GGD-based Vertical Drift Geometry

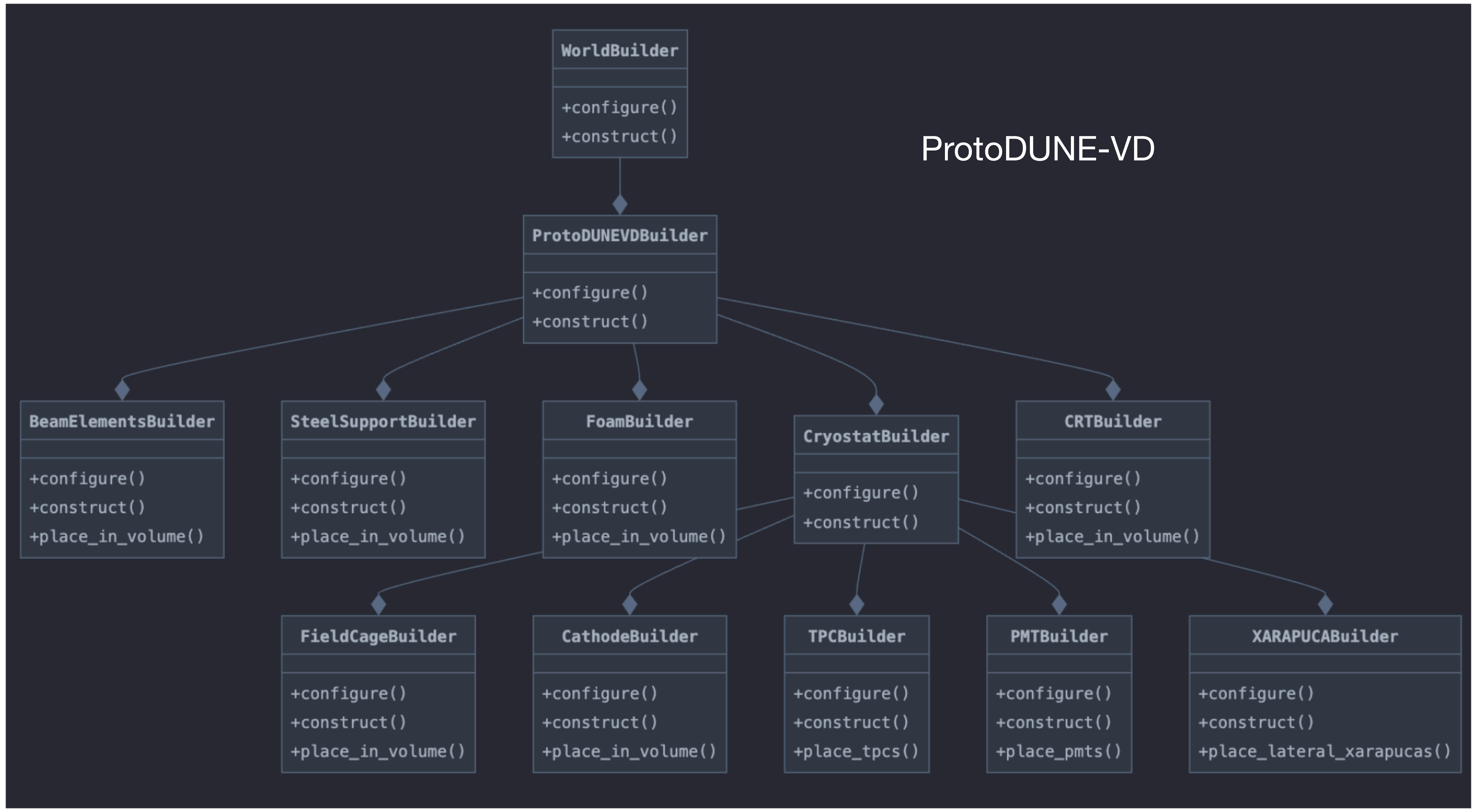
Wenqiang Gu, Nitish Nayak, Xin Qian

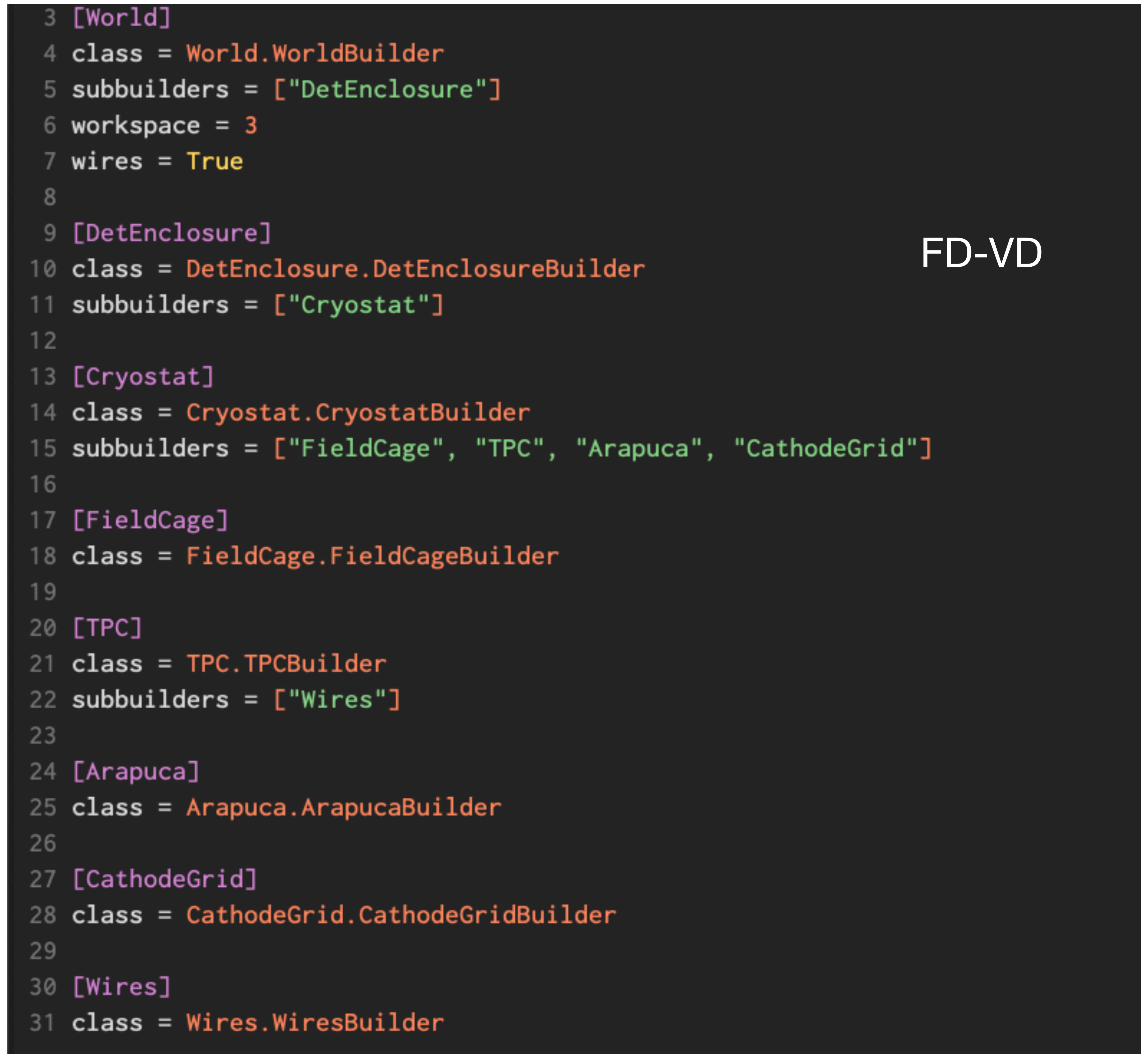
6th Feb, 2025

Introduction

- Motivated by discussion related to low energy studies in VD (Eric Church et al.)
- Geometry construction handled by single Perl scripts
  - Various options allowed but not very configurable and extensible to new requirements
  - Requires detailed knowledge of various elements, in consultation with Geometry experts
  - => Longer development cycles, more chance for bugs, more technical debt etc..

GeGeDe



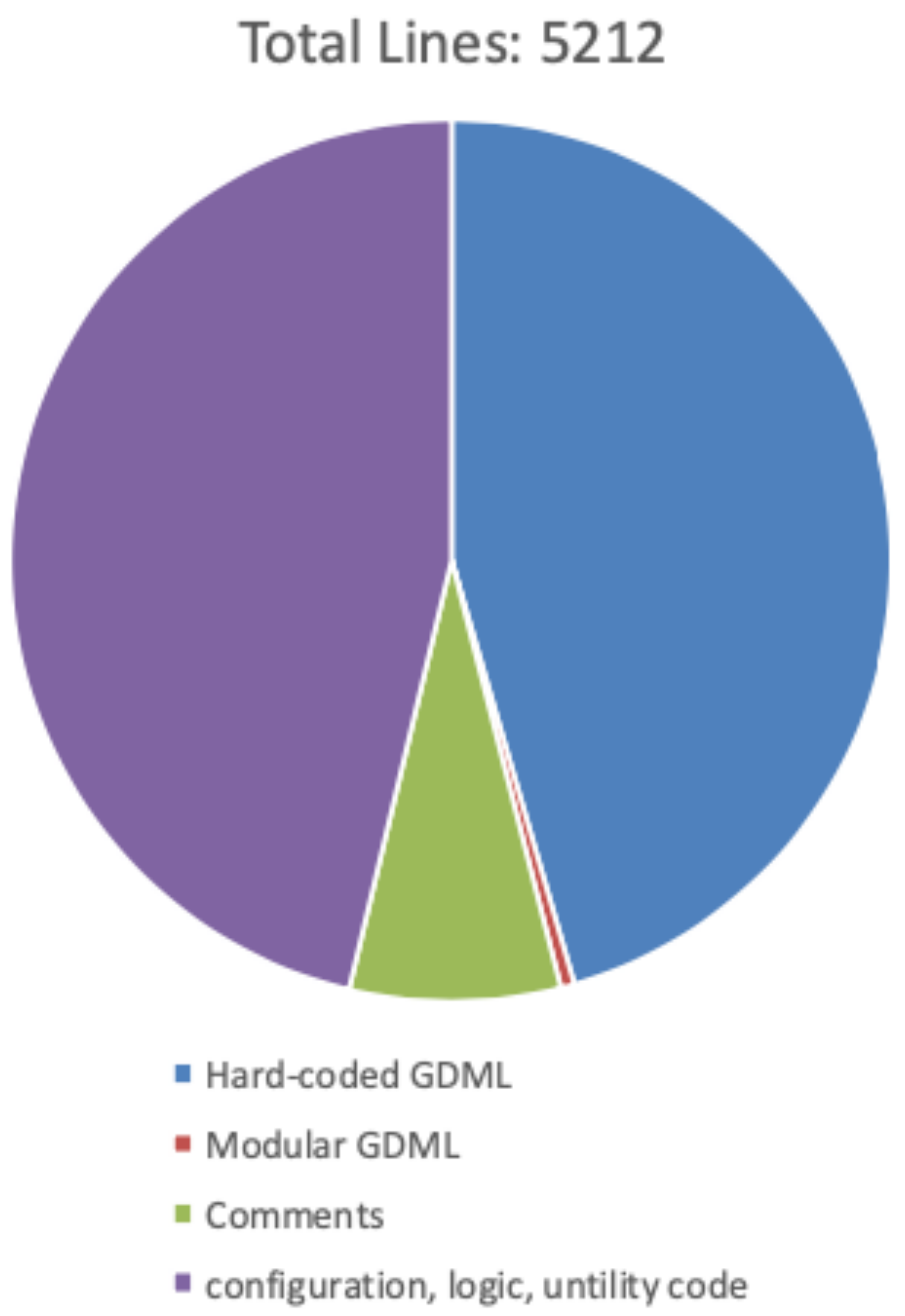


```
[World]
class = World.WorldBuilder
subbuilders = ["DetEnclosure"]
workspace = 3
wires = True

[DetEnclosure]
class = DetEnclosure.DetEnclosureBuilder
subbuilders = ["Cryostat"]

[Cryostat]
class = Cryostat.CryostatBuilder
subbuilders = ["FieldCage", "TPC", "Arapuca", "CathodeGrid"]

[FieldCage]
class = FieldCage.FieldCageBuilder

[TPC]
class = TPC.TPCBuilder
subbuilders = ["Wires"]

[Arapuca]
class = Arapuca.ArapucaBuilder

[CathodeGrid]
class = CathodeGrid.CathodeGridBuilder

[Wires]
class = Wires.WiresBuilder
```

- Python-based geometry constructor designed by Brett Viren : https://github.com/brettviren/gegede
- Can emit GDML to interface with Geant4/ROOT, enforces consistent units through Pint
- Easily customizable schema for describing various shapes, materials etc
- Easy human-driven configuration, classes for layer-by-layer building

