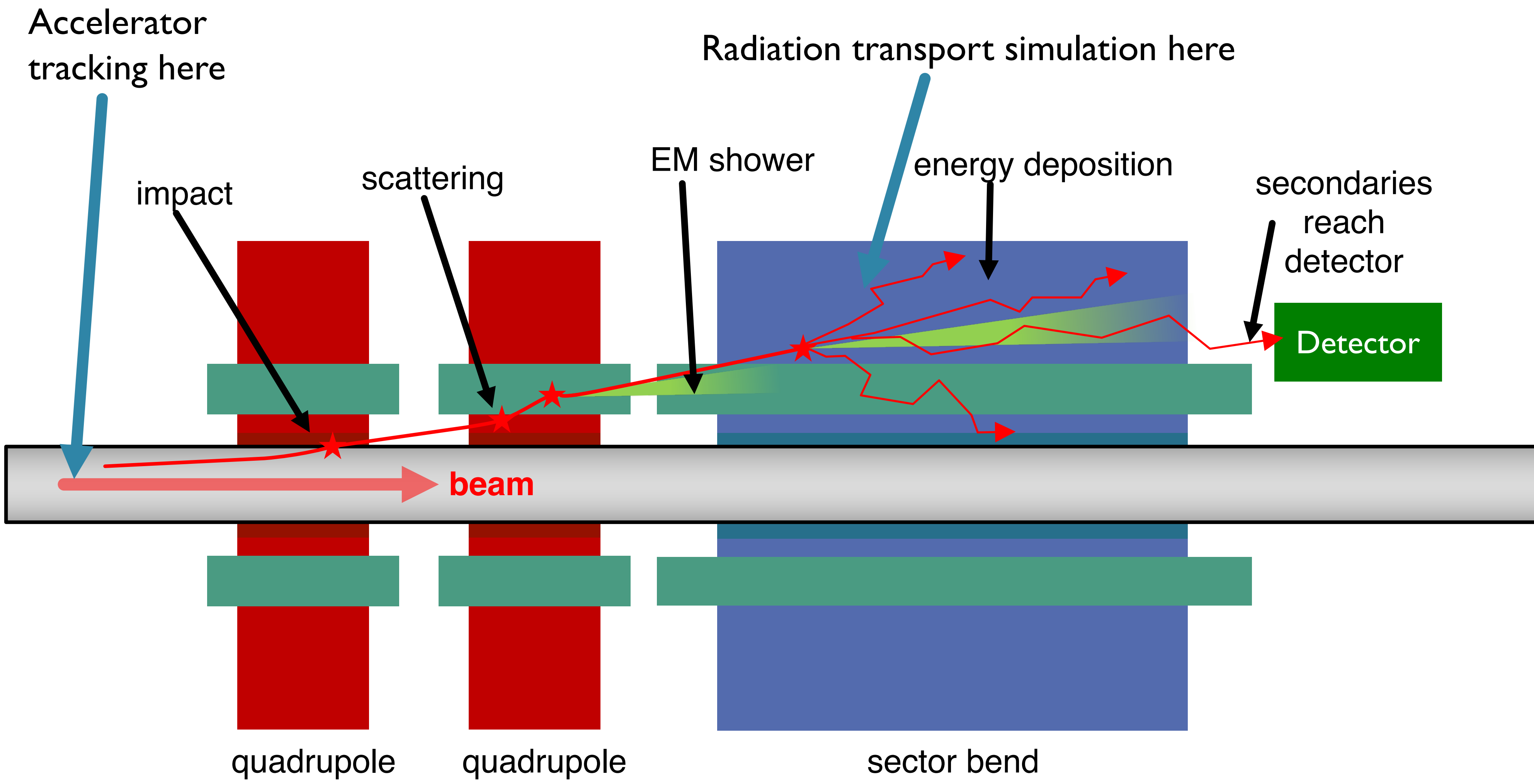




+ GEANT4 A SIMULATION TOOLKIT = BDSIM Beam Delivery Simulation

BEAM LOSSES

- Particles don't just stop when impacting upon material.
- Will generally travel downstream for some distance.
- Depositing energy along the beamline, some of it possibly sensitive or cold.
- Secondaries possibly picked up in detectors.

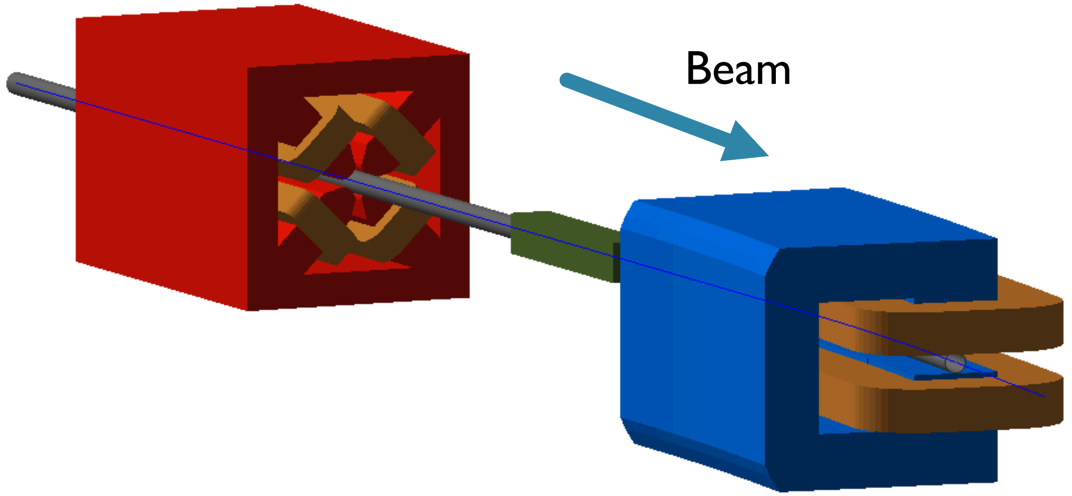

Accelerator tracking here
Radiation transport simulation here
impact
scattering
EM shower
energy deposition
secondaries reach detector
Detector
beam
quadrupole
quadrupole
sector bend

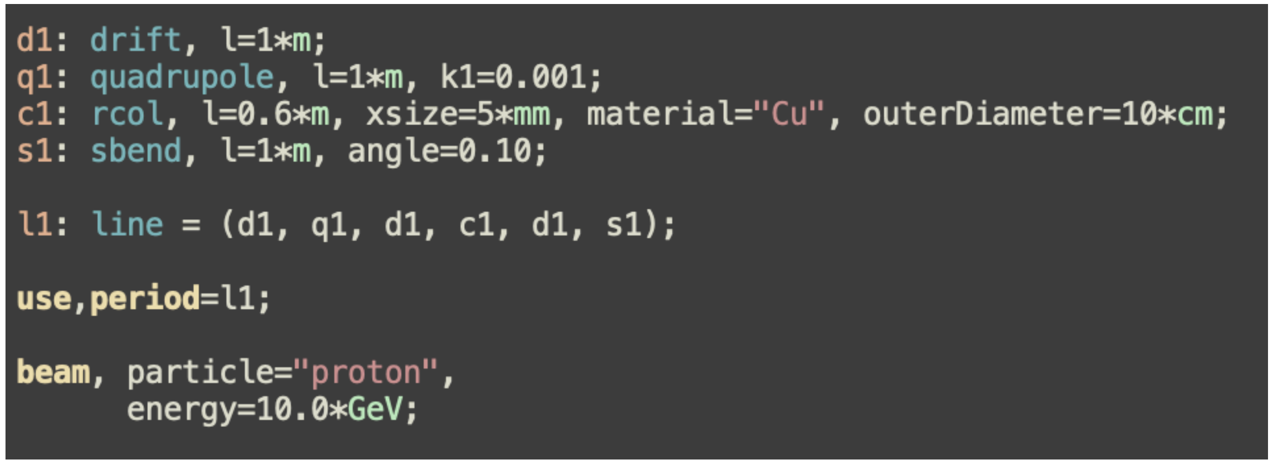
BEAM DELIVERY SIMULATION (BDSIM)

- Automatic Geant4 accelerator models.
- Combine particle physics of Geant4 with accelerator tracking routines.
- Many applications to study of beam losses and experimental backgrounds.
- Simple MAD-X-style input.
- Developed at RHUL over the past 15 years.

