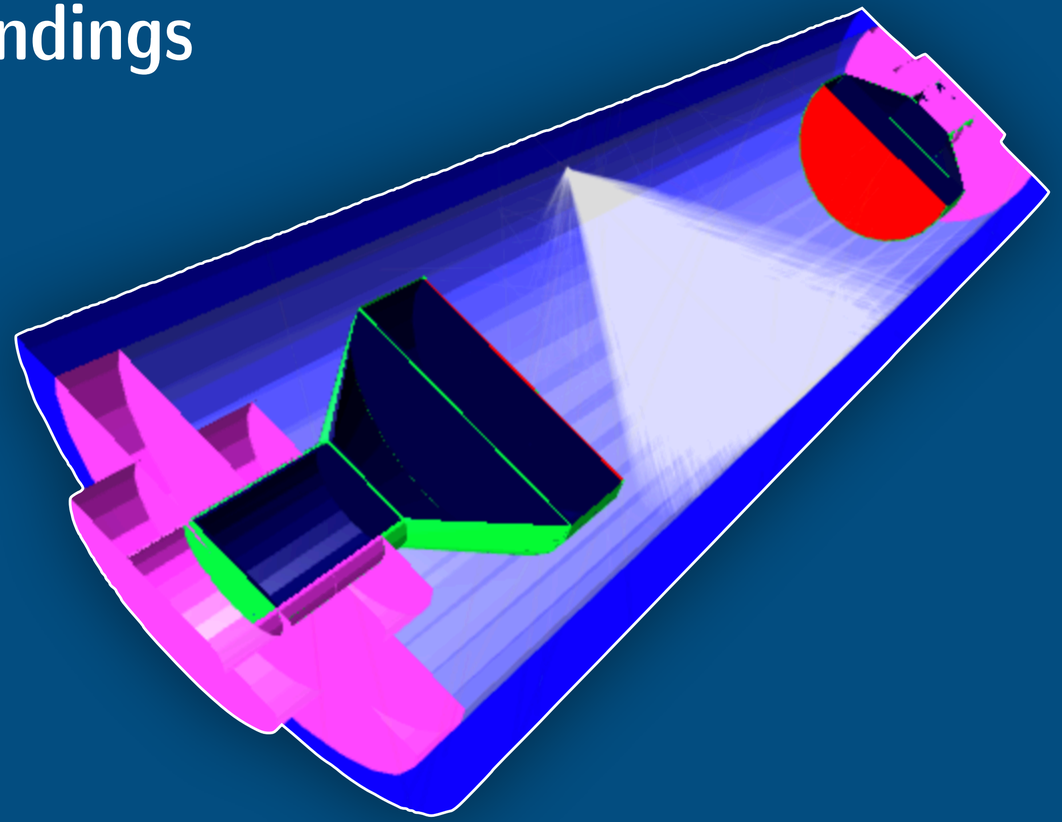
G4PPYY : Automated Geant4 Python Bindings

P. Stowell, R. Foster



University of Sheffield

Introduction

- Spent the past few years trying to run Geant4 based projects with undergraduate students.
- Many undergraduate courses moving to python focus, so clear need for better python dev tools.
- Trying to develop a fully containerised solution to avoid time lost installing software on different machines.
- Suggesting ‘g4ppyy’ as a solution to automated python bindings for geant4 which are easier to maintain.

Pybind11 Bindings vs CPPYY

- Python bindings already exist for Geant4 based on pybind11 (g4-pybind, g4python)

- Very useful, to teach the basics, but several limitations:
  - Pure virtual classes can require 'trampoline' intermediate classes
  - Bindings need to be manually written, so limited coverage of entire framework.
  - Mappings between C++/python not always exact, so building basic examples difficult.

- CPPYY is an alternative solution for **automated** binding of large frameworks.
  - Back end for the PyROOT framework built on cling (many HEP users already have this)
  - Available as a pypi package for those who don't want to use ROOT.
  - Fully automatic binding, based on shared libraries and headers.

Library Loading

- **Challenge** : Need to load all the relevant libraries + headers in to CPPYY.
- **Solution** : Build g4ppyy, a wrapper around cppy, that uses geant4-config to get libraries.

- Setup existing geant4 install

```
source geant4.sh
```

- Install CPPYY (specific version needed for matching C++ standard)

```
STDCXX=17 python3 -m pip install cppyy
```

- Install G4ppyy from GitHub

```
Pip install g4ppyy
```

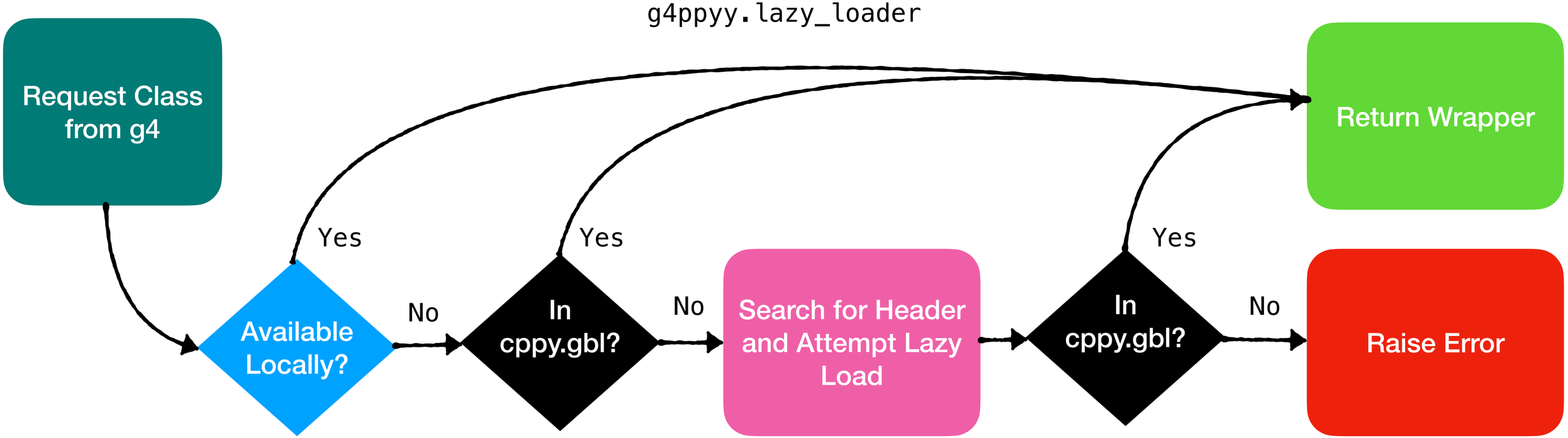
- Run!

```
python3 >> import g4ppyy as g4
```

```
[G4PPYY] : Geant4 Python wrapper for CPPYY
[G4PPYY] : Author: P. Stowell (p.stowell@sheffield.ac.uk)
[G4PPYY] :         R. Foster
[G4PPYY] : Loading G4 Modules.
[G4PPYY] : G4PREFIX : /app/geant4-v11.2.2/install
[G4PPYY] : G4VERSION : 11.2.2
```