Perl Scripts

- Existing implementation for Horizontal Drift (P. Lasorak, A. Borkum) : https://github.com/DUNE/duneggd/tree/master/python/duneggd
  - Seems like not widely used but its there
- ProtoDUNE-VD geometry uses v4 Perl script
  - ~5000 lines, challenging to maintain
- Similarly for FD-VD : uses v6 script
  - ~2000 lines
- We want to port this over to GGD and maintain it
- Validation to ensure all details are captured, including testing by running GEANT over GGD-based gdml



Human-AI workflow

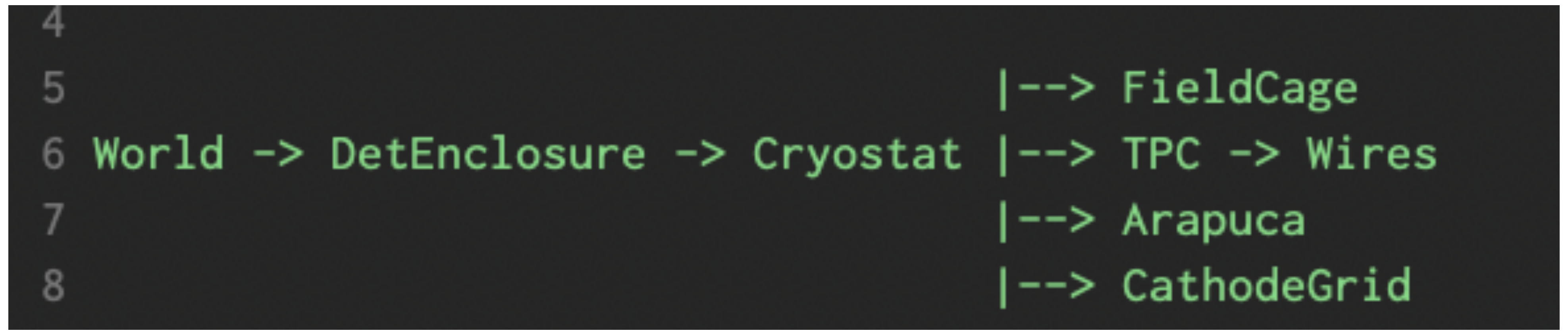
- More details in Xin's talk at DUNE CM

- Xin has had lot of success using Github Co-pilot + Claude 3.5 Sonnet
  - Able to construct detailed ProtoDUNE-VD geometry within ~1 week
    - Translating messy PERL code for subcomponents
    - Using GeGeDe package as context input for LLM
    - Validation using existing GDML vs produced GDML and visualization tools (from Chao)
- For FD-VD, I didn't use LLMs (because I like going through messy code) but it was useful to see ProtoDUNE-VD code to learn GGD

- Usage Cases:
  - Understand the original PERL script
  - Understand the GeGeDe Conventions
  - Write > 90% of the code for ProtoDUNE-VD
  - Establish the validation methods

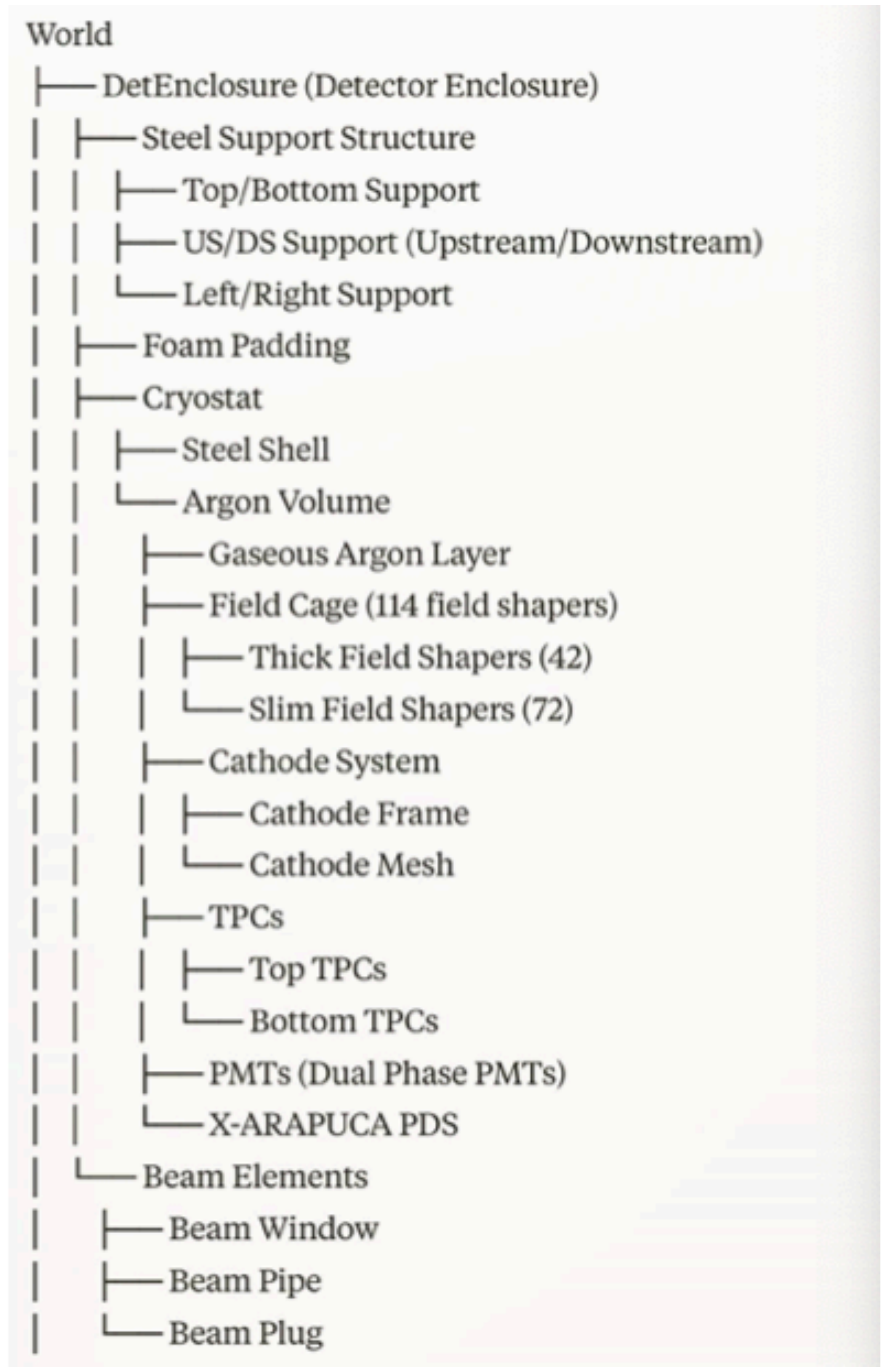
Implementation

FD-VD (Mine)



```

                                |--> FieldCage
World -> DetEnclosure -> Cryostat |--> TPC -> Wires
                                |--> Arapuca
                                |--> CathodeGrid
```

ProtoDUNE-VD (Xin’s)

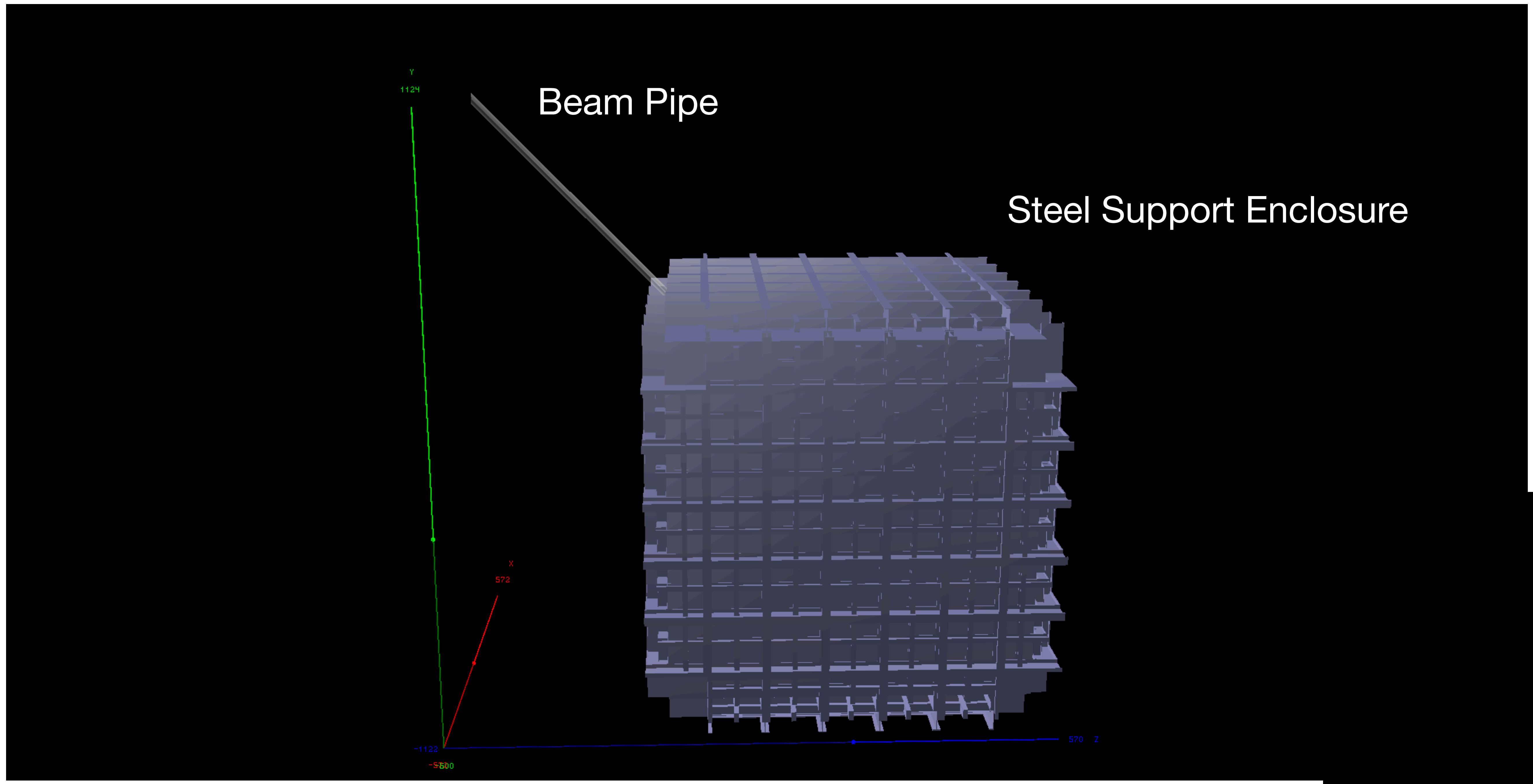


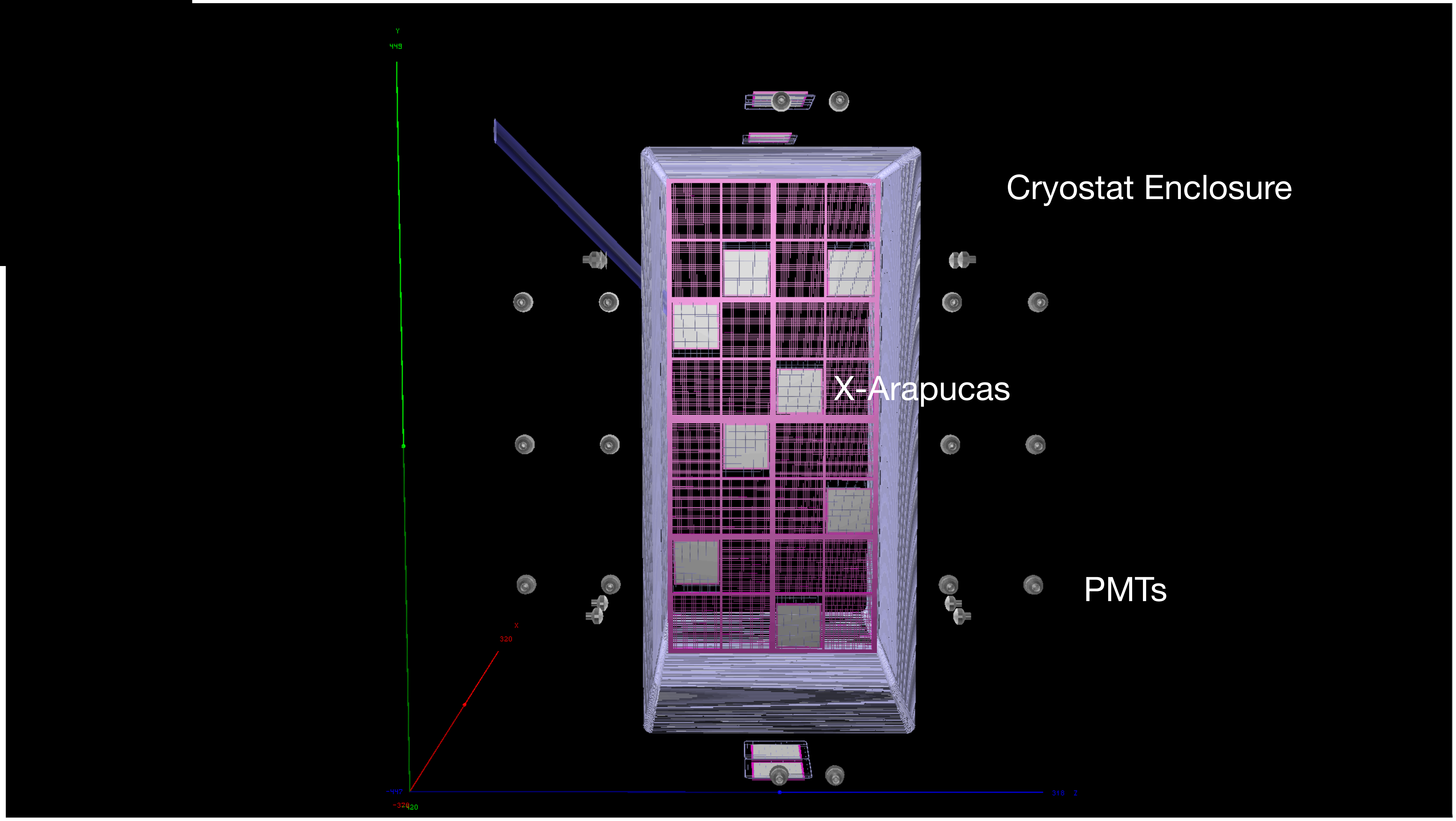
```
World
├── DetEnclosure (Detector Enclosure)
│   ├── Steel Support Structure
│   │   ├── Top/Bottom Support
│   │   ├── US/DS Support (Upstream/Downstream)
│   │   └── Left/Right Support
│   ├── Foam Padding
│   ├── Cryostat
│   │   ├── Steel Shell
│   │   └── Argon Volume
│   │       ├── Gaseous Argon Layer
│   │       ├── Field Cage (114 field shapers)
│   │       │   ├── Thick Field Shapers (42)
│   │       │   └── Slim Field Shapers (72)
│   │       ├── Cathode System
│   │       │   ├── Cathode Frame
│   │       │   └── Cathode Mesh
│   │       ├── TPCs
│   │       │   ├── Top TPCs
│   │       │   └── Bottom TPCs
│   │       ├── PMTs (Dual Phase PMTs)
│   │       └── X-ARAPUCA PDS
│   └── Beam Elements
│       ├── Beam Window
│       ├── Beam Pipe
│       └── Beam Plug
```