BDSIM: An accelerator tracking code with particle-matter interactions, Computer Physics Communications, 2020.

https://doi.org/10.1016/j.cpc.2020.107200



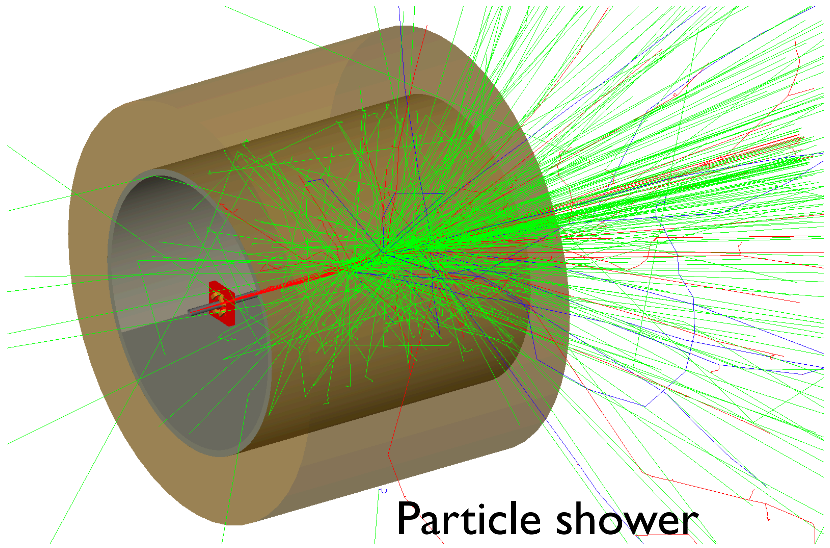


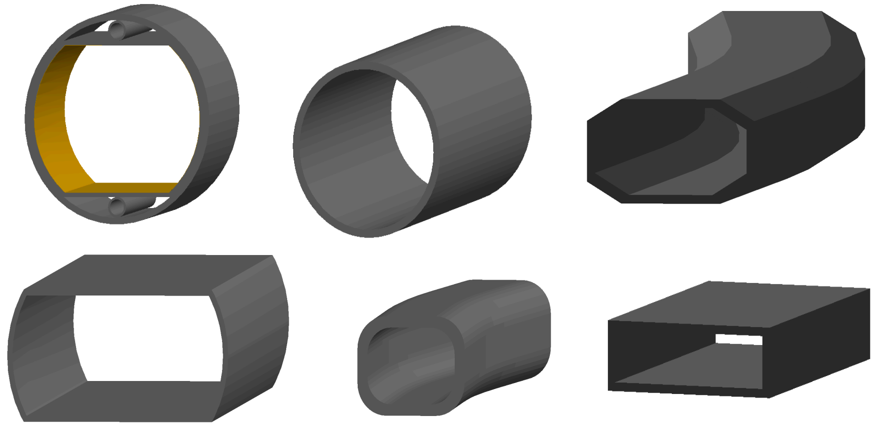
```
d1: drift, l=1*m;
q1: quadrupole, l=1*m, k1=0.001;
c1: rcol, l=0.6*m, xsize=5*mm, material="Cu", outerDiameter=10*cm;
s1: sbend, l=1*m, angle=0.10;

l1: line = (d1, q1, d1, c1, d1, s1);

use,period=l1;

beam, particle="proton",
      energy=10.0*GeV;
```

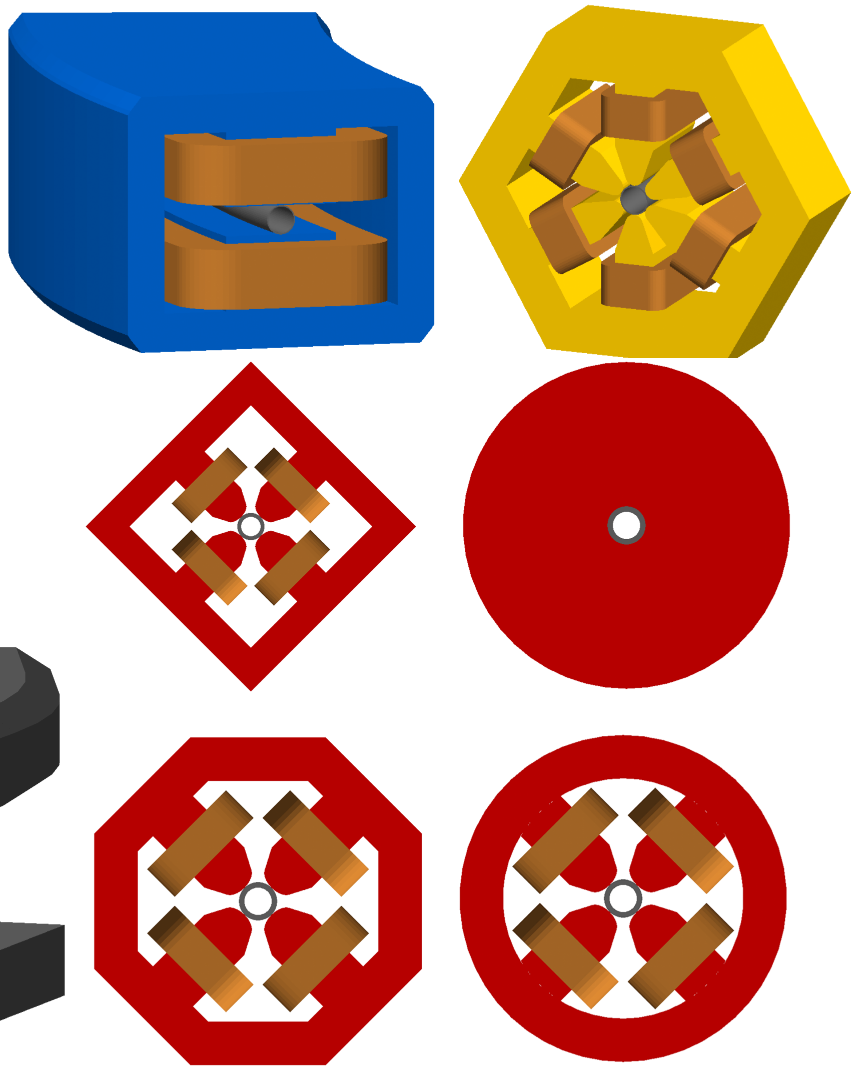
Simple machine textual description with resulting Geant4 model in BDSIM.

BDSIM (CONT.)

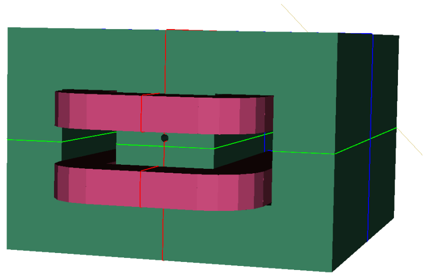
- Fully customizable geometries (both predefined component styles and custom GDML) and fields.
- Full event-level output (ROOT) storage enabling full access to features within the event.
  - Precisely identify source of backgrounds or energy deposition, e.g. position in phase space or loss point.
- All of Geant4's validated physics easily applied to beamlines.