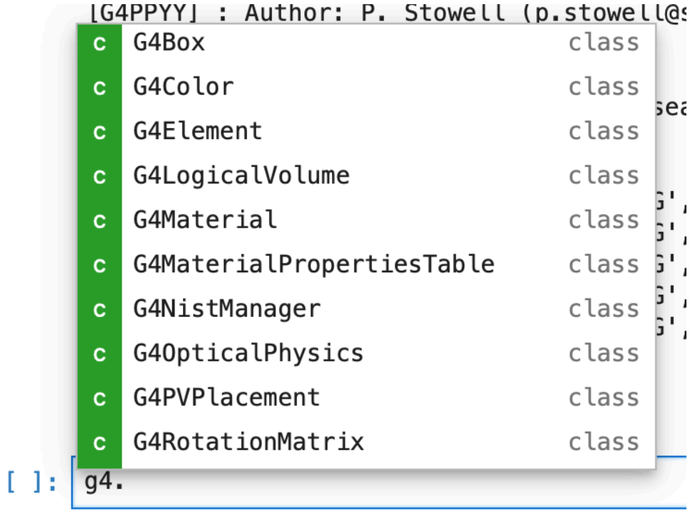
Lazy Classs Loader

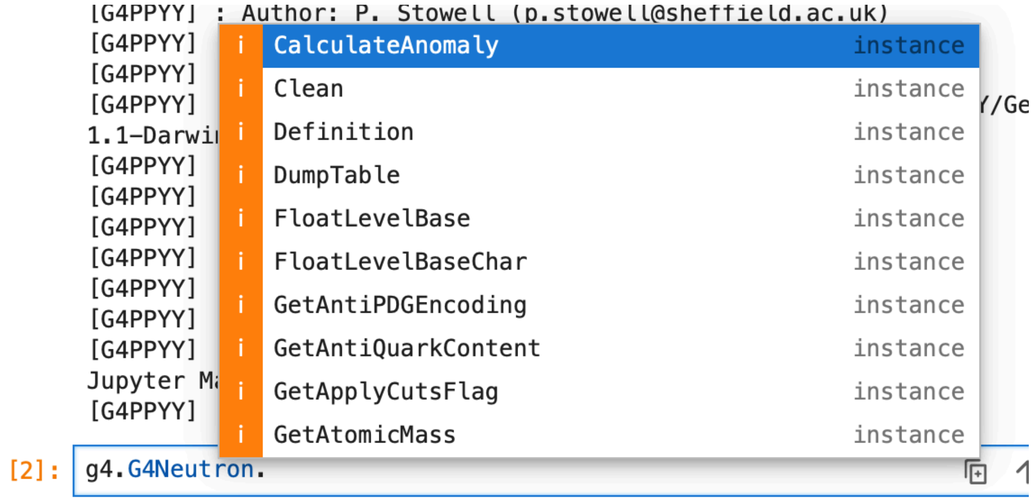
- **Challenge**: Loading the entire of the Geant4 definitions into CPPYY crashes the kernel
- **Solution**: Lazy Load Geant4 components on the fly using a module facade.

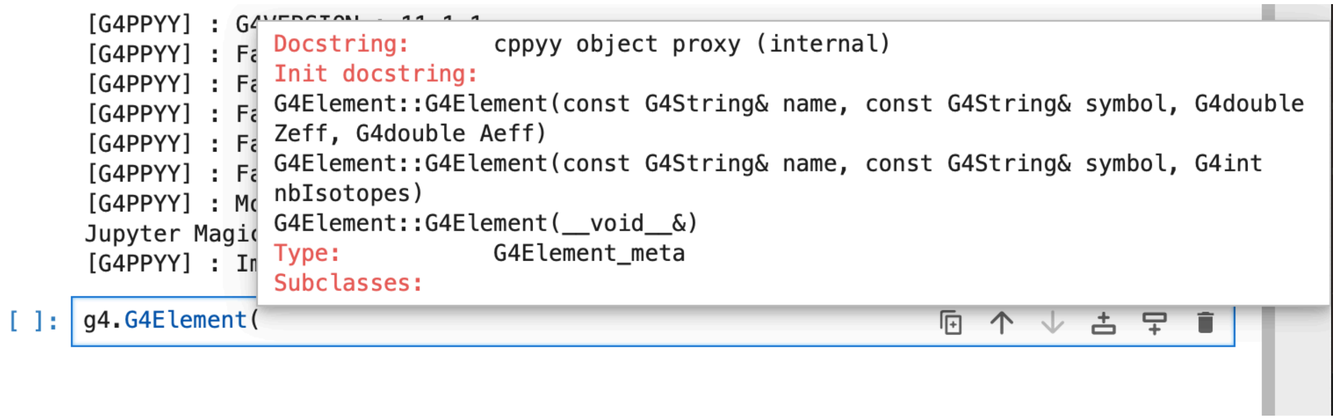


Jupyter Based Interface

- Once imported tab completion available for all currently loaded classes available in g4ppyy