- Xin and I have implemented GGD-based builders for both FD and ProtoDUNE
  - Done in parallel, different approaches, code structure etc (for self-learning and validation purposes)
  - Mine : https://github.com/nitish-nayak/ggd_dunefdvd
  - Xin’s : https://github.com/lastgeorge/gegede_protodune_vd
  - PR for existing duneggd repository : https://github.com/DUNE/duneggd/pull/13
    - Combines both codes

Validation

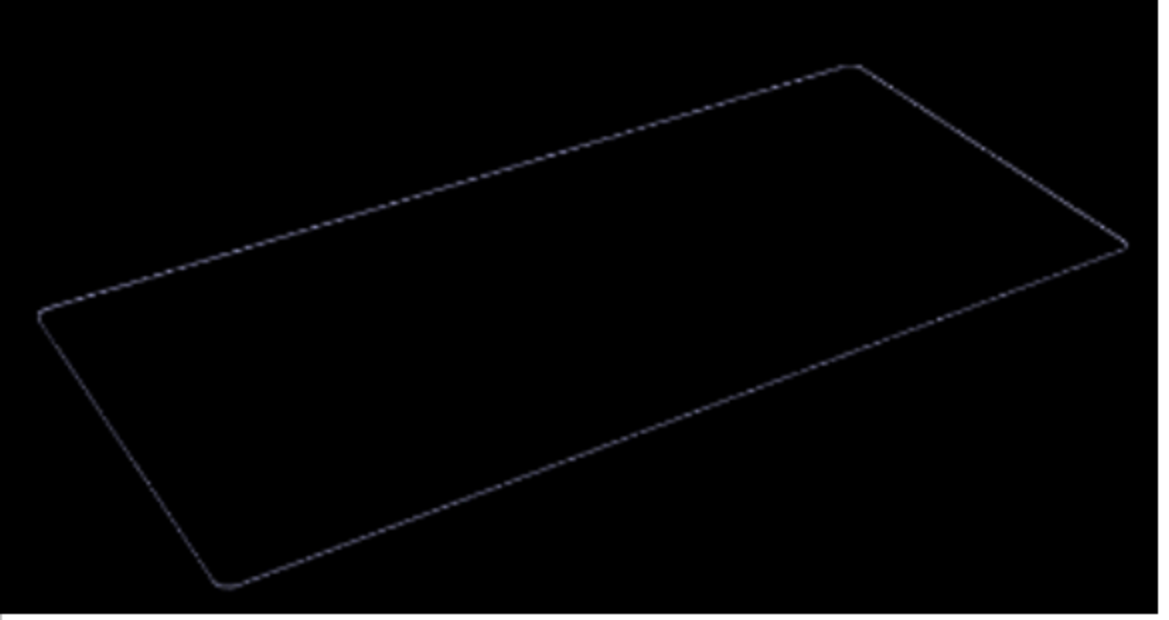
ProtoDUNE-VD

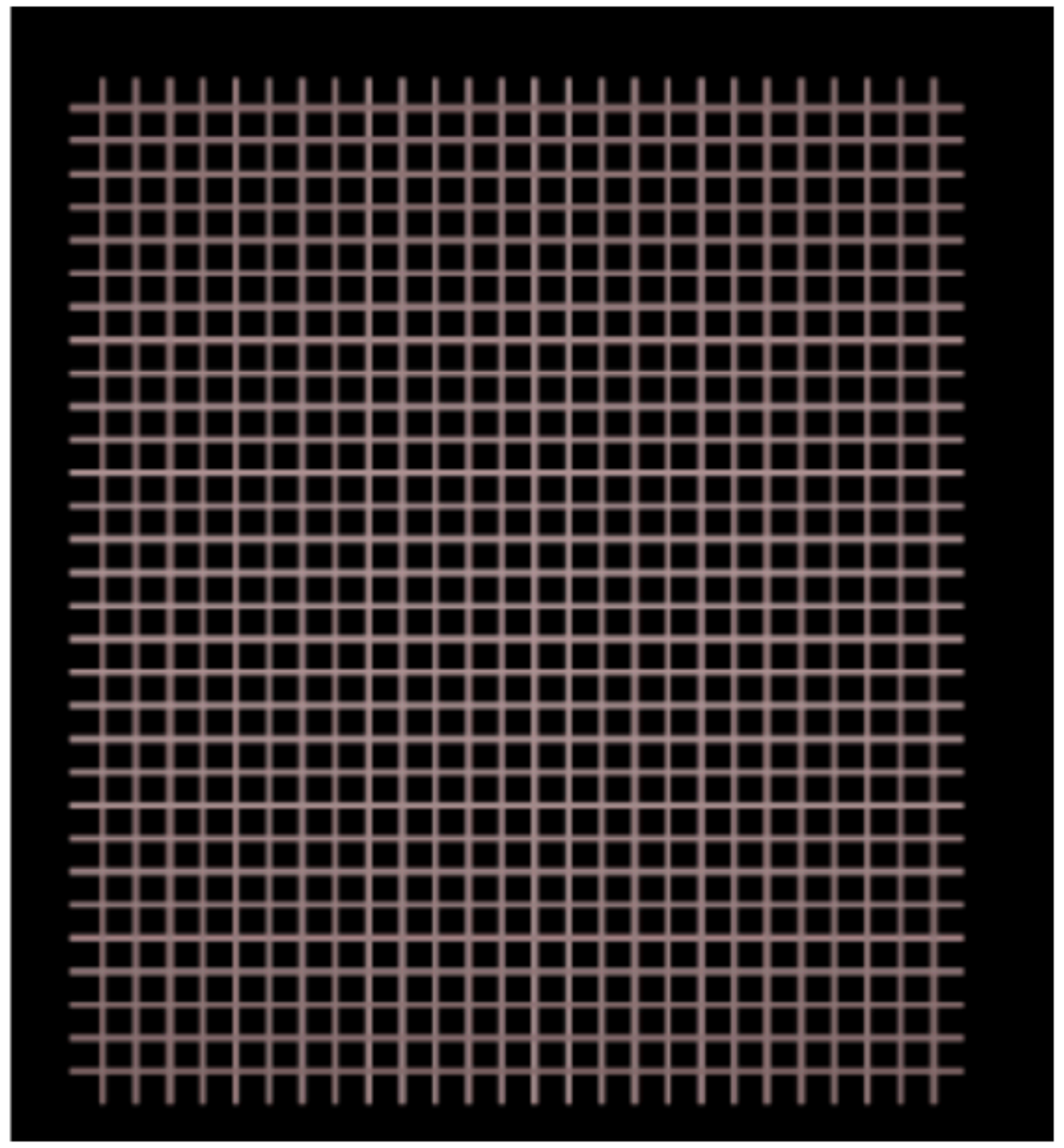




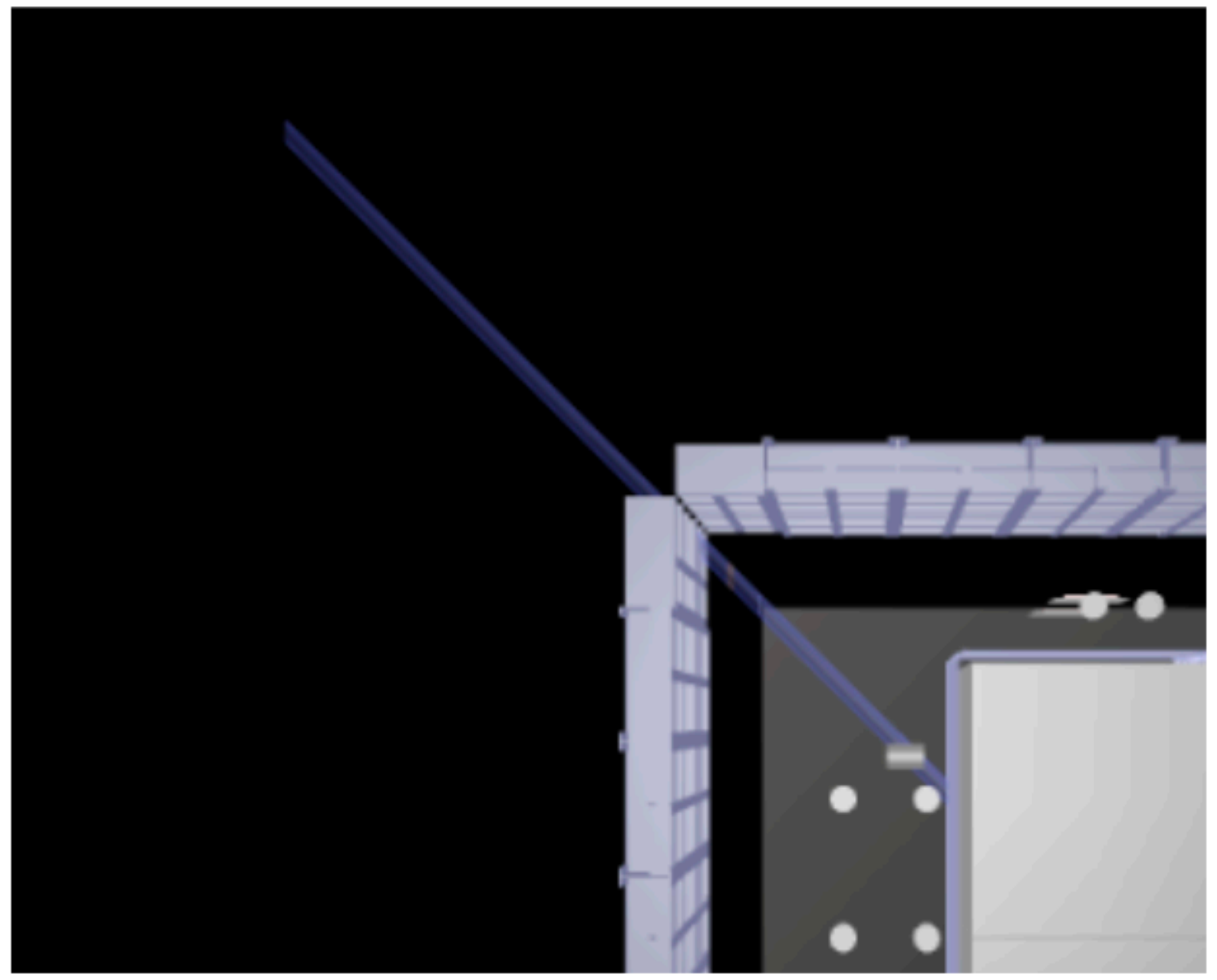
- All sub-components have been implemented
- Xin has validated that his implementation is functionally consistent with perl script