Particle shower

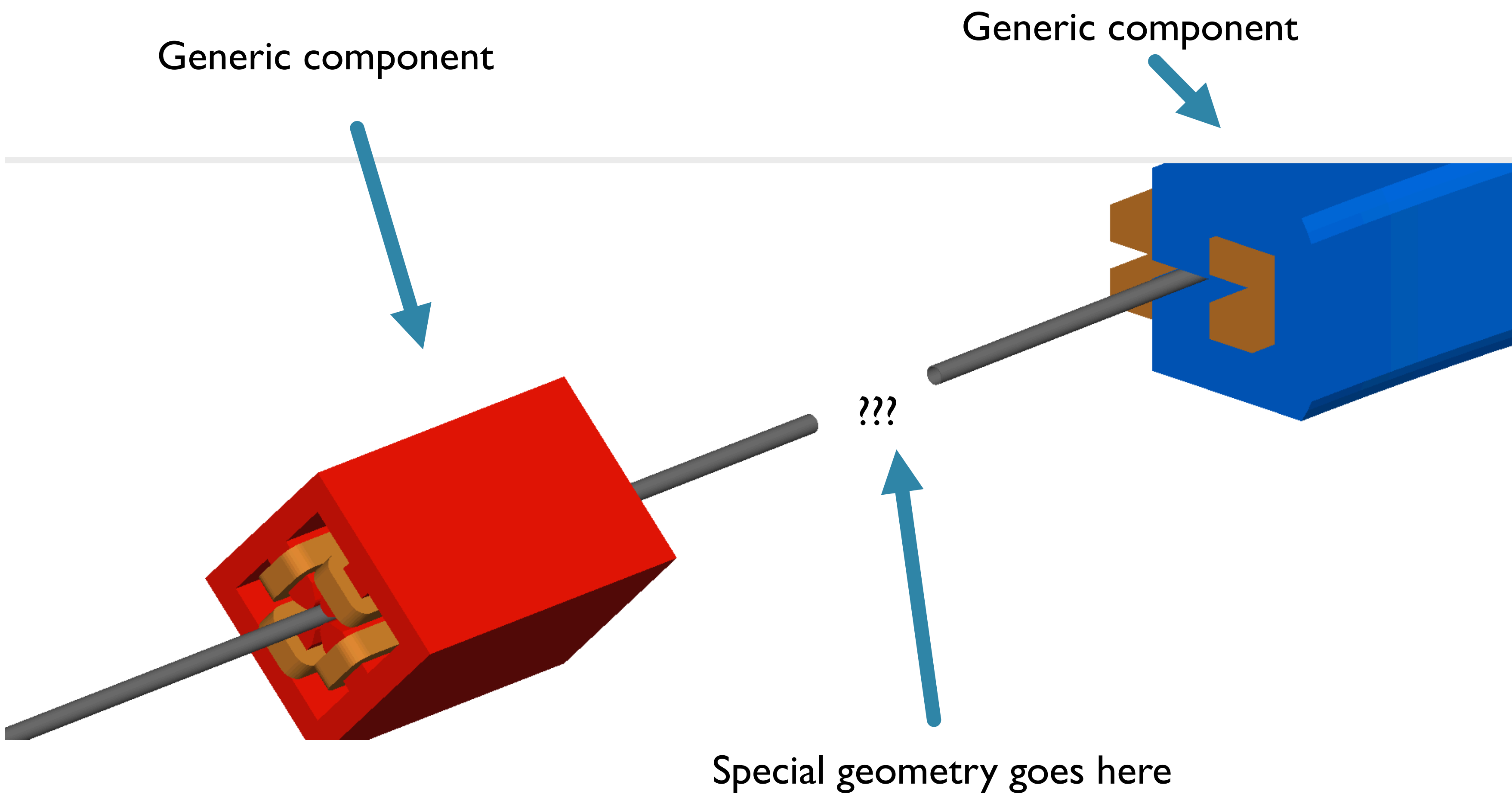

Aperture types


Quadrupole yoke styles

CUSTOM GEOMETRY

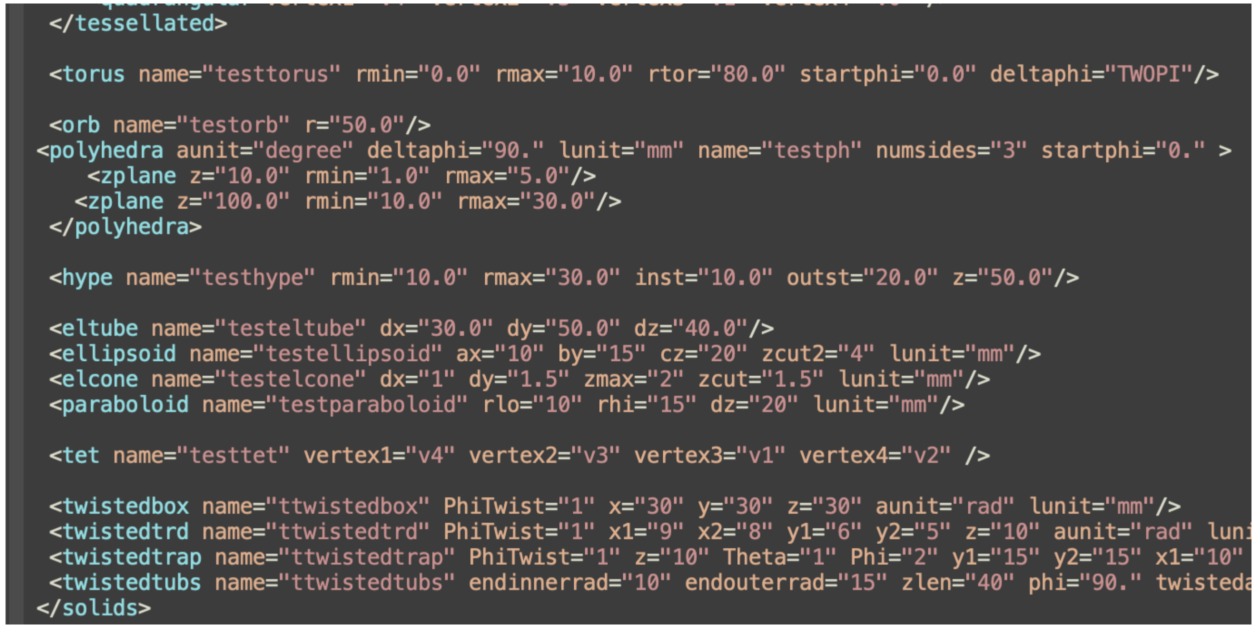


- Automating the building of Monte Carlo radiation transport geometry is one solution.
- What about when it is not enough?
  - Shielding design.
  - Special components.
  - Targets.



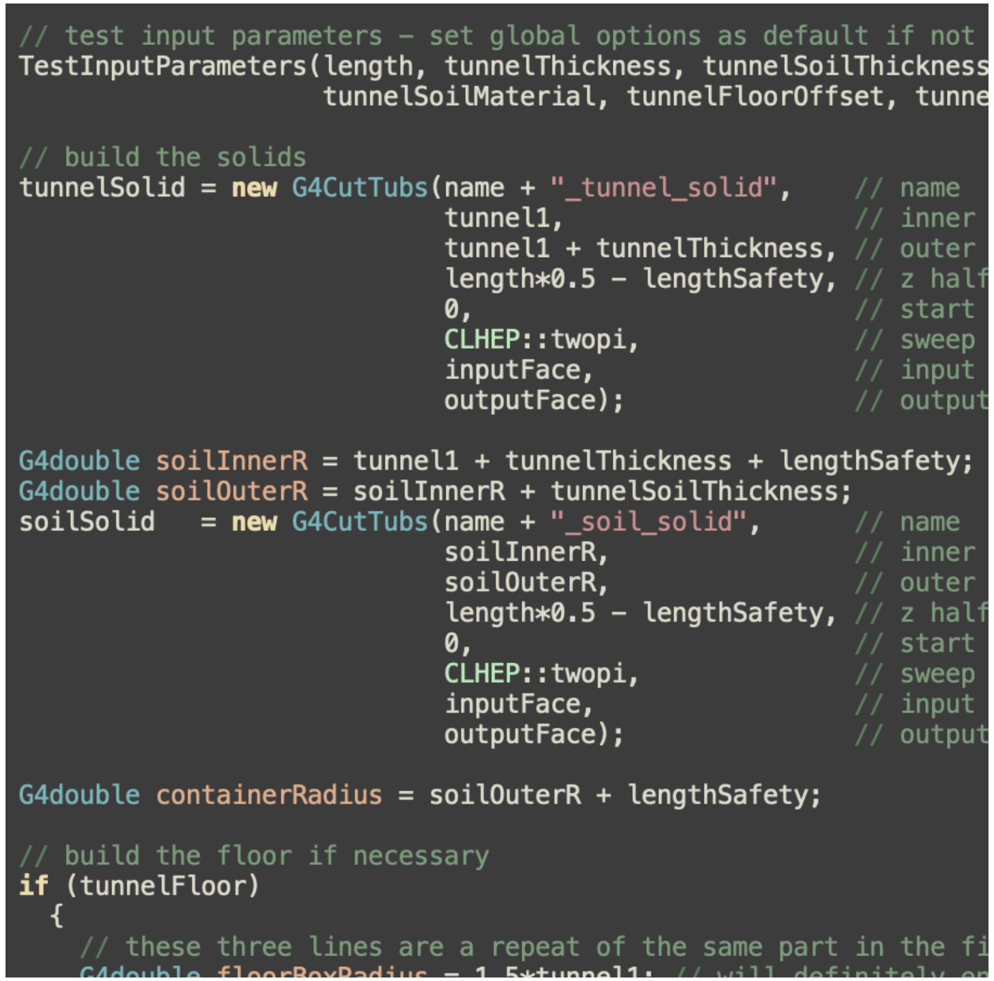
DEVELOPING BESPOKE GEOMETRY FOR GEANT4

- Implement the geometry directly in C++ or use Geant4's persistency format GDML.
- These solutions can be cumbersome and difficult to get right.
  - Compile times and nonconforming GDML



```
  </tessellated>

  <torus name="testtorus" rmin="0.0" rmax="10.0" rtor="80.0" startphi="0.0" deltaphi="TWOPI"/>

  <orb name="testorb" r="50.0"/>
 <polyhedra aunit="degree" deltaphi="90." lunit="mm" name="testph" numsides="3" startphi="0." >
     <zplane z="10.0" rmin="1.0" rmax="5.0"/>
    <zplane z="100.0" rmin="10.0" rmax="30.0"/>
  </polyhedra>

  <hype name="testhype" rmin="10.0" rmax="30.0" inst="10.0" outst="20.0" z="50.0"/>

  <eltube name="testeltube" dx="30.0" dy="50.0" dz="40.0"/>
  <ellipsoid name="testellipsoid" ax="10" by="15" cz="20" zcut2="4" lunit="mm"/>
  <elcone name="testelcone" dx="1" dy="1.5" zmax="2" zcut="1.5" lunit="mm"/>
  <paraboloid name="testparaboloid" rlo="10" rhi="15" dz="20" lunit="mm"/>

  <tet name="testtet" vertex1="v4" vertex2="v3" vertex3="v1" vertex4="v2" />

  <twistedbox name="ttwistedbox" PhiTwist="1" x="30" y="30" z="30" aunit="rad" lunit="mm"/>
  <twistedtrd name="ttwistedtrd" PhiTwist="1" x1="9" x2="8" y1="6" y2="5" z="10" aunit="rad" luni
  <twistedtrap name="ttwistedtrap" PhiTwist="1" z="10" Theta="1" Phi="2" y1="15" y2="15" x1="10"
  <twistedtubs name="ttwistedtubs" endinnerrad="10" endouterrad="15" zlen="40" phi="90." twisteda
 </solids>
```