Python Detector Construction

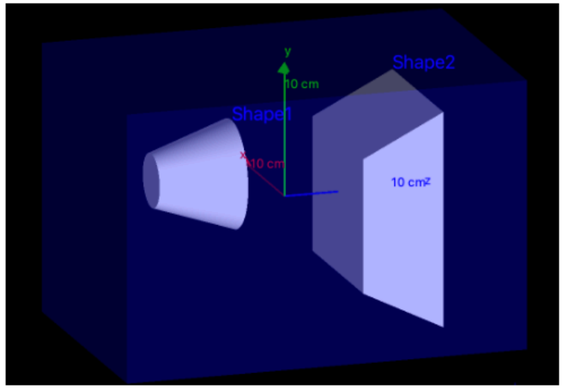
```python
import g4ppyy as g4

class CustomDetectorConstruction(g4.G4VUserDetectorConstruction):

    def Construct(self):
        nist = g4.gNistManager

        box_material = nist.FindOrBuildMaterial("G4_WATER")

        cm = g4.SI.cm # SI units wrapped in namespace
        solid_box = g4.G4Box("Box", 5*cm, 5*cm, 5*cm)

        logical_box = g4.G4LogicalVolume(solid_box, # Solid
            box_material,  #G4Material
            "Box")         #Name

        physical_box = g4.G4PVPlacement(0,       # no rotation
                            g4.G4ThreeVector(), # at (0,0,0)
                            logical_box,        # logical
                            "Box",              # name
                            0,                  # mother
                            False,              # boolean
                            0,                  # copy number
                            True)               # check overlap

        return physical_box
```

Similar thing possible in g4-pybind, we just didn't need to write the bindings!

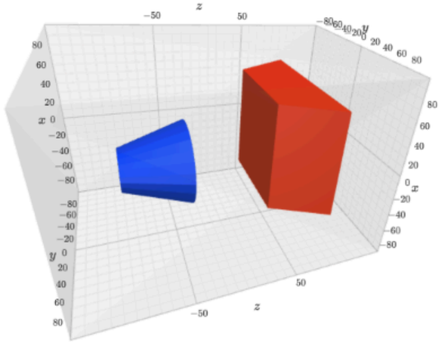
Benchmarking

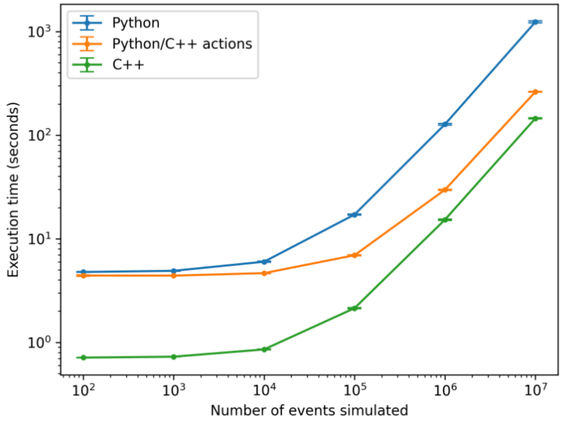
- Expect to see a performance drop running main event loop components in python.
- Initial benchmarking fully recreated the ExampleB1 in python.
- Find a factor of 7-8 slower in python. Majority of additional CPU processing lost in CustomEventAction.
- Use of C++ bindings in main CPU intensive components reduces this to factor of 2 loss.