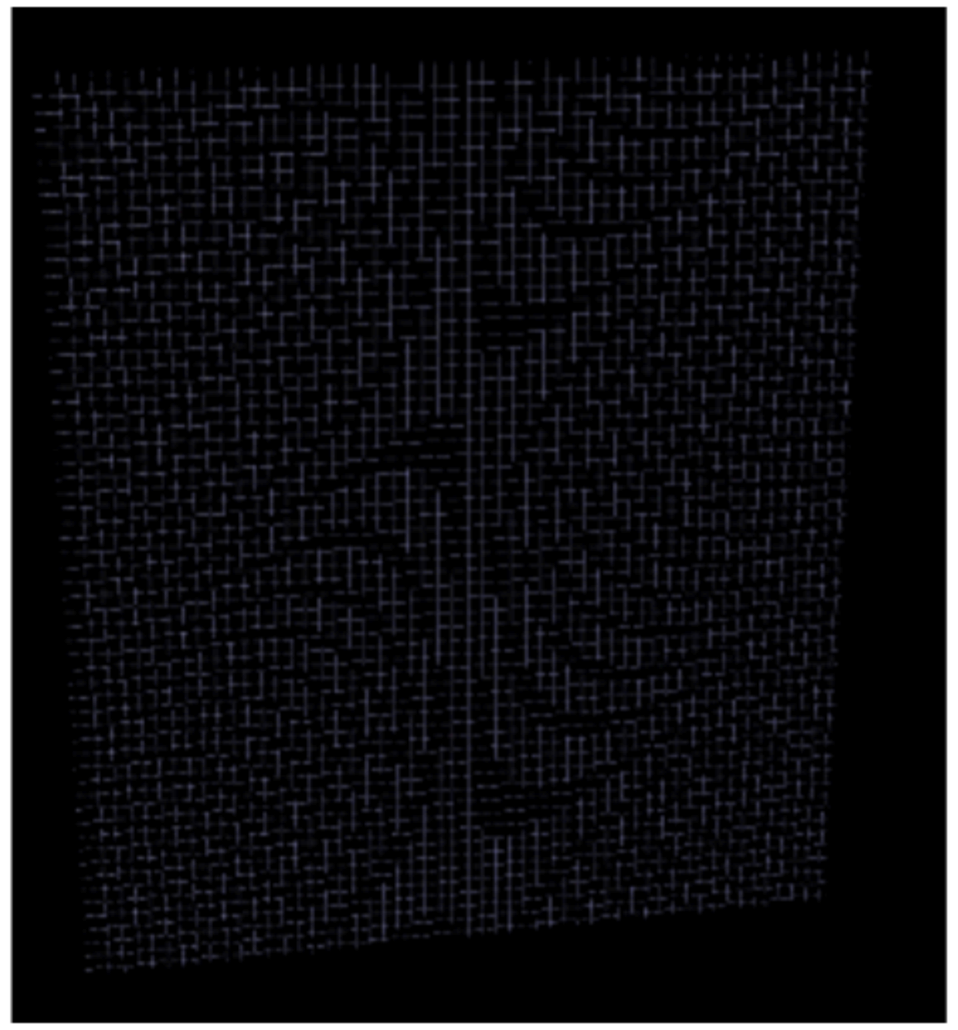
Some AI-made Mistakes


Field cage ring, consists of corners, short edge, long edge etc., was not constructed correctly


Cathode mesh was constructed incorrectly

Beam pipe location was not correct


X-arapuca mesh was not constructed correctly

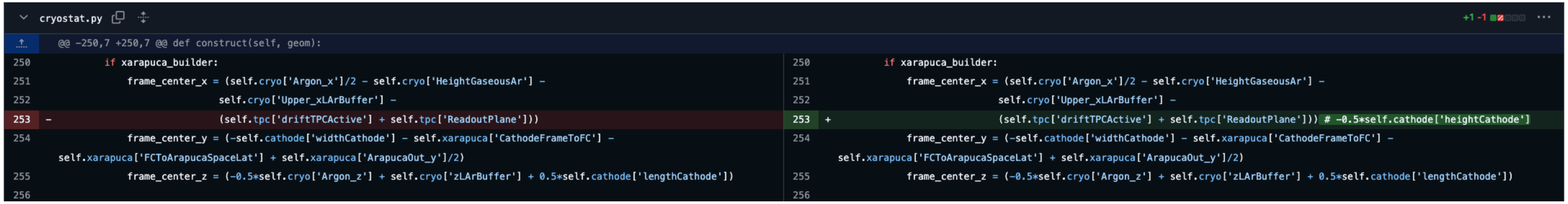


- **Initial AI Output**: For most component builders, the initial code generated by AI often contains some inaccuracies
- **Common Mistakes**: Once identified, these errors are typically minor, such as: Sign errors in equations, Confusion between axes (e.g., x vs. y), Misinterpretation of naming conventions, Misunderstanding of units etc
- **Effective Debugging**: The key to success lies in debugging skills, which enable human to efficiently "find the needle in the haystack" and resolve these issues

Validation

ProtoDUNE-VD

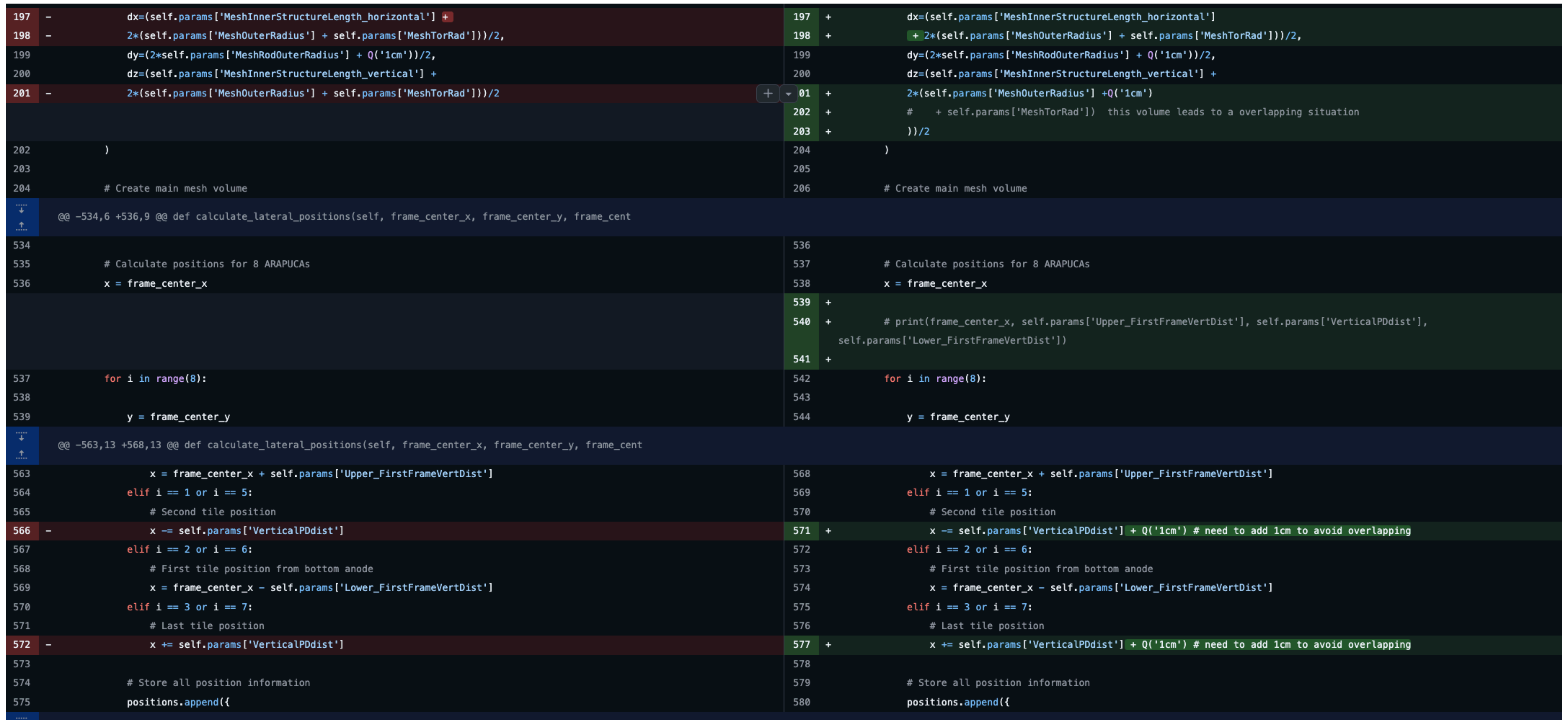
- Another component of validation is to check for volume vs volume overlaps using `TGeoChecker`. Example script by Xin : https://github.com/lastgeorge/gegede_protodune_vd/blob/main/check_overlap.C



```
cryostat.py                                                    +1 -1
@@ -250,7 +250,7 @@ def construct(self, geom):
```

```
        if xarapuca_builder:
            frame_center_x = (self.cryo['Argon_x']/2 - self.cryo['HeightGaseousAr'] -
                                self.cryo['Upper_xLArBuffer'] -
-                               (self.tpc['driftTPCActive'] + self.tpc['ReadoutPlane']))
            frame_center_y = (-self.cathode['widthCathode'] - self.xarapuca['CathodeFrameToFC'] -
self.xarapuca['FCToArapucaSpaceLat'] + self.xarapuca['ArapucaOut_y']/2)
            frame_center_z = (-0.5*self.cryo['Argon_z'] + self.cryo['zLArBuffer'] + 0.5*self.cathode['lengthCathode'])
```

```
        if xarapuca_builder:
            frame_center_x = (self.cryo['Argon_x']/2 - self.cryo['HeightGaseousAr'] -
                                self.cryo['Upper_xLArBuffer'] -
+                               (self.tpc['driftTPCActive'] + self.tpc['ReadoutPlane'])) # -0.5*self.cathode['heightCathode']
            frame_center_y = (-self.cathode['widthCathode'] - self.xarapuca['CathodeFrameToFC'] -
self.xarapuca['FCToArapucaSpaceLat'] + self.xarapuca['ArapucaOut_y']/2)
            frame_center_z = (-0.5*self.cryo['Argon_z'] + self.cryo['zLArBuffer'] + 0.5*self.cathode['lengthCathode'])
```



```
-                   dx=(self.params['MeshInnerStructureLength_horizontal'] +
-                   2*(self.params['MeshOuterRadius'] + self.params['MeshTorRad']))/2,
                    dy=(2*self.params['MeshRodOuterRadius'] + Q('1cm'))/2,
                    dz=(self.params['MeshInnerStructureLength_vertical'] +
-                   2*(self.params['MeshOuterRadius'] + self.params['MeshTorRad']))/2
                )

                # Create main mesh volume
@@ -534,6 +536,9 @@ def calculate_lateral_positions(self, frame_center_x, frame_center_y, frame_cent

                # Calculate positions for 8 ARAPUCAs
                x = frame_center_x
                for i in range(8):

                    y = frame_center_y
@@ -563,13 +568,13 @@ def calculate_lateral_positions(self, frame_center_x, frame_center_y, frame_cent
                        x = frame_center_x + self.params['Upper_FirstFrameVertDist']
                    elif i == 1 or i == 5:
                        # Second tile position
-                       x -= self.params['VerticalPDdist']
                    elif i == 2 or i == 6:
                        # First tile position from bottom anode
                        x = frame_center_x - self.params['Lower_FirstFrameVertDist']
                    elif i == 3 or i == 7:
                        # Last tile position
-                       x += self.params['VerticalPDdist']

                # Store all position information
                positions.append({
```