Declarative XML-based GDML

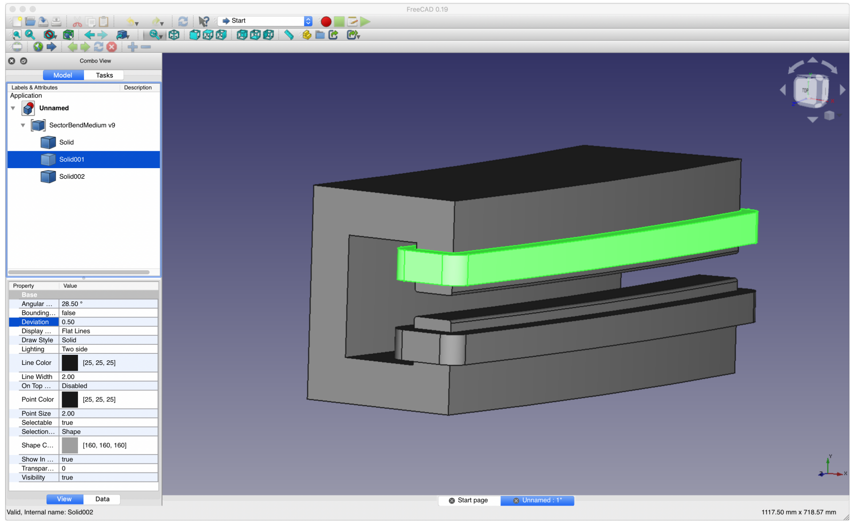


```
// test input parameters - set global options as default if not
TestInputParameters(length, tunnelThickness, tunnelSoilThickness
                    tunnelSoilMaterial, tunnelFloorOffset, tunne

// build the solids
tunnelSolid = new G4CutTubs(name + "_tunnel_solid",    // name
                            tunnel1,                   // inner
                            tunnel1 + tunnelThickness, // outer
                            length*0.5 - lengthSafety, // z half
                            0,                         // start
                            CLHEP::twopi,              // sweep
                            inputFace,                 // input
                            outputFace);               // output

G4double soilInnerR = tunnel1 + tunnelThickness + lengthSafety;
G4double soilOuterR = soilInnerR + tunnelSoilThickness;
soilSolid   = new G4CutTubs(name + "_soil_solid",      // name
                            soilInnerR,                // inner
                            soilOuterR,                // outer
                            length*0.5 - lengthSafety, // z half
                            0,                         // start
                            CLHEP::twopi,              // sweep
                            inputFace,                 // input
                            outputFace);               // output

G4double containerRadius = soilOuterR + lengthSafety;

// build the floor if necessary
if (tunnelFloor)
  {
    // these three lines are a repeat of the same part in the fi
    G4double floorBoxRadius = 1.5*tunnel1; // will definitely en
```

Geant4 geometry in compiled C++

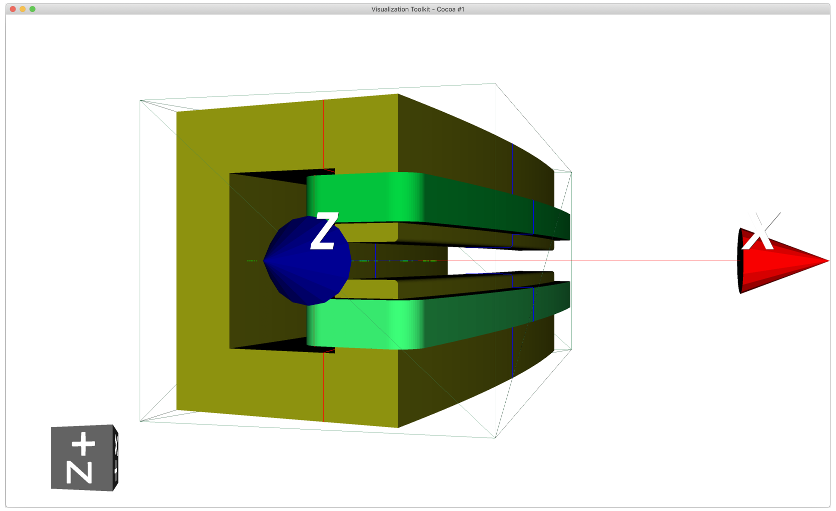
LOADING CUSTOM GEOMETRY IN GEANT4 FROM CAD

- A good option is to directly use CAD geometry in the Monte Carlo simulation.
- However, these can often be poorly-suited for such simulations (bugs, overlaps, missing materials), and require additional preprocessing.
  - Would benefit from a set of utilities to make this easier.

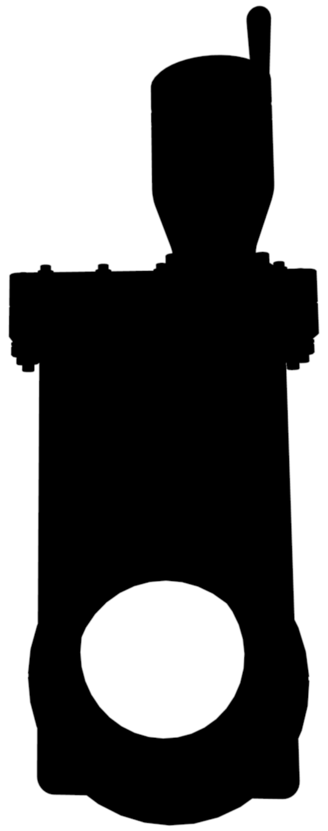


Property	Value
Base	
Angular …	28.50 °
Bounding…	false
Deviation	0.50
Display …	Flat Lines
Draw Style	Solid
Lighting	Two side
Line Color	[25, 25, 25]
Line Width	2.00
On Top …	Disabled
Point Color	[25, 25, 25]
Point Size	2.00
Selectable	true
Selection…	Shape
Shape C…	[160, 160, 160]
Show In …	true
Transpar…	0
Visibility	true

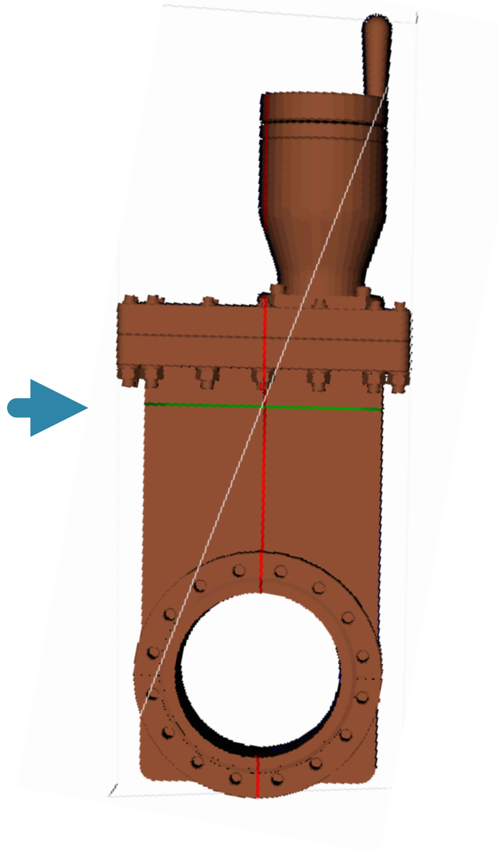
Valid, Internal name: Solid002



FreeCAD (above), Geant4 (bottom)



Standard Tessellated Format (STL)



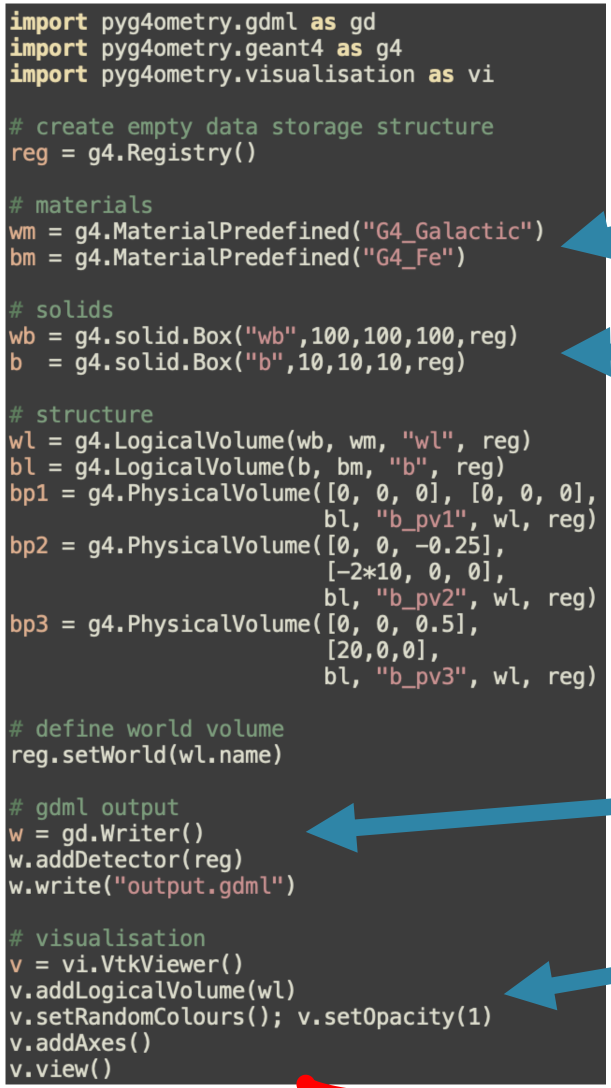
Geant4

RATIONALE

- Geometry preparation for Monte Carlo (MC) simulations is difficult.
- Develop a general-purpose Python API for writing, visualizing and debugging Monte Carlo Radiation Transport geometry.
- Read and write many different MC formats as possible for maximum flexibility.
- Maximize use of extremely mature open-source libraries in Python.
- More physicists know Python than any other language.