C++ Geo Vis

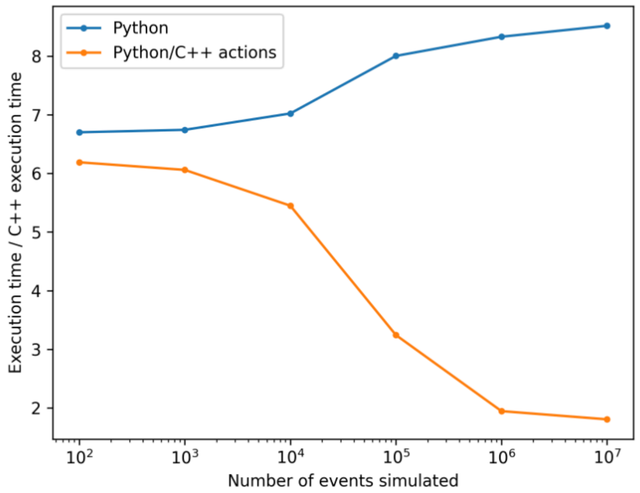


Python Geo Vis





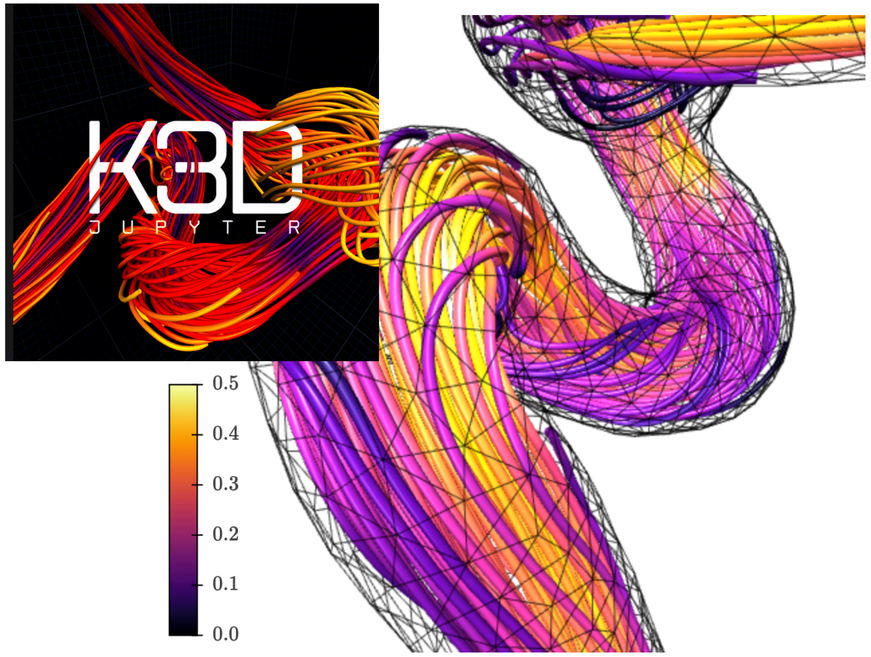
Benchmarking Results

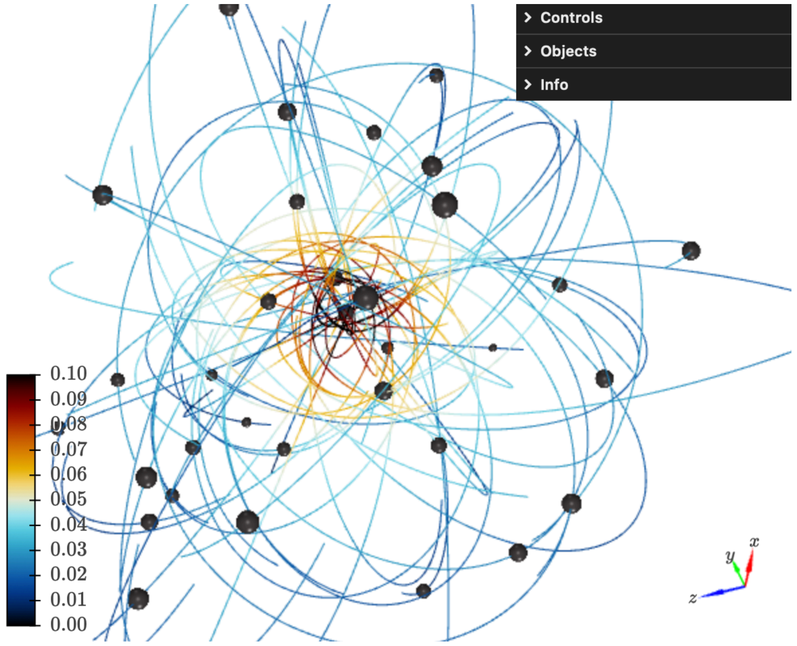


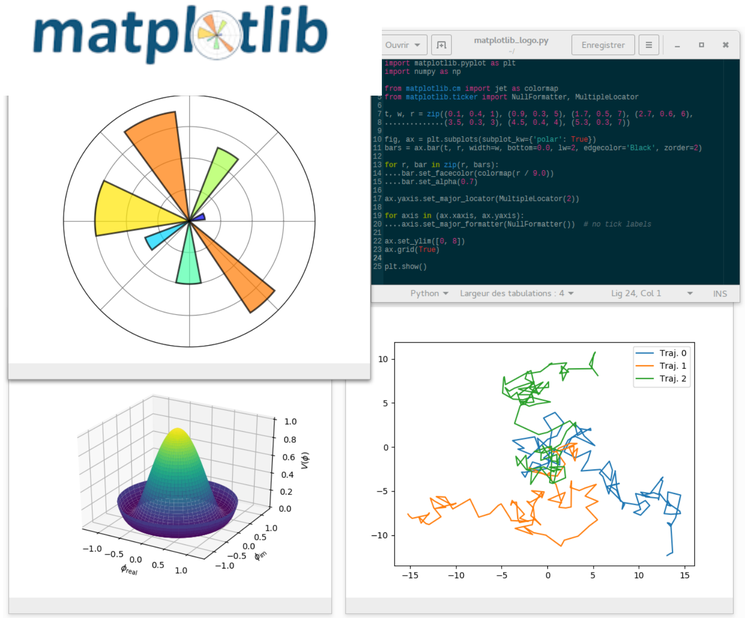
Python Visualisers

- **Challenge** : Setting up visualisation in Geant4 can be difficult. Either needs a local install, or need to set up proper port forwarding and handling of X11 on remote machines.

- **Solution** : Base python visualisers entirely on drawing tools with browser based support that are easy to extend.