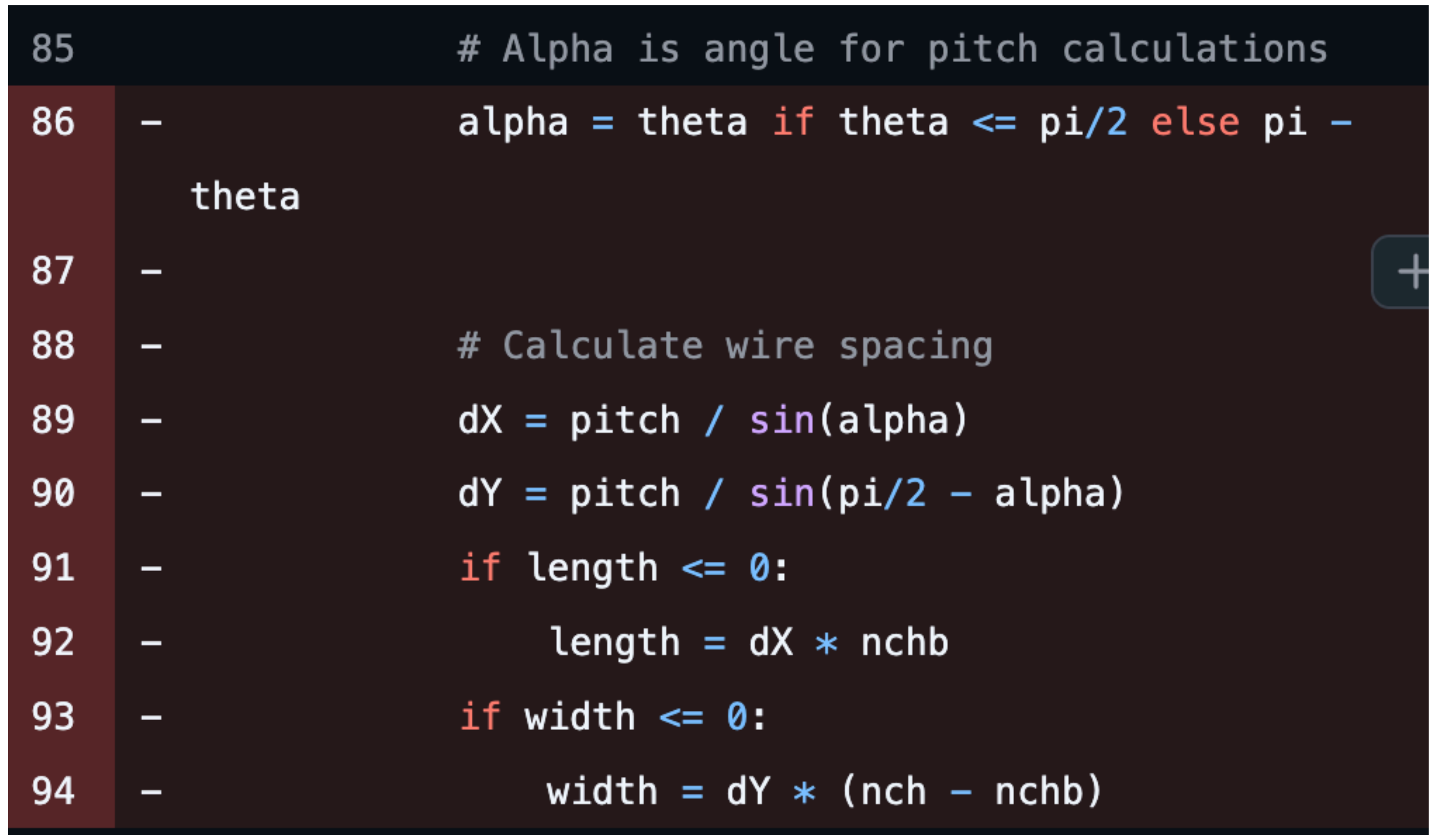
```
+                   dx=(self.params['MeshInnerStructureLength_horizontal']
+                   + 2*(self.params['MeshOuterRadius'] + self.params['MeshTorRad']))/2,
                    dy=(2*self.params['MeshRodOuterRadius'] + Q('1cm'))/2,
                    dz=(self.params['MeshInnerStructureLength_vertical'] +
+                   2*(self.params['MeshOuterRadius'] +Q('1cm')
+                   #    + self.params['MeshTorRad'])  this volume leads to a overlapping situation
+                   ))/2
                )

                # Create main mesh volume


                # Calculate positions for 8 ARAPUCAs
                x = frame_center_x
+
+               # print(frame_center_x, self.params['Upper_FirstFrameVertDist'], self.params['VerticalPDdist'],
self.params['Lower_FirstFrameVertDist'])
+
                for i in range(8):

                    y = frame_center_y

                        x = frame_center_x + self.params['Upper_FirstFrameVertDist']
                    elif i == 1 or i == 5:
                        # Second tile position
+                       x -= self.params['VerticalPDdist'] + Q('1cm') # need to add 1cm to avoid overlapping
                    elif i == 2 or i == 6:
                        # First tile position from bottom anode
                        x = frame_center_x - self.params['Lower_FirstFrameVertDist']
                    elif i == 3 or i == 7:
                        # Last tile position
+                       x += self.params['VerticalPDdist'] + Q('1cm') # need to add 1cm to avoid overlapping

                # Store all position information
                positions.append({
```

- Seems like the xArapucas were being accidentally put inside the LAr volume a little bit
- Currently, Xin moved these by 1cm to prevent this overlap
  - Full changes here
- Present in Perl script as well
- Maybe need some discussion with geometry experts for better implementation

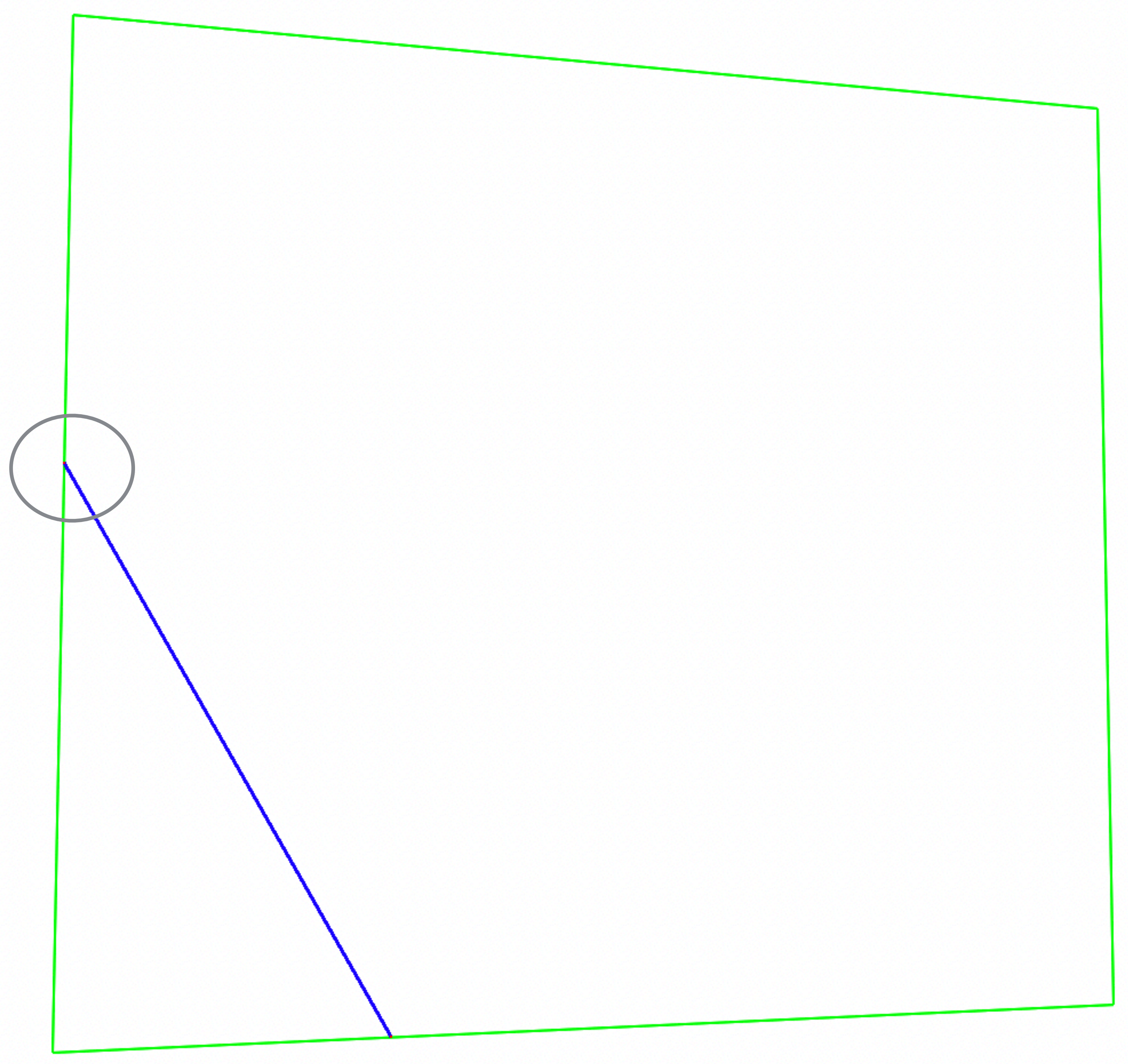
Validation

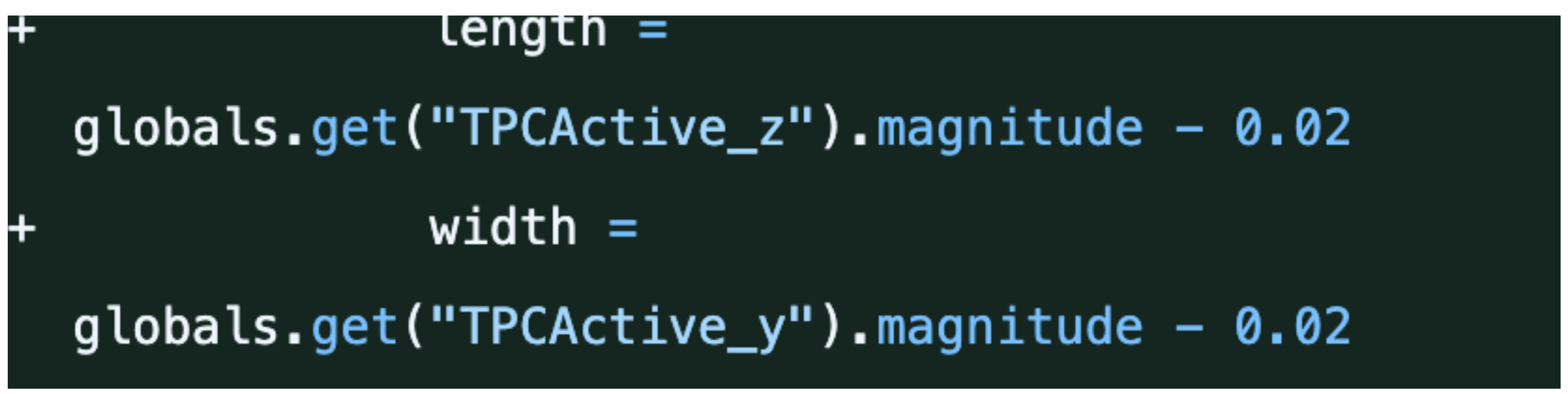
ProtoDUNE-VD

- Some more overlaps of wires with boundary of active TPC volume
  - Hard to spot, exists in perl script as well
- Easy fix, just reducing length by 0.2mm removes all the overlaps
  - Both in FD-VD and PD-VD



```
                # Alpha is angle for pitch calculations
-               alpha = theta if theta <= pi/2 else pi -
  theta
-
-               # Calculate wire spacing
-               dX = pitch / sin(alpha)
-               dY = pitch / sin(pi/2 - alpha)
-               if length <= 0:
-                   length = dX * nchb
-               if width <= 0:
-                   width = dY * (nch - nchb)
```



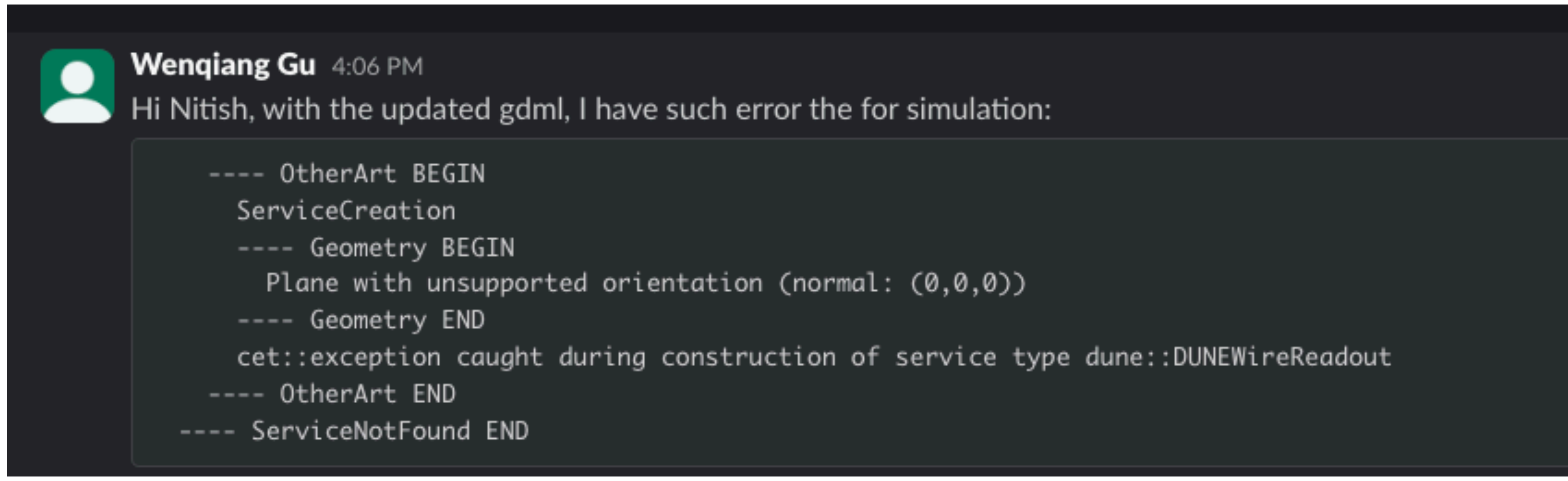


```
+               length =
  globals.get("TPCActive_z").magnitude - 0.02
+               width =
  globals.get("TPCActive_y").magnitude - 0.02
```