FreeCAD

PYTHON GEOMETRY SCRIPTING USING PYG4OMETRY



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

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

# materials
wm = g4.MaterialPredefined("G4_Galactic")
bm = g4.MaterialPredefined("G4_Fe")

# solids
wb = g4.solid.Box("wb",100,100,100,reg)
b  = g4.solid.Box("b",10,10,10,reg)

# structure
wl = g4.LogicalVolume(wb, wm, "wl", reg)
bl = g4.LogicalVolume(b, bm, "b", reg)
bp1 = g4.PhysicalVolume([0, 0, 0], [0, 0, 0],
                        bl, "b_pv1", wl, reg)
bp2 = g4.PhysicalVolume([0, 0, -0.25],
                        [-2*10, 0, 0],
                        bl, "b_pv2", wl, reg)
bp3 = g4.PhysicalVolume([0, 0, 0.5],
                        [20,0,0],
                        bl, "b_pv3", wl, reg)

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

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

# visualisation
v = vi.VtkViewer()
v.addLogicalVolume(wl)
v.setRandomColours(); v.setOpacity(1)
v.addAxes()
v.view()
```

Geometry scripting using Python

Materials

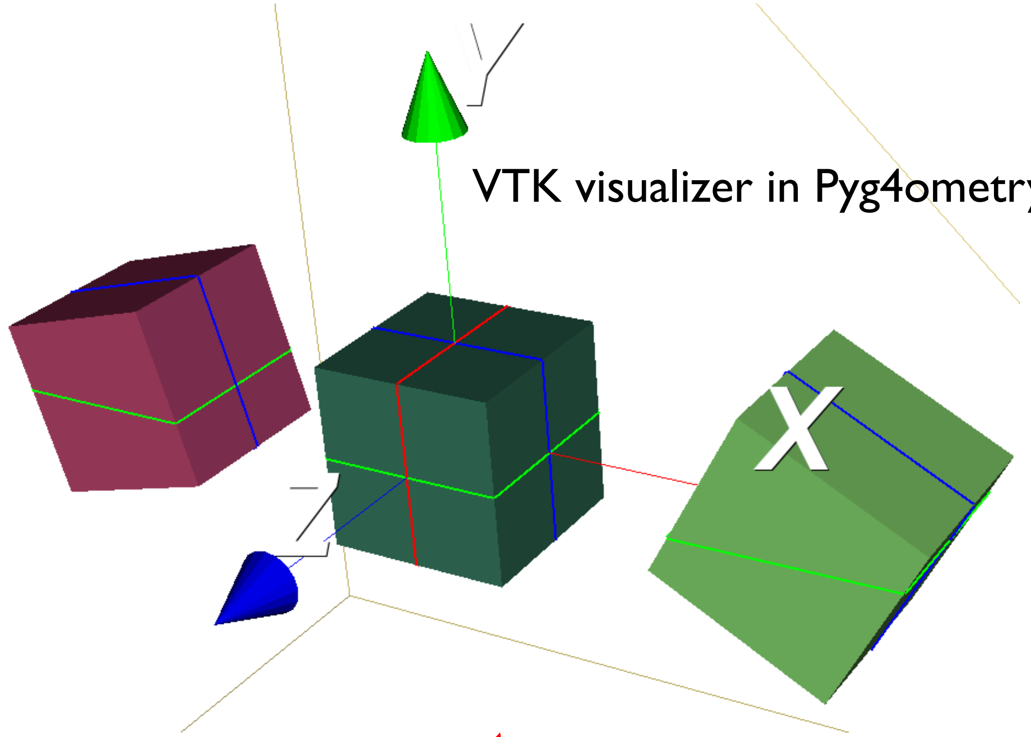
Solids

Geometry hierarchy definition

Write to GDML

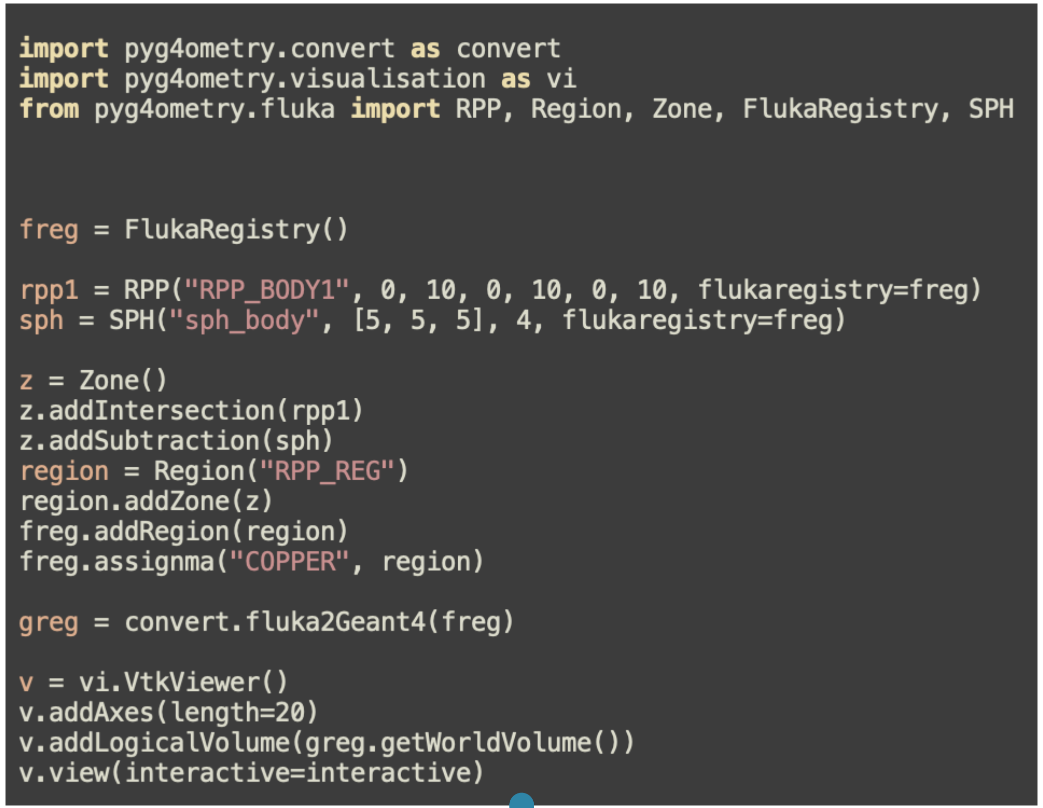
Visualise (shown right)

VTK visualizer in Pyg4ometry

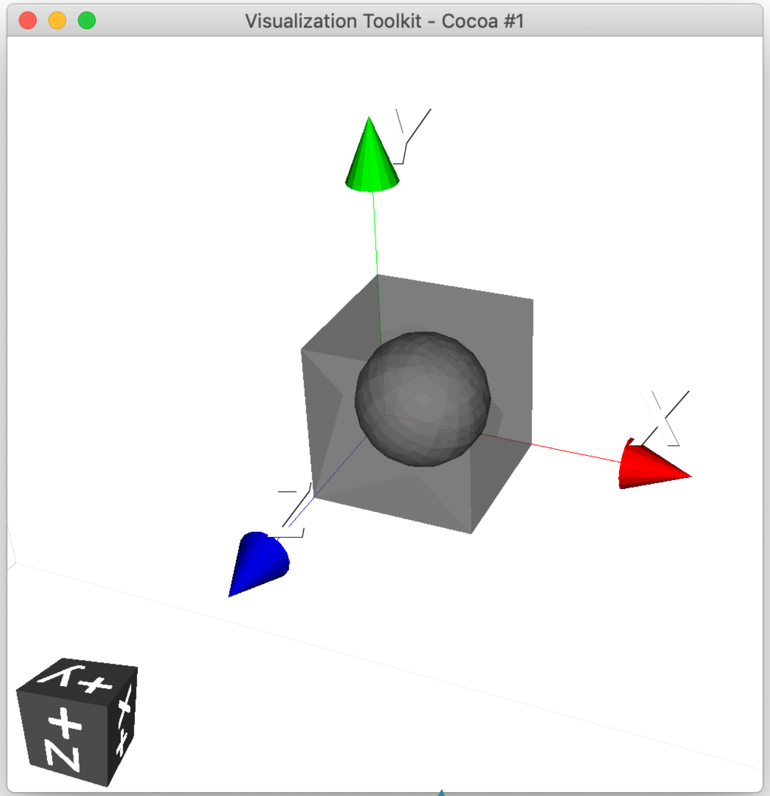


FLUKA SCRIPTING

- Also support pure Python FLUKA scripting.
- Enables programmatic manipulation of FLUKA geometries: previously not possible.
- Includes conversion to and from FLUKA.

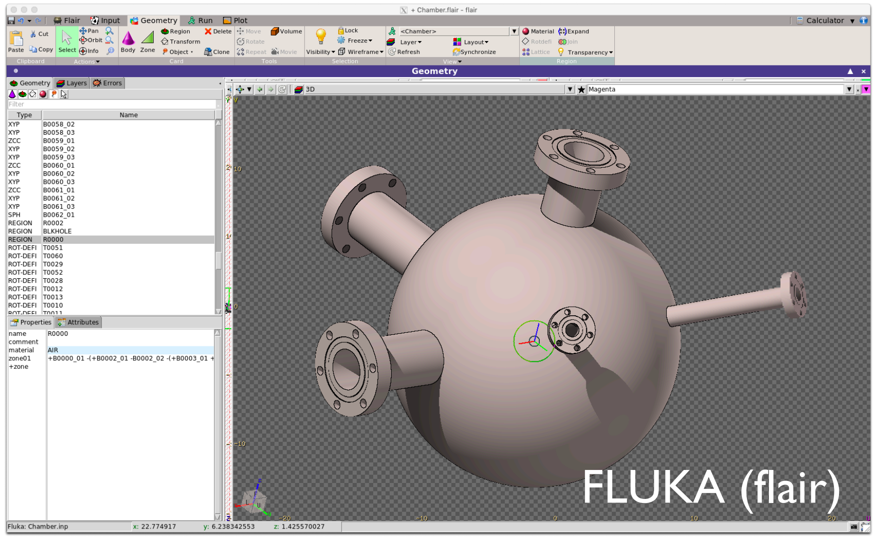