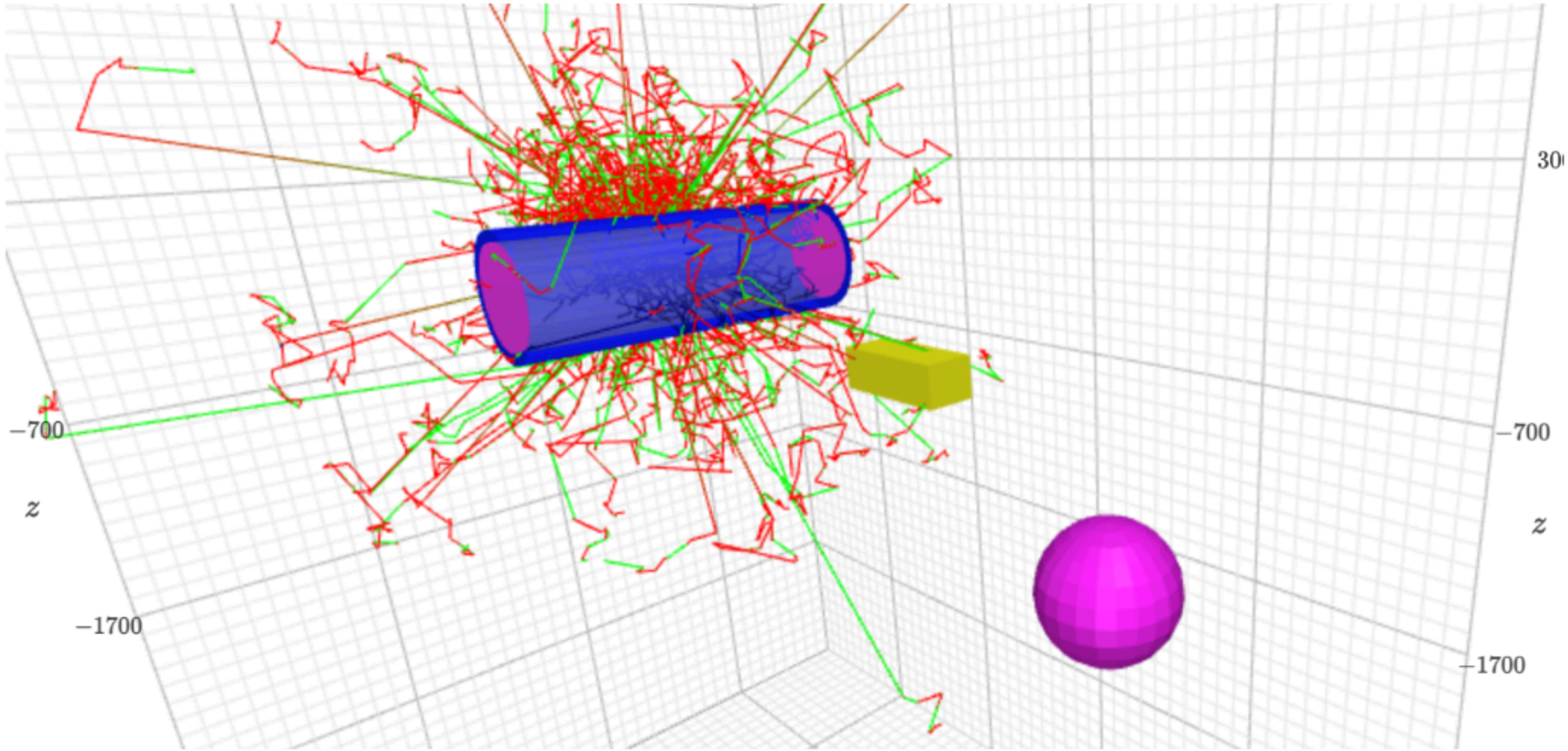






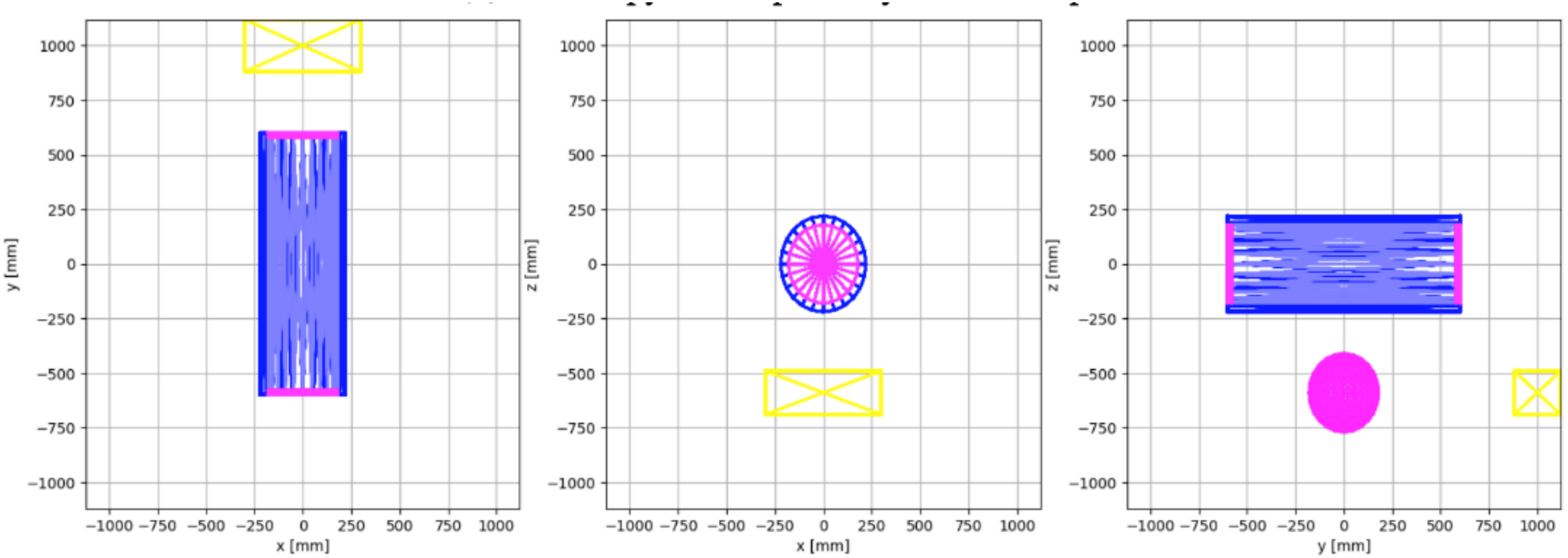
Python Visualiser : K3D

- K3D open access 3D interface available in Jupyter : jupyter-k3d



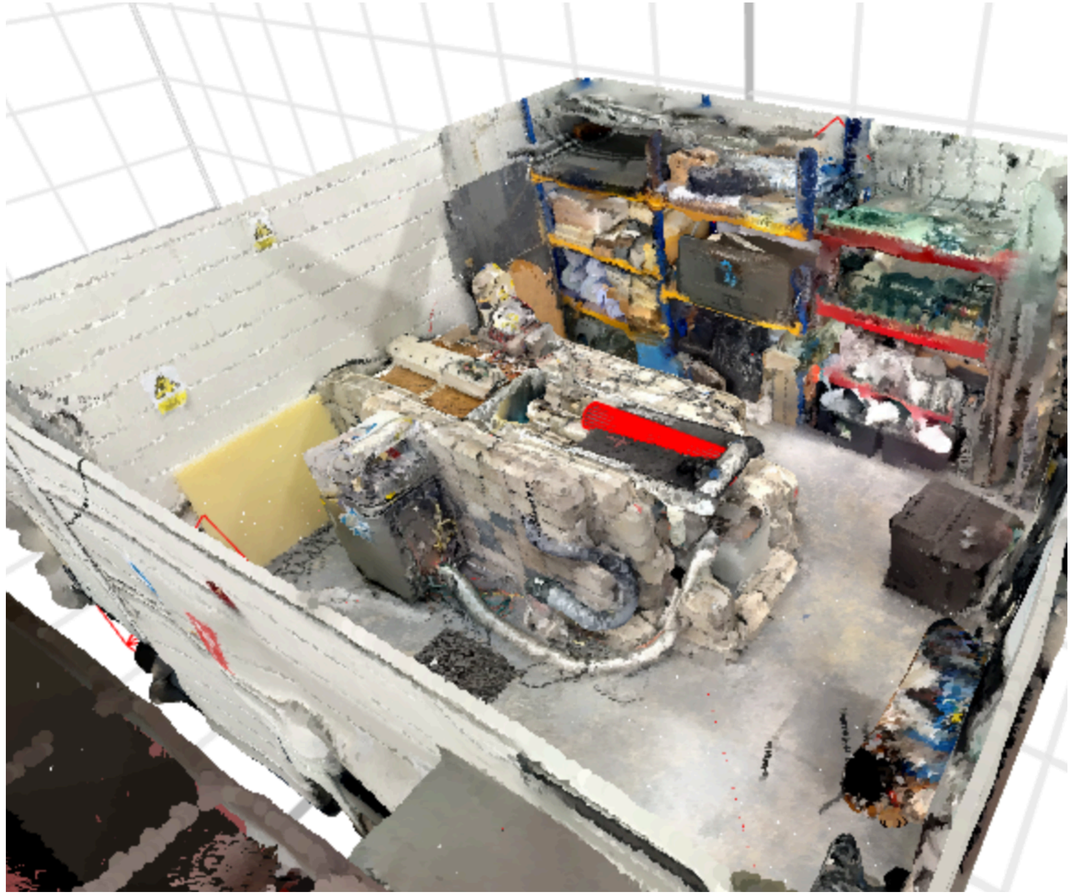
Python Visualiser : MatplotLib 2D