Validation

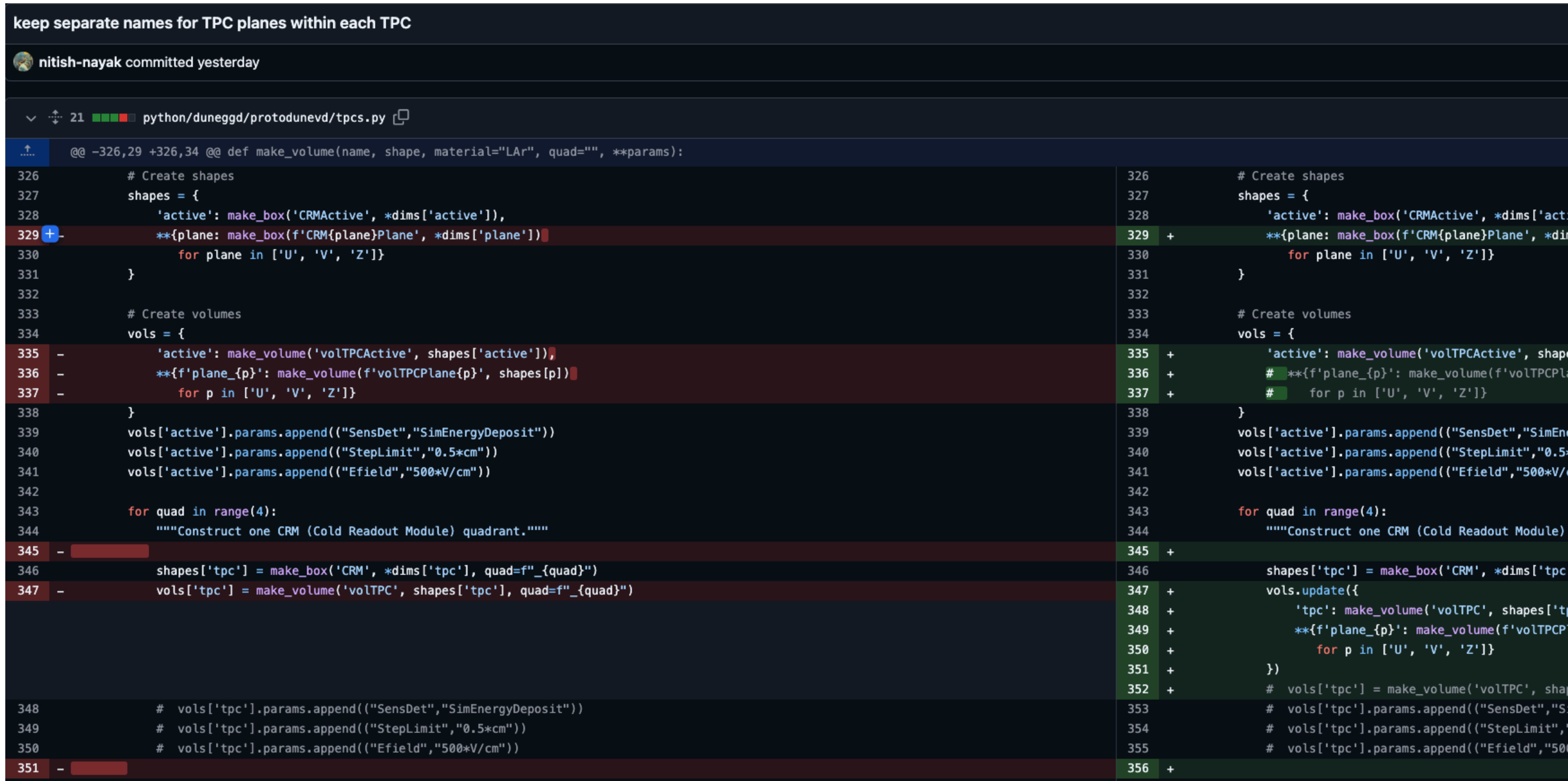
ProtoDUNE-VD

- Wenqiang has been testing workflow with larsoft (generator + g4 stage)
  - Caught some more issues
  - Related to larsoft assumptions for naming schemes and volume placements



Wenqiang Gu 4:06 PM
Hi Nitish, with the updated gdml, I have such error the for simulation:

```
  ---- OtherArt BEGIN
    ServiceCreation
    ---- Geometry BEGIN
      Plane with unsupported orientation (normal: (0,0,0))
    ---- Geometry END
    cet::exception caught during construction of service type dune::DUNEWireReadout
  ---- OtherArt END
---- ServiceNotFound END
```



keep separate names for TPC planes within each TPC

nitish-nayak committed yesterday

21 python/duneggd/protodunevd/tpcs.py

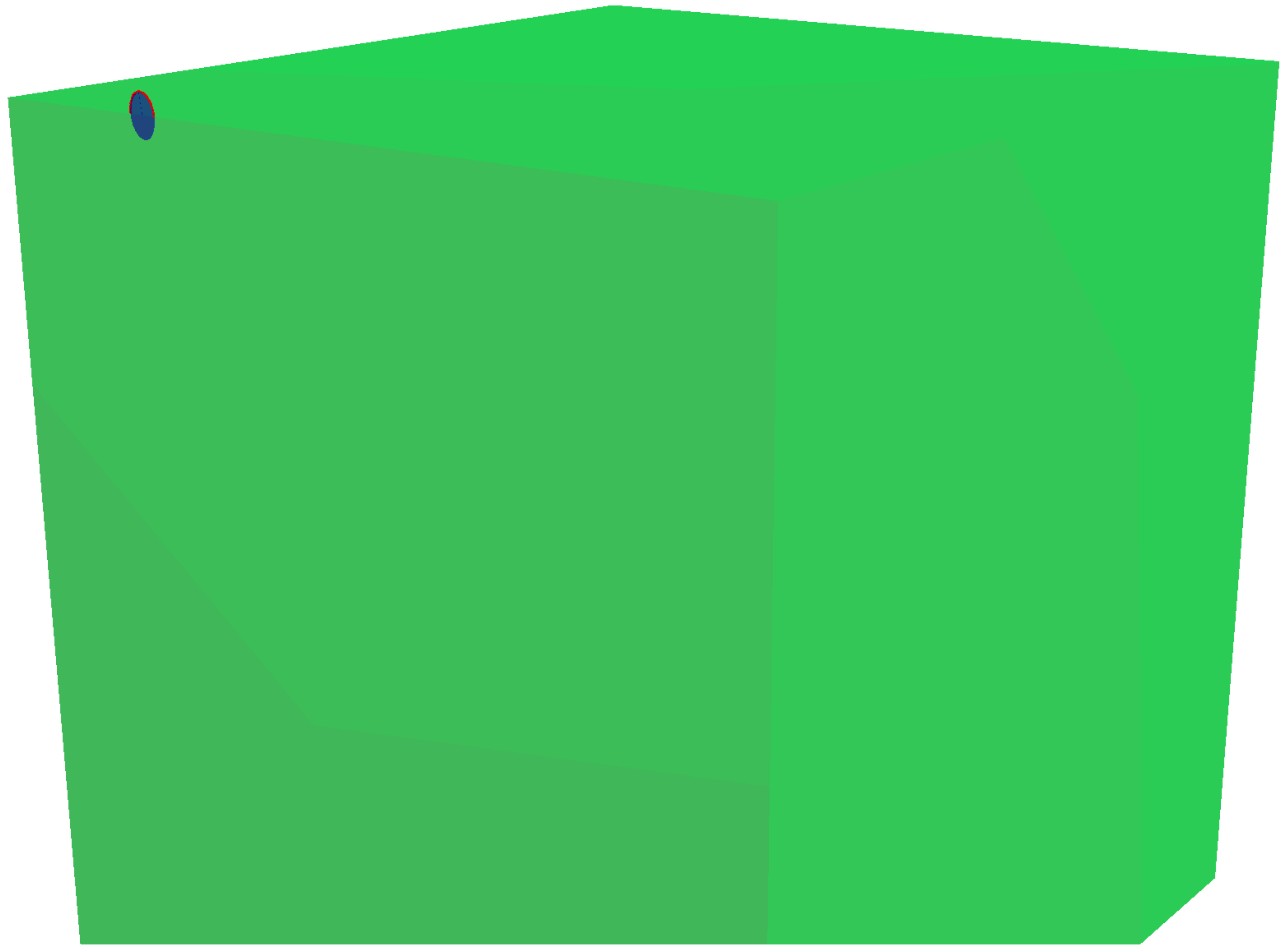
```
@@ -326,29 +326,34 @@ def make_volume(name, shape, material="LAr", quad="", **params):
         # Create shapes
         shapes = {
             'active': make_box('CRMActive', *dims['active']),
-            **{plane: make_box(f'CRM{plane}Plane', *dims['plane'])
               for plane in ['U', 'V', 'Z']}
         }

         # Create volumes
         vols = {
-            'active': make_volume('volTPCActive', shapes['active']),
-            **{f'plane_{p}': make_volume(f'volTPCPlane{p}', shapes[p])
-               for p in ['U', 'V', 'Z']}
         }
         vols['active'].params.append(("SensDet","SimEnergyDeposit"))
         vols['active'].params.append(("StepLimit","0.5*cm"))
         vols['active'].params.append(("Efield","500*V/cm"))

         for quad in range(4):
             """Construct one CRM (Cold Readout Module) quadrant."""
-
             shapes['tpc'] = make_box('CRM', *dims['tpc'], quad=f"_{quad}")
-            vols['tpc'] = make_volume('volTPC', shapes['tpc'], quad=f"_{quad}")




             #  vols['tpc'].params.append(("SensDet","SimEnergyDeposit"))
             #  vols['tpc'].params.append(("StepLimit","0.5*cm"))
             #  vols['tpc'].params.append(("Efield","500*V/cm"))
-
```

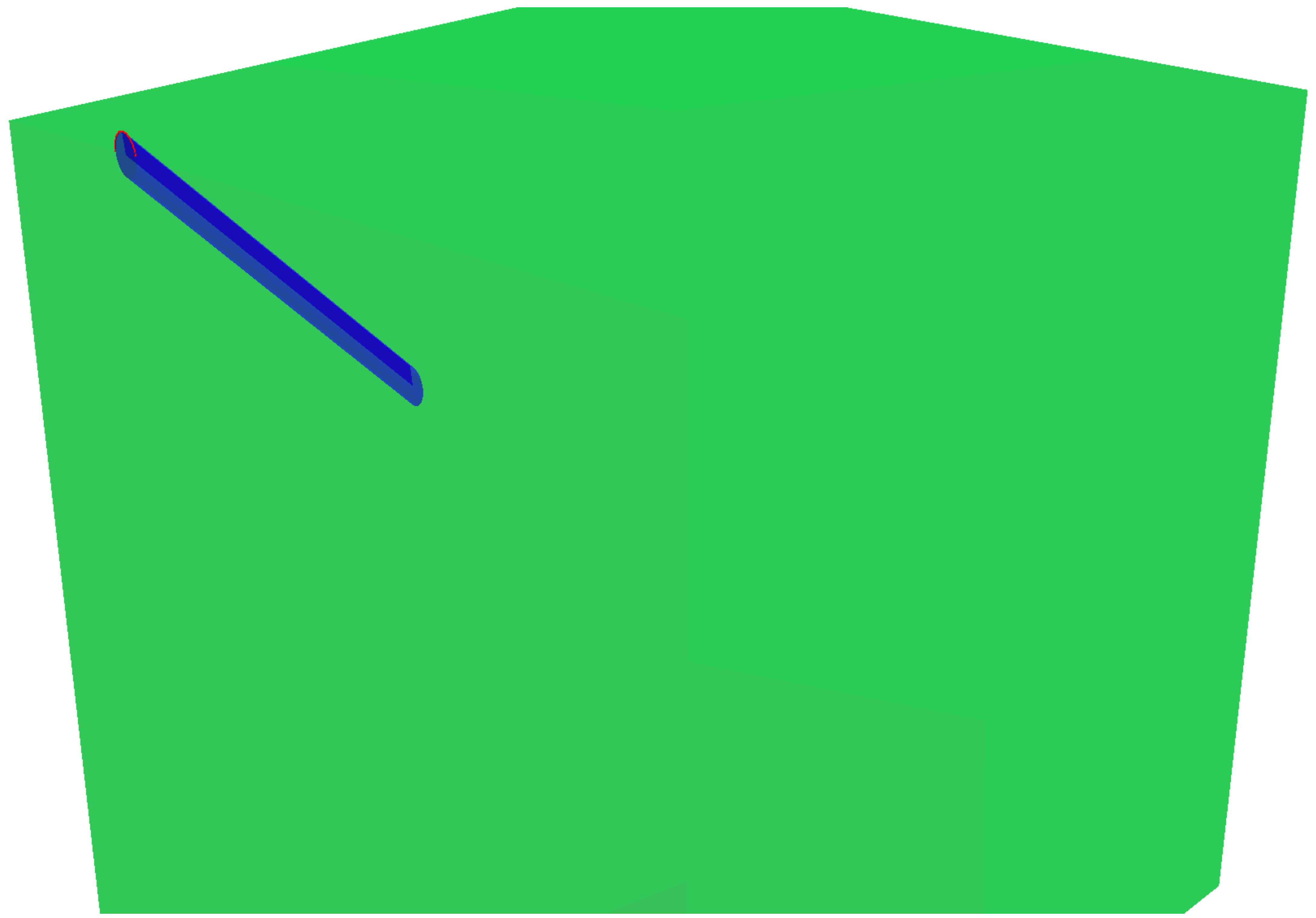
```
         # Create shapes
         shapes = {
             'active': make_box('CRMActive', *dims['act
+            **{plane: make_box(f'CRM{plane}Plane', *di
               for plane in ['U', 'V', 'Z']}
         }

         # Create volumes
         vols = {
+            'active': make_volume('volTPCActive', shap
+            #  **{f'plane_{p}': make_volume(f'volTPCPl
+            #     for p in ['U', 'V', 'Z']}
         }
         vols['active'].params.append(("SensDet","SimEn
         vols['active'].params.append(("StepLimit","0.5
         vols['active'].params.append(("Efield","500*V/

         for quad in range(4):
             """Construct one CRM (Cold Readout Module)
+
             shapes['tpc'] = make_box('CRM', *dims['tpc
+            vols.update({
+                'tpc': make_volume('volTPC', shapes['t
+                **{f'plane_{p}': make_volume(f'volTPCP
+                   for p in ['U', 'V', 'Z']}
+            })
+            #  vols['tpc'] = make_volume('volTPC', sha
             #  vols['tpc'].params.append(("SensDet","S
             #  vols['tpc'].params.append(("StepLimit",
             #  vols['tpc'].params.append(("Efield","50
+
```

- Everything runs now!