```python
import pyg4ometry.convert as convert
import pyg4ometry.visualisation as vi
from pyg4ometry.fluka import RPP, Region, Zone, FlukaRegistry, SPH


freg = FlukaRegistry()

rpp1 = RPP("RPP_BODY1", 0, 10, 0, 10, 0, 10, flukaregistry=freg)
sph = SPH("sph_body", [5, 5, 5], 4, flukaregistry=freg)

z = Zone()
z.addIntersection(rpp1)
z.addSubtraction(sph)
region = Region("RPP_REG")
region.addZone(z)
freg.addRegion(region)
freg.assignma("COPPER", region)

greg = convert.fluka2Geant4(freg)

v = vi.VtkViewer()
v.addAxes(length=20)
v.addLogicalVolume(greg.getWorldVolume())
v.view(interactive=interactive)
```

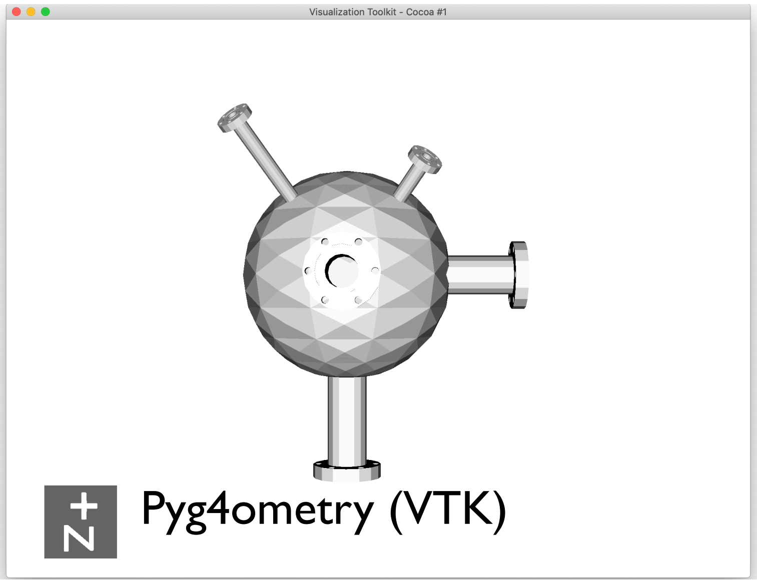


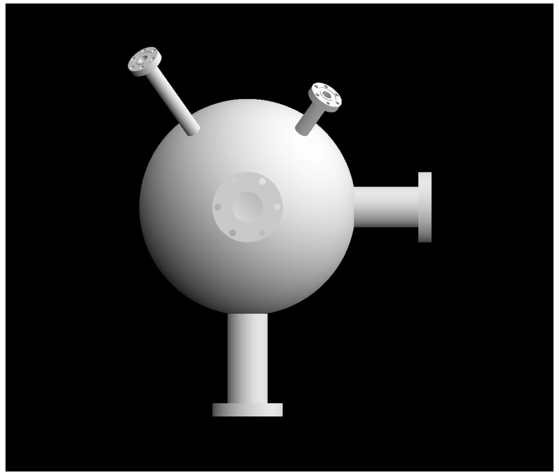
GEOMETRY CONVERSION

- Arbitrary transformations of in-memory geometry enabling geometry conversions between different formats:
  - FLUKA to Geant4
  - Geant4 Geant4 to FLUKA
  - STL and CAD to Geant4
- Difficult and error-prone to design a geometry in either FLUKA or Geant4.
- If you want to try a different code, you must start from scratch.
- Instead with Pyg4ometry simply convert between the two formats, saving many hours of time, without errors.



Vacuum chamber designed in FLUKA using flair before automatic conversion to Geant4.

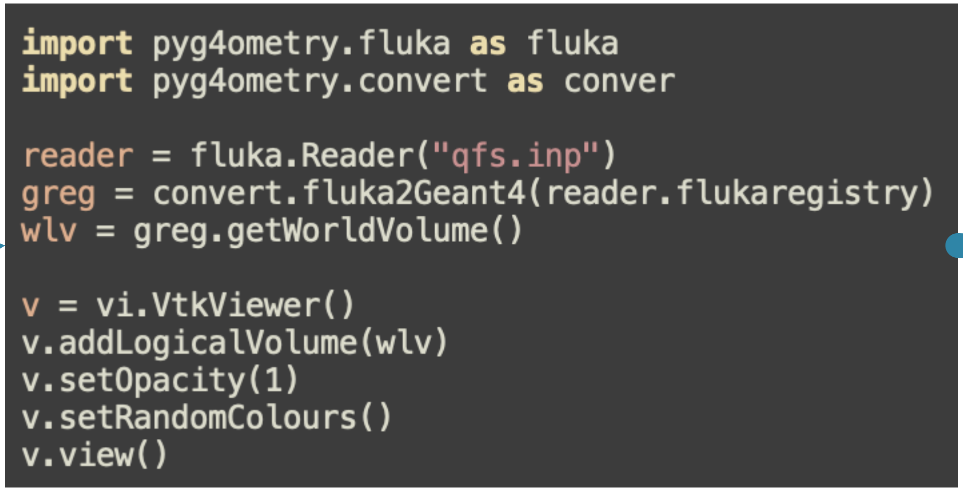




Geant4 ray tracer

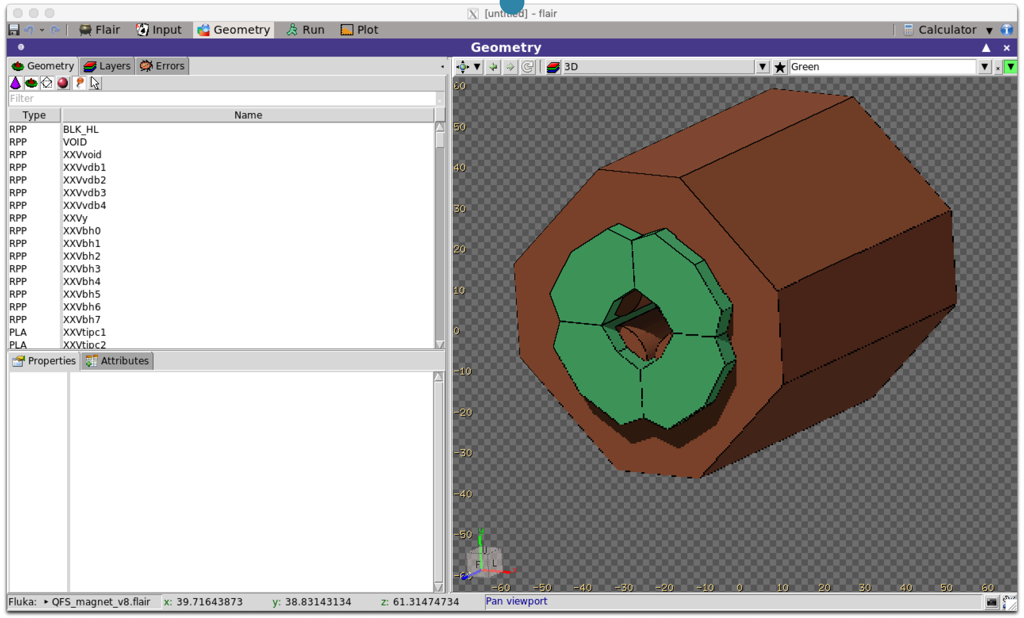
CONVERSION: FLUKA TO GDML

- KLEVER QFS Quadrupole designed in FLAIR and translated to Geant4.