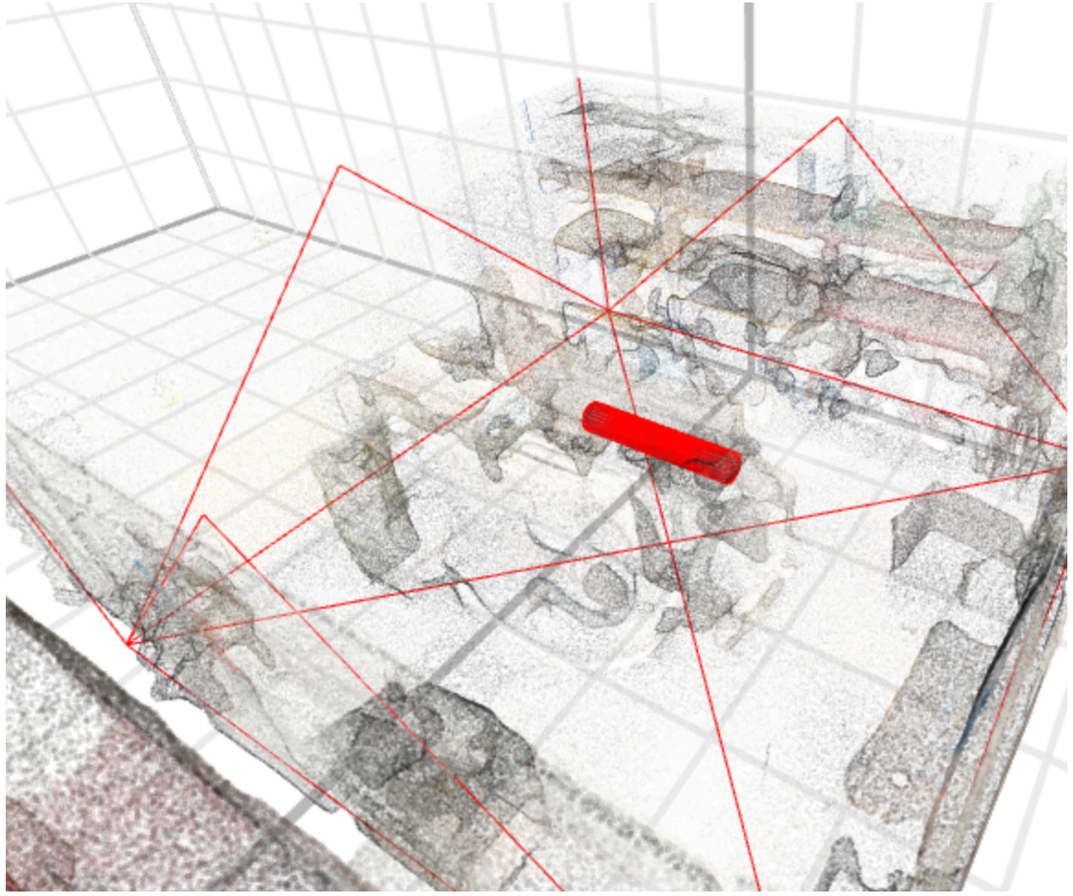
- Matplotlib is a common drawing tool in python many users will be familiar with.



Future Extensions

- Fast to iterate on drawing tools python side. Recent extension of k3d to include LIDAR point clouds to check alignment.