Validation

ProtoDUNE-VD

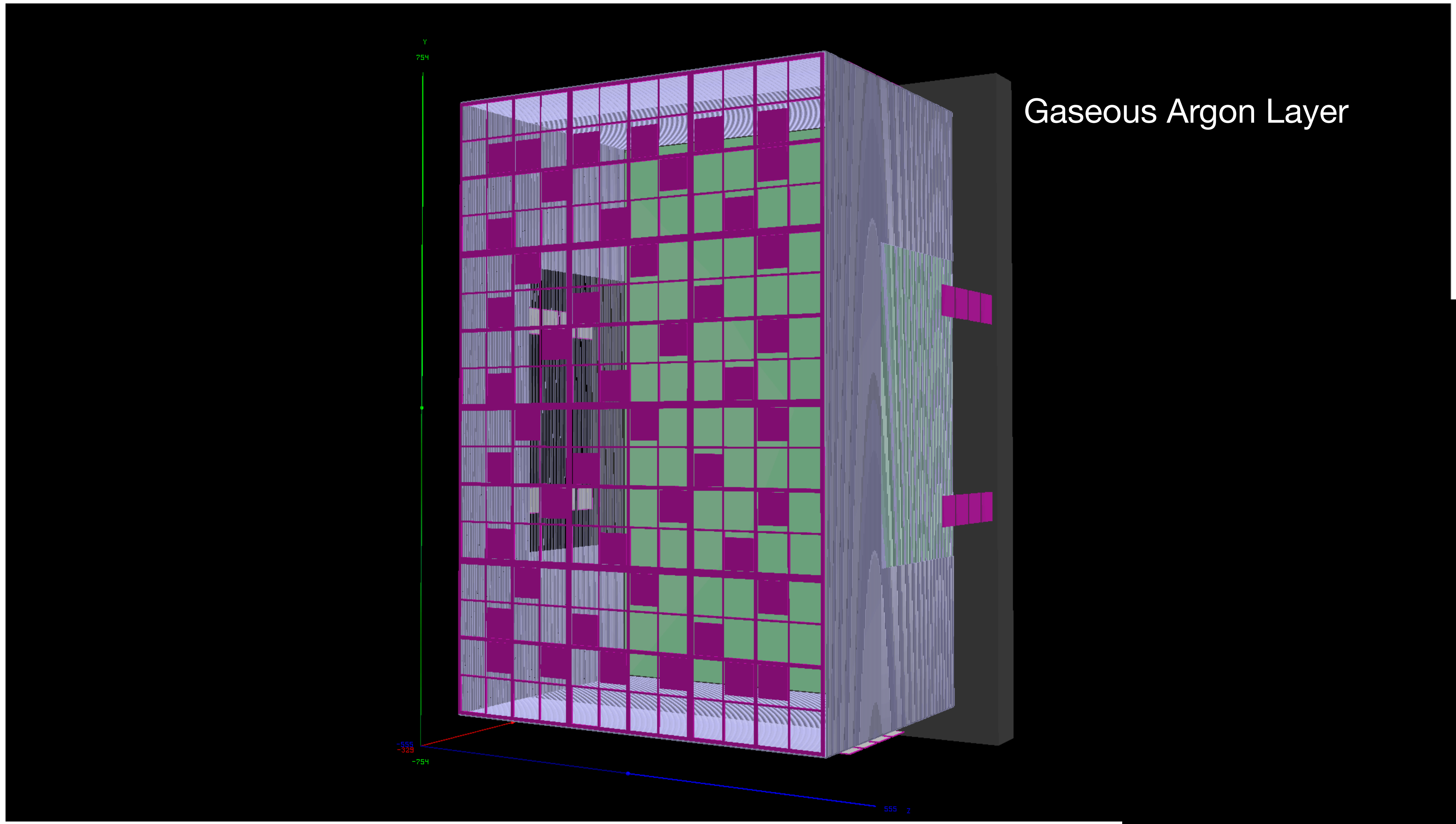


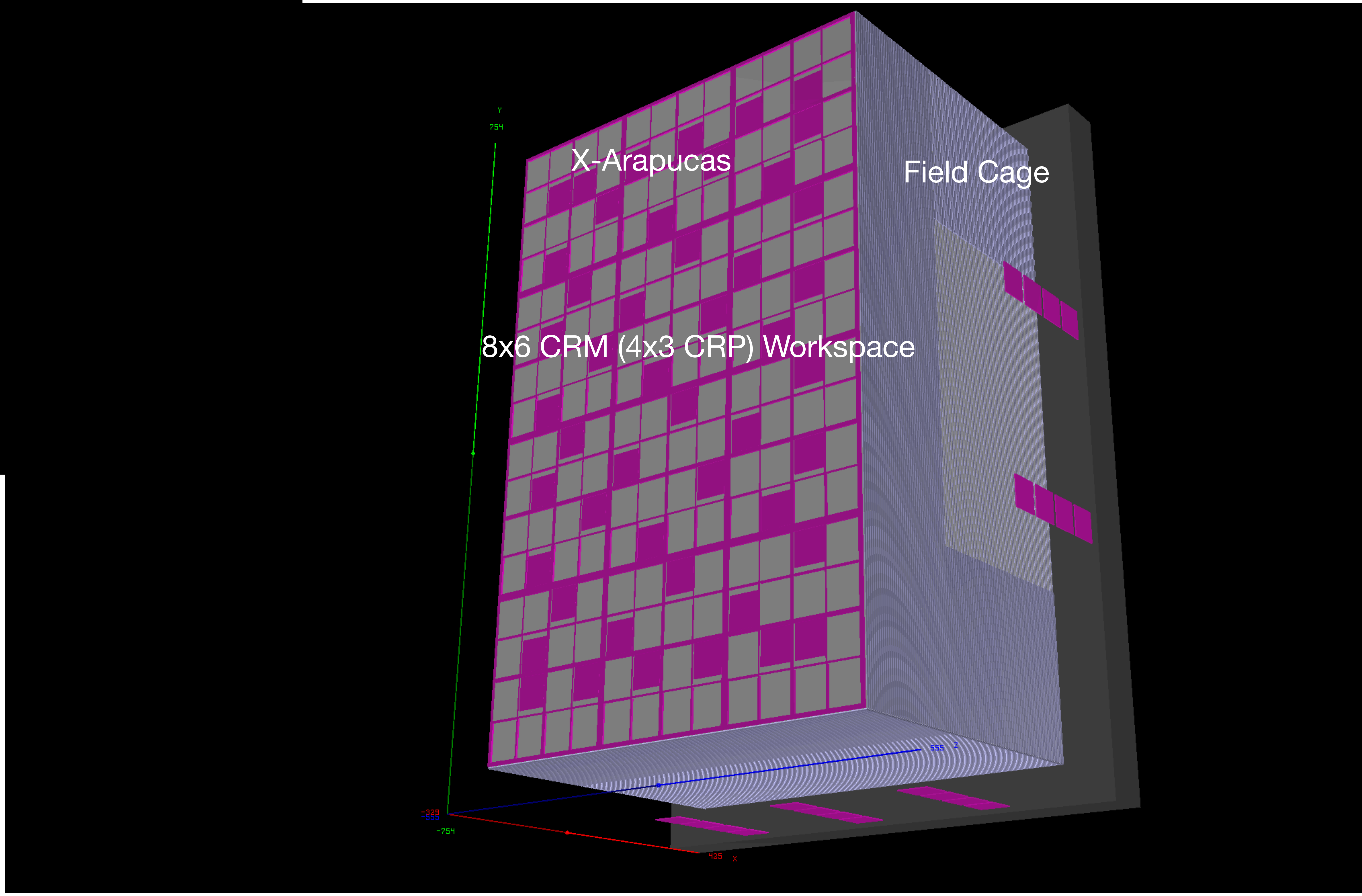


- Last remaining overlaps (exists in PERL script as well)
  - Extrusions from beam plug and upstream end cap
  - Looking into breaking off the extruded volume into its own object to remove these overlaps (a bit tricky because it has sub volumes of its own)
- I don't think this matters (based on Wenqiang's tests) and might require more code surgery than is worth, but will try to sit with it a bit more, maybe with help from Claude etc

Validation

FD-VD





- All sub-components have been implemented
- Validated that this implementation is functionally consistent with perl script
- Removed all overlaps, with and without wires

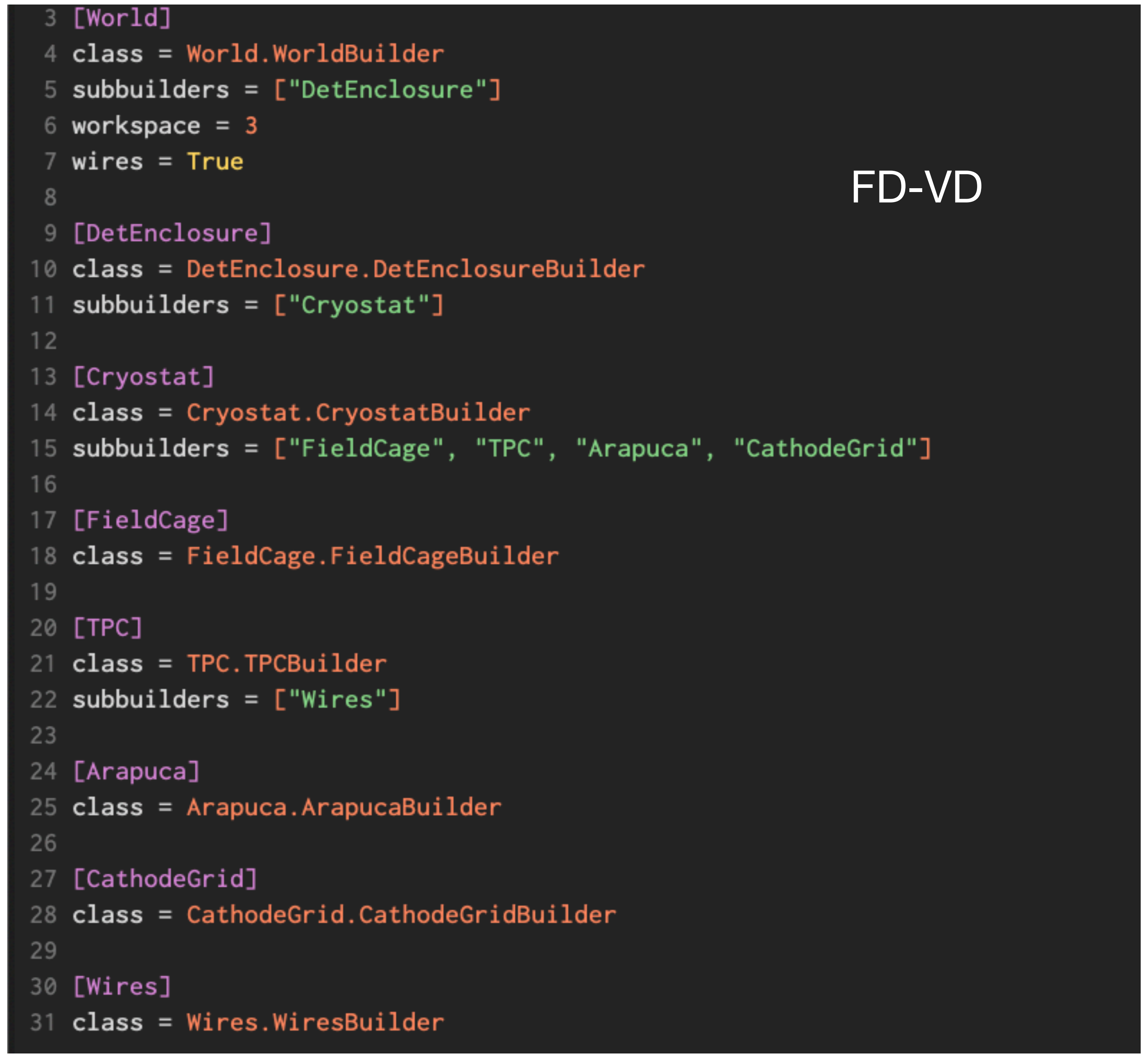
Summary & Next Steps

- GGD is a very useful tool for geometry construction in DUNE
  - Improvements in code modularity, maintainability, extensibility etc
  - Much easier to configure and thereby optimize GEANT running for our complicated geometries
- Xin and I have implemented new constructors within this framework for VD, both ProtoDUNE and FD (for all current workspaces)
  - Corresponding to latest versions of geometry (v4 for ProtoDUNE, v6 for FD)
  - Validation basically complete, everything runs. Less overlaps than Perl Script as well
  - PR in duneggd repository : https://github.com/DUNE/duneggd/pull/13
  - Ready to use - further validation ongoing (Biao Wang et al. - Alabama, Edinburgh groups)
- Next Steps (order of priority) :
  - See how feasible it is to fix beam pipe overlaps
  - Merging changes to shotcrete and rock layer in HD to FD-VD geometry as well
  - Merge code structure across PD, FD (both HD and VD)
  - Any others?