```
import pyg4ometry.fluka as fluka
import pyg4ometry.convert as conver

reader = fluka.Reader("qfs.inp")
greg = convert.fluka2Geant4(reader.flukaregistry)
wlv = greg.getWorldVolume()

v = vi.VtkViewer()
v.addLogicalVolume(wlv)
v.setOpacity(1)
v.setRandomColours()
v.view()
```

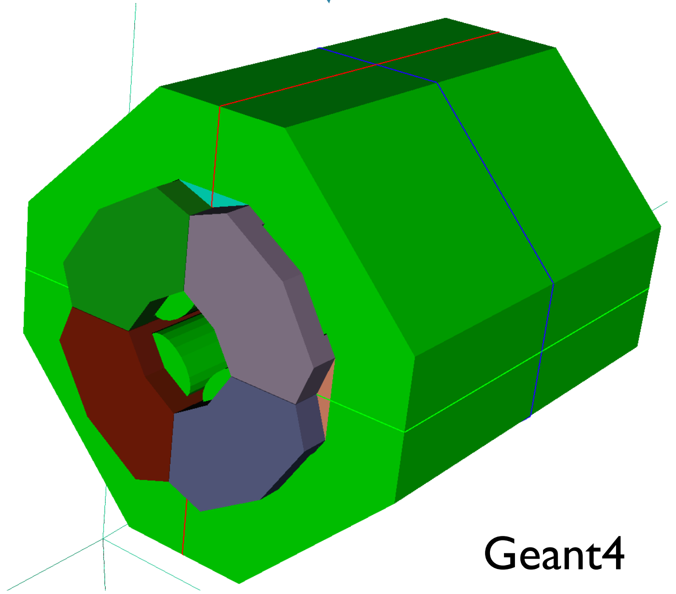
Simple Pyg4ometry script



Type	Name
RPP	BLK_HL
RPP	VOID
RPP	XXVvoid
RPP	XXVvdb1
RPP	XXVvdb2
RPP	XXVvdb3
RPP	XXVvdb4
RPP	XXVy
RPP	XXVbh0
RPP	XXVbh1
RPP	XXVbh2
RPP	XXVbh3
RPP	XXVbh4
RPP	XXVbh5
RPP	XXVbh6
RPP	XXVbh7
PLA	XXVtipc1
PLA	XXVtipc2

Fluka: ▸ QFS_magnet_v8.flair x: 39.71643873 y: 38.83143134 z: 61.31474734

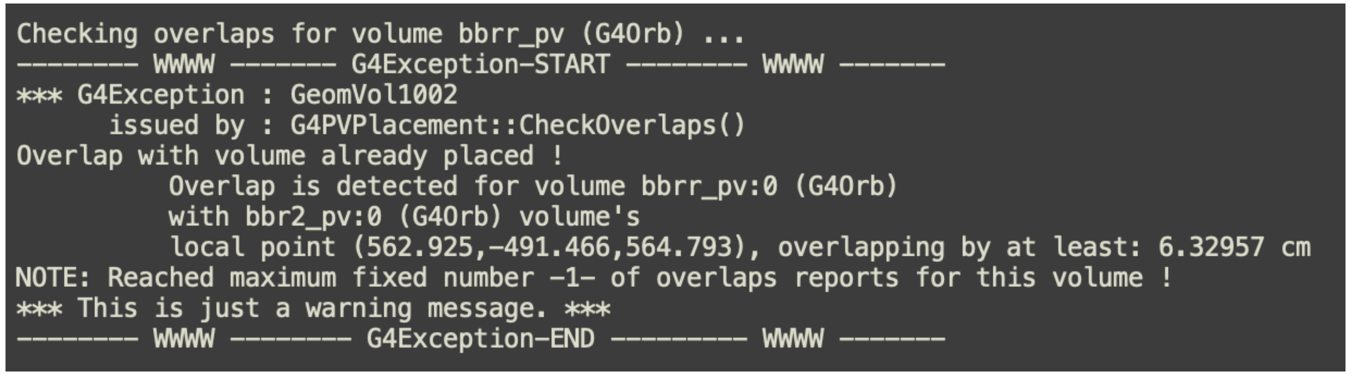
FLUKA (FLAIR)



Geant4

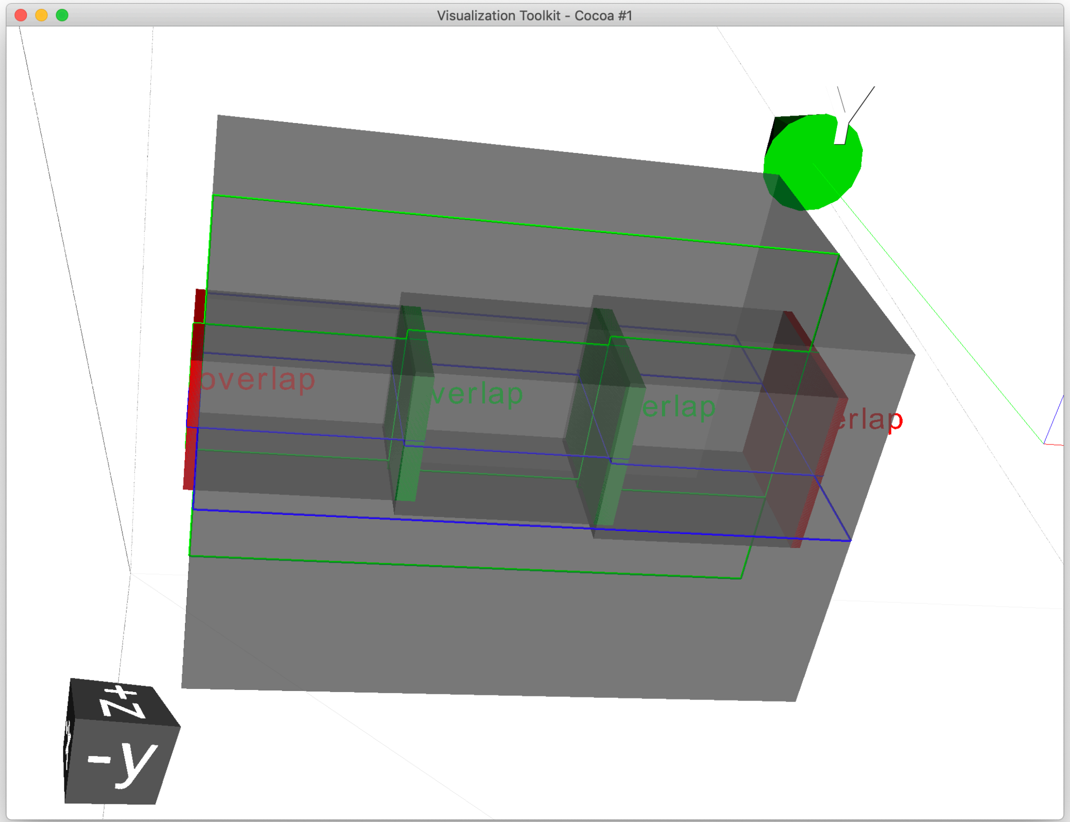
OVERLAP DETECTION

- Errors in the geometry preparation can result in incorrect results at run-time in any Monte Carlo code.
- Generally, these errors are "overlaps".
- Must be eliminated for accurate simulation
- Extremely user-friendly overlap visualization in Pyg4ometry offers substantial improvement over previous solutions.