Python Helper Layer

- To make it easier for new users to get started with simple problems we are writing a python specific helper layer which has pythonized approaches to geometry assembly.

```
# Build a G4Box for the world made from loaded_water
world = g4.builder.build_component(
    name = "world",
    solid = "box",
    x=4*m, y=4*m, z=2*m,      # Box Dimensions
    material = loaded_water) # Box Material

# Build a G4Tubs POLYETHYLENE shell and place inside the world
hdpe_shell = g4.helper.build_component(
    name = "hdpe_shell",
    solid = "tubs",                # Tube Geometry
    rmax=11*cm, z=0.6*m/2,         # Tube Dimensions
    material = "G4_POLYETHYLENE", # Tube Material
    mother = world,                # Mother Placement
    rot = [90*deg, 0.0, 0.0],      # Rotation Angles
    color = [0.0,0.0,1.0,0.8],     # Drawing options
    drawstyle = "solid")           # Drawing options
```

Listing 6: Python helper functions to support rapid placement of logical volumes based on simple primitives.

Jupyter Magic

- For more advanced users, Jupyter Magic commands added to allow compilation of C++ code, or running of macros in a single Jupyter call.

[8]:
```
%%g4_compile
G4Box* build_box(){
    G4Box* my_box = new G4Box("cpp_box", 5*m, 5*m, 5*m);
    return my_box;
}
```

%%g4_compile [magic]
Compiles all of the cell as C++
Allows interlinked classes

[9]:
```
g4.build_box()
```

[9]:
```
<cppyy.gbl.G4Box object at 0x7fb4c699a8b0>
```

[10]:
```
%%g4_macro
/run/initalize
/run/beamOn 1000
```

%%g4_macro [magic]
Runs all of the cell as a G4 Macro

Jupyter Macro Interface

- Prototyping a python based macro interface that can be run directly in a cell alongside other python commands.

- g4.mc() used to build standard UI macro commands, with tab-completion listing available commands in a given path.

```
traj_mc = g4.mc.vis.modeling.trajectories
traj_mc.create.drawByCharge()
traj_mc.drawByParticleID_0.default.setDrawStepPts(True)
traj_mc.drawByParticleID_0.default.setStepPtsSize(1)
traj_mc.drawByParticleID_0.set("e+","white")
traj_mc.drawByParticleID_0.set("e-","white")
traj_mc.drawByParticleID_0.set("gamma","yellow")
traj_mc.drawByParticleID_0.set("neutron","magenta")
traj_mc.drawByParticleID_0.set("proton","blue")
traj_mc.drawByParticleID_0.set("pi+","red")
traj_mc.drawByParticleID_0.set("pi-","red")
traj_mc.drawByParticleID_0.set("pi0","grey")
```

Equivalent to
/vis/modelling/trajectories/create/drawByCharge

```
[6]: dir(mac)

Command directory path : /
 Sub-directories :
   /control/   UI control commands.
   /units/   Available units.
   /profiler/   Profiler controls.
   /material/   Commands for materials
 Commands :
Command directory path : /control/
```