Backup

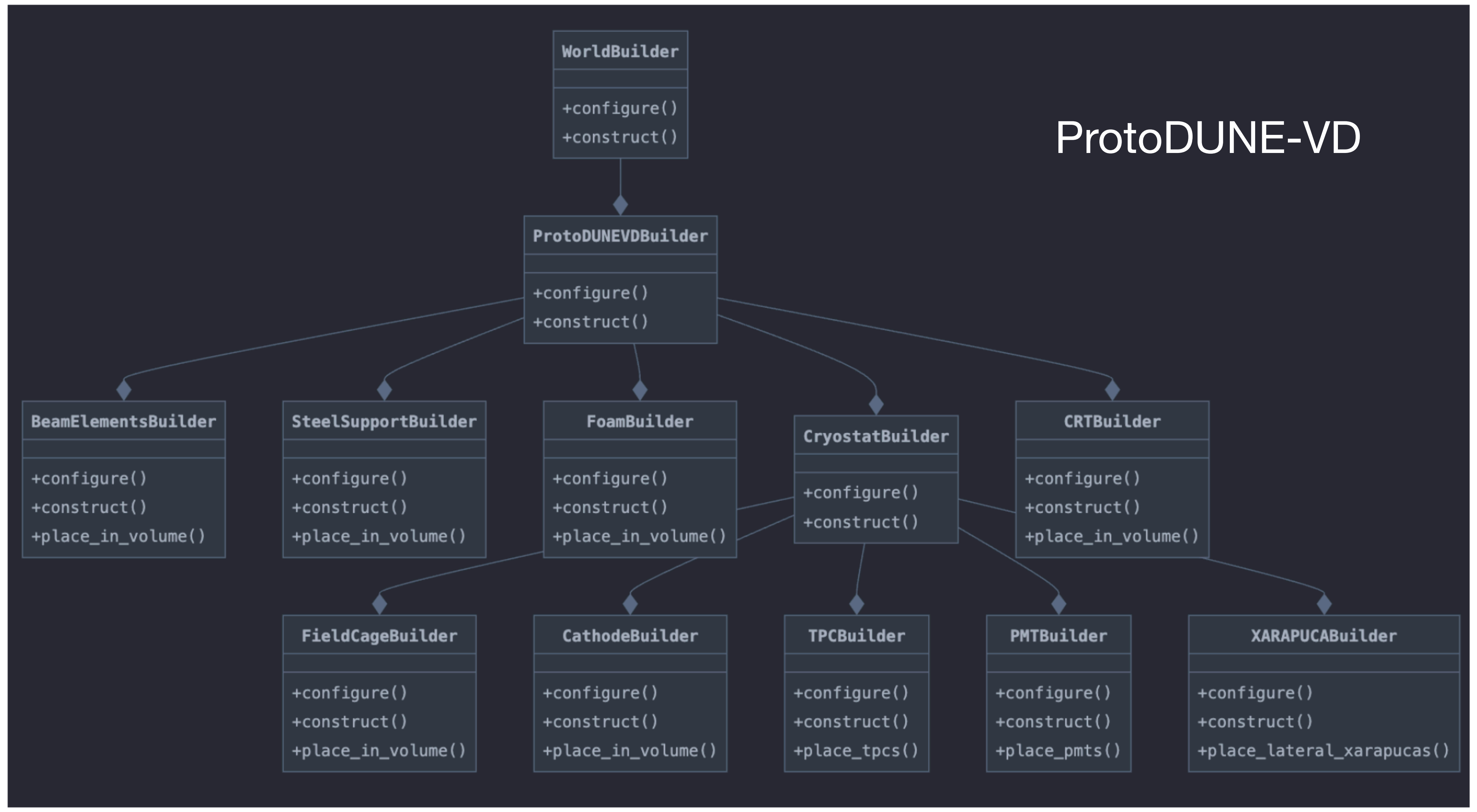
Overview

- GGD configuration overview
  - Each builder can have multiple sub-builders enclosing different logical volumes
  - Each builder can be configured separately
  - Beyond construction itself, each builder responsible for how the LVs from sub-builders are placed within its own LV
  - GGD uses this to then emit the GDML based on its schema



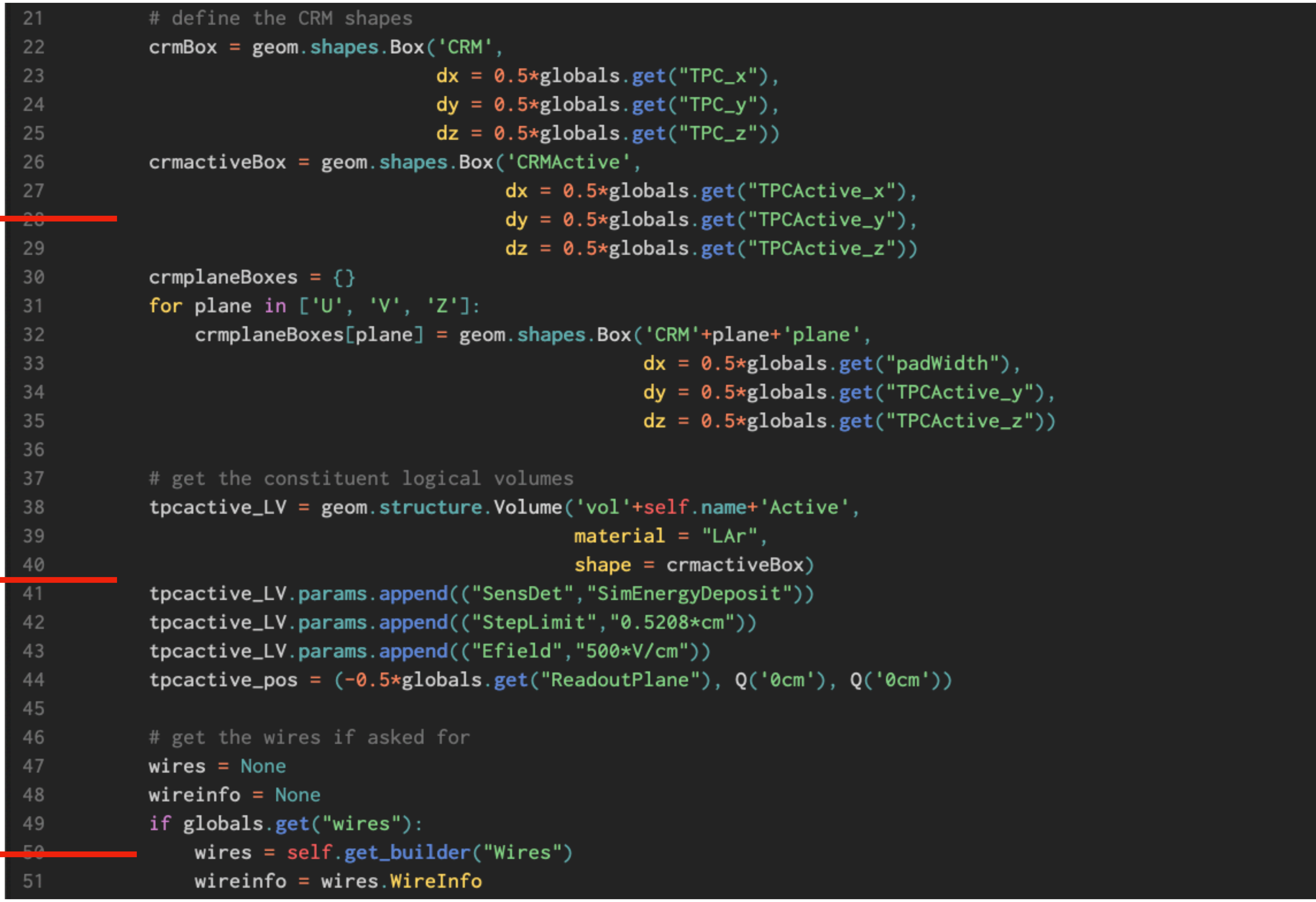
FD-VD

```
[World]
class = World.WorldBuilder
subbuilders = ["DetEnclosure"]
workspace = 3
wires = True

[DetEnclosure]
class = DetEnclosure.DetEnclosureBuilder
subbuilders = ["Cryostat"]

[Cryostat]
class = Cryostat.CryostatBuilder
subbuilders = ["FieldCage", "TPC", "Arapuca", "CathodeGrid"]

[FieldCage]
class = FieldCage.FieldCageBuilder

[TPC]
class = TPC.TPCBuilder
subbuilders = ["Wires"]

[Arapuca]
class = Arapuca.ArapucaBuilder

[CathodeGrid]
class = CathodeGrid.CathodeGridBuilder

[Wires]
class = Wires.WiresBuilder
```



Builders

TPC Example (FD-VD)

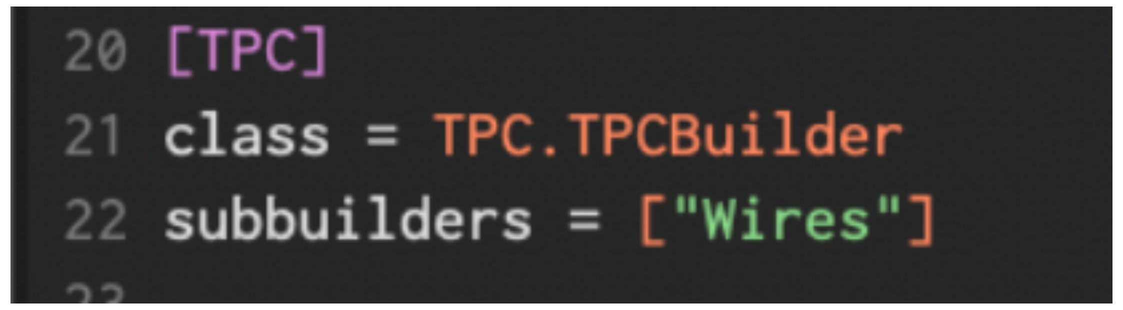


```python
        # define the CRM shapes
        crmBox = geom.shapes.Box('CRM',
                                 dx = 0.5*globals.get("TPC_x"),
                                 dy = 0.5*globals.get("TPC_y"),
                                 dz = 0.5*globals.get("TPC_z"))
        crmactiveBox = geom.shapes.Box('CRMActive',
                                       dx = 0.5*globals.get("TPCActive_x"),
                                       dy = 0.5*globals.get("TPCActive_y"),
                                       dz = 0.5*globals.get("TPCActive_z"))
        crmplaneBoxes = {}
        for plane in ['U', 'V', 'Z']:
            crmplaneBoxes[plane] = geom.shapes.Box('CRM'+plane+'plane',
                                                   dx = 0.5*globals.get("padWidth"),
                                                   dy = 0.5*globals.get("TPCActive_y"),
                                                   dz = 0.5*globals.get("TPCActive_z"))

        # get the constituent logical volumes
        tpcactive_LV = geom.structure.Volume('vol'+self.name+'Active',
                                             material = "LAr",
                                             shape = crmactiveBox)
        tpcactive_LV.params.append(("SensDet","SimEnergyDeposit"))
        tpcactive_LV.params.append(("StepLimit","0.5208*cm"))
        tpcactive_LV.params.append(("Efield","500*V/cm"))
        tpcactive_pos = (-0.5*globals.get("ReadoutPlane"), Q('0cm'), Q('0cm'))

        # get the wires if asked for
        wires = None
        wireinfo = None
        if globals.get("wires"):
            wires = self.get_builder("Wires")
            wireinfo = wires.WireInfo
```

Define shapes for charge readout planes

Define its corresponding logical volume