```
Checking overlaps for volume bbrr_pv (G4Orb) ...
-------- WWWW ------- G4Exception-START -------- WWWW -------
*** G4Exception : GeomVol1002
      issued by : G4PVPlacement::CheckOverlaps()
Overlap with volume already placed !
          Overlap is detected for volume bbrr_pv:0 (G4Orb)
          with bbr2_pv:0 (G4Orb) volume's
          local point (562.925,-491.466,564.793), overlapping by at least: 6.32957 cm
NOTE: Reached maximum fixed number -1- of overlaps reports for this volume !
*** This is just a warning message. ***
-------- WWWW -------- G4Exception-END --------- WWWW -------
```

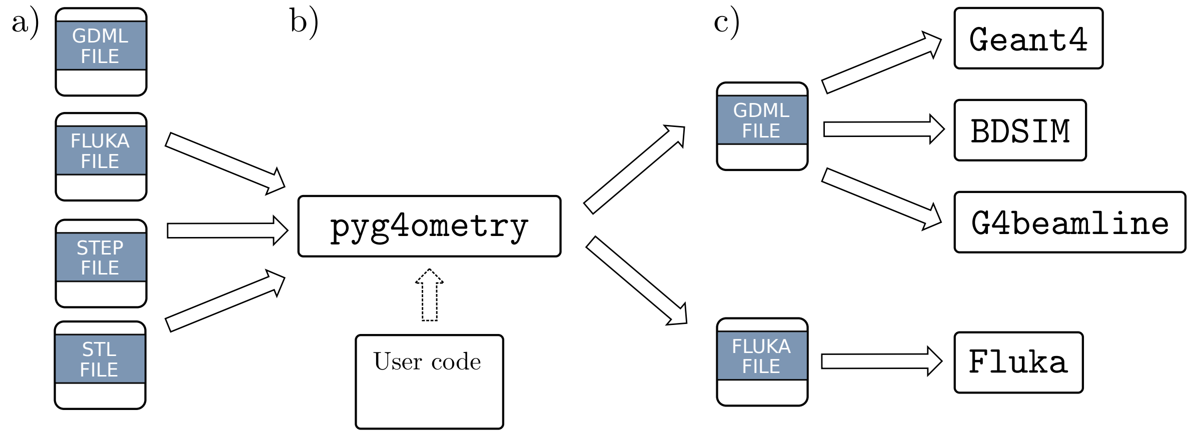
Geant4 overlap diagnostic

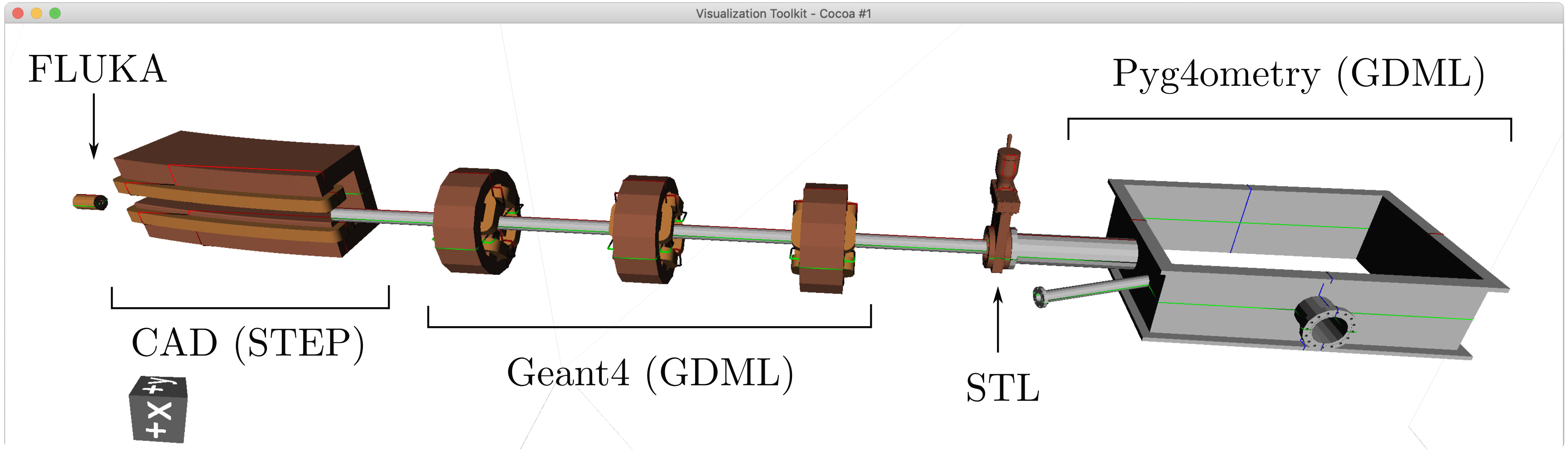


Pyg4ometry overlap diagnostic (shown in highlighted volumes)

COMPOSITION AND NOVEL WORKFLOWS

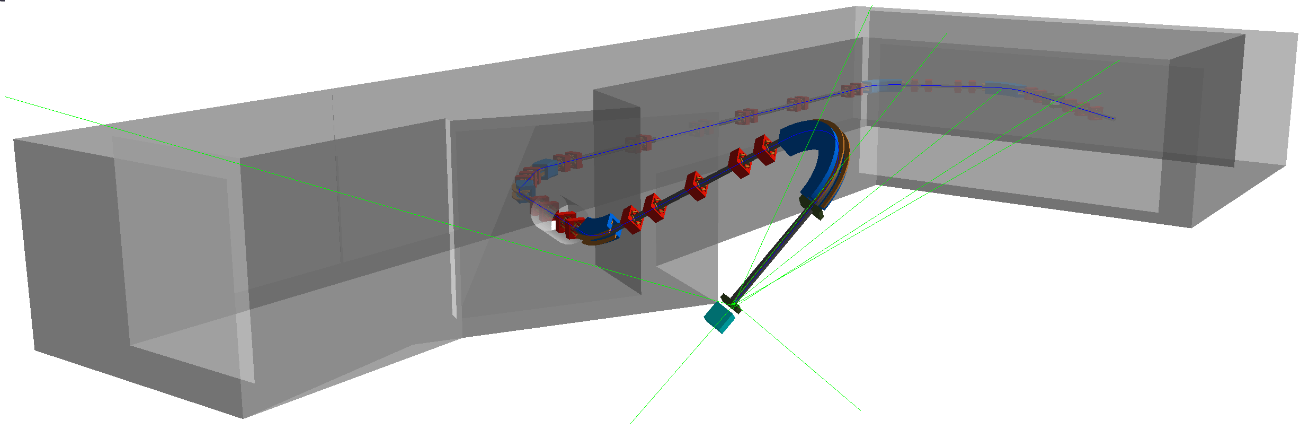
- Novel workflows enable composing a model from many different sources with ease.

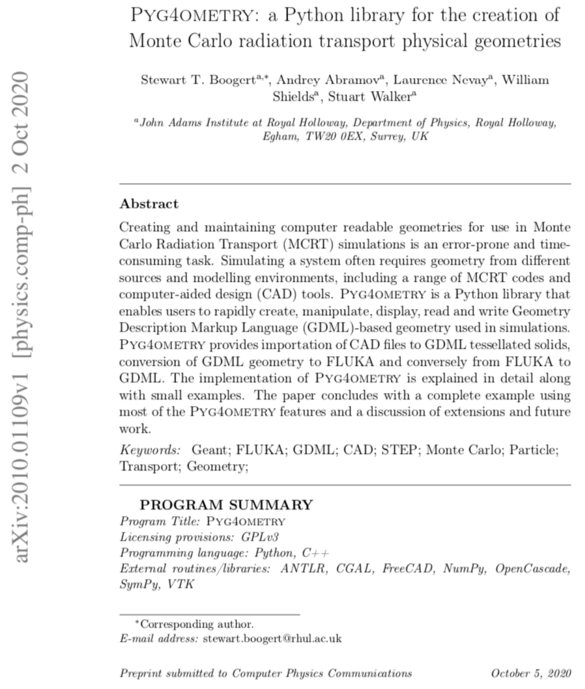




CONCLUSION

- All radiation transport codes need a sense of geometry, but developing that geometry can be very difficult.
- We have developed a general-purpose Python API for Monte Carlo Radiation transport geometry to massively ease this this process.
- Conversions to and from many formats.
- Proven usefulness to the physics community as now being used in simulating many experiments:
  - FASER, LUXE, ATLAS NCB, LHC collimation, medical beamlines, and others.
- Future is bright, great deal of possibilities due to Python being the standard language for hundreds of advanced libraries.





arXiv:2010.01109v1 [physics.comp-ph] 2 Oct 2020

PYG4OMETRY: a Python library for the creation of Monte Carlo radiation transport physical geometries

Stewart T. Boogert[a,*], Andrey Abramov[a], Laurence Nevay[a], William Shields[a], Stuart Walker[a]

[a]*John Adams Institute at Royal Holloway, Department of Physics, Royal Holloway, Egham, TW20 0EX, Surrey, UK*

Abstract

Creating and maintaining computer readable geometries for use in Monte Carlo Radiation Transport (MCRT) simulations is an error-prone and time-consuming task. Simulating a system often requires geometry from different sources and modelling environments, including a range of MCRT codes and computer-aided design (CAD) tools. PYG4OMETRY is a Python library that enables users to rapidly create, manipulate, display, read and write Geometry Description Markup Language (GDML)-based geometry used in simulations. PYG4OMETRY provides importation of CAD files to GDML tessellated solids, conversion of GDML geometry to FLUKA and conversely from FLUKA to GDML. The implementation of PYG4OMETRY is explained in detail along with small examples. The paper concludes with a complete example using most of the PYG4OMETRY features and a discussion of extensions and future work.

Keywords: Geant; FLUKA; GDML; CAD; STEP; Monte Carlo; Particle; Transport; Geometry;

PROGRAM SUMMARY

Program Title: PYG4OMETRY
Licensing provisions: GPLv3
Programming language: Python, C++
External routines/libraries: ANTLR, CGAL, FreeCAD, NumPy, OpenCascade, SymPy, VTK

[*]Corresponding author.
E-mail address: stewart.boogert@rhul.ac.uk

Preprint submitted to Computer Physics Communications *October 5, 2020*

arXiv:2010.01109