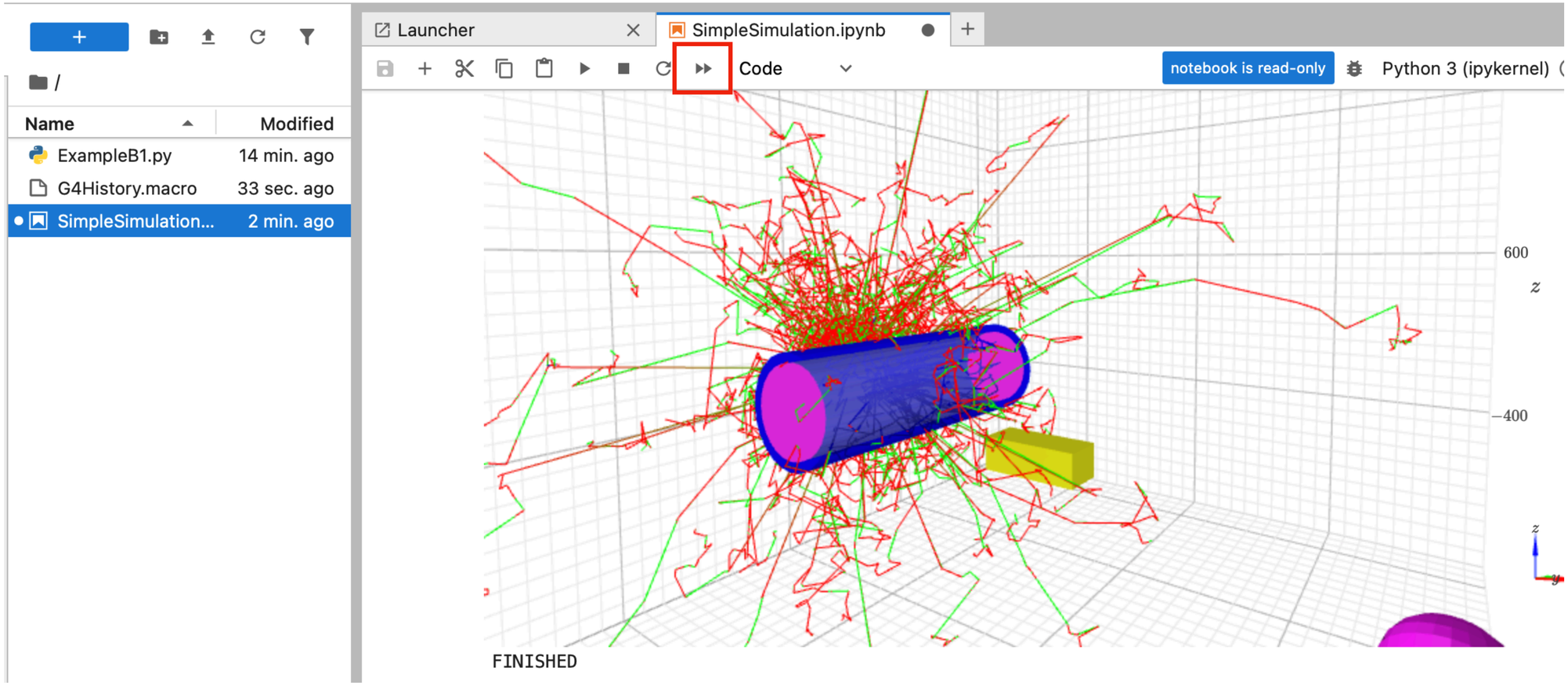
Containerised Solution

- All those components combine in to something that can be rapidly deployed.
  - docker run --rm -it --network=host johnpatrickstowell/g4ppyy:latest g4ppyy-jupyter

```
[I 2024-12-12 15:57:23.062 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-12-12 15:57:23.063 ServerApp] Serving notebooks from local directory: /data
[I 2024-12-12 15:57:23.063 ServerApp] Jupyter Server 2.14.2 is running at:
[I 2024-12-12 15:57:23.063 ServerApp] http://orbstack:8168/lab?token=5788bdc826156f385d5898e6aab8e52a7ddc4f3919dfdea6
[I 2024-12-12 15:57:23.063 ServerApp]     http://127.0.0.1:8168/lab?token=5788bdc826156f385d5898e6aab8e52a7ddc4f3919dfdea6
[I 2024-12-12 15:57:23.063 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 2024-12-12 15:57:23.068 ServerApp] No web browser found: Error('could not locate runnable browser').
[C 2024-12-12 15:57:23.069 ServerApp]

    To access the server, open this file in a browser:
        file:///home/g4user1/.local/share/jupyter/runtime/jpserver-7-open.html
    Or copy and paste one of these URLs:
        http://127.0.0.1:8168/lab?token=5788bdc826156f385d5898e6aab8e52a7ddc4f3919dfdea6
[I 2024-12-12 15:57:23.397 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-nodejs,
```

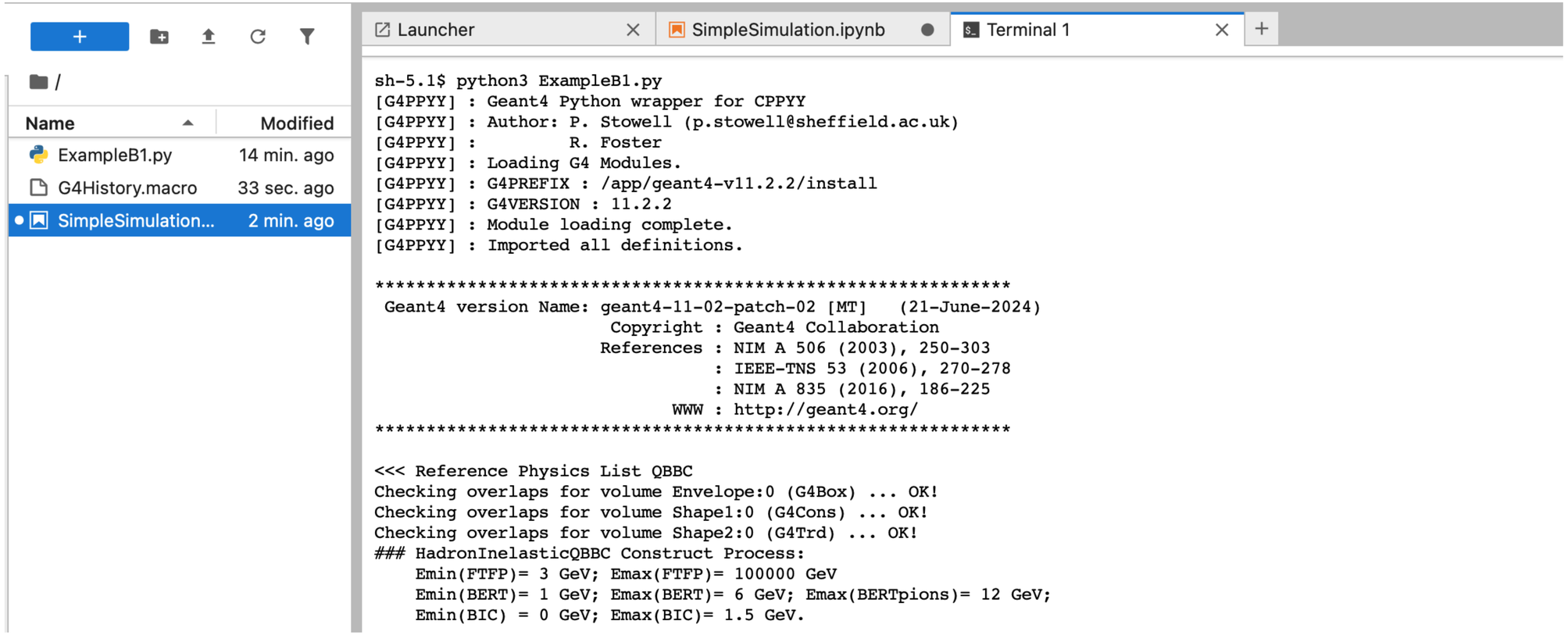
Containerised Solution

- All those components combine in to something that can be rapidly deployed.
  - docker run --rm -it --network=host johnpatrickstowell/g4ppyy:latest g4ppyy-jupyter



Containerised Solution

- All those components combine in to something that can be rapidly deployed.
  - docker run --rm -it --network=host johnpatrickstowell/g4ppyy:latest g4ppyy-jupyter



Conclusions & Future Work

- Demonstrated the potential for using CPPYY to build python bindings for Geant4.
- Overcome most of the major issues that would limit containerised tutorials within python.
- Two simplistic drawing tools developed for Jupyter based development, that are compatible with standard visualisation macros.
- Suggesting this as a potential way forward for well maintained python bindings in the future inside Geant4. Pre-release has been tagged as v0.1.0 here:
  - https://github.com/patrickstowell/G4ppyy
  - Brief summary here : https://arxiv.org/abs/2412.05593
- Comments, pull requests, issues, collaborators all very welcome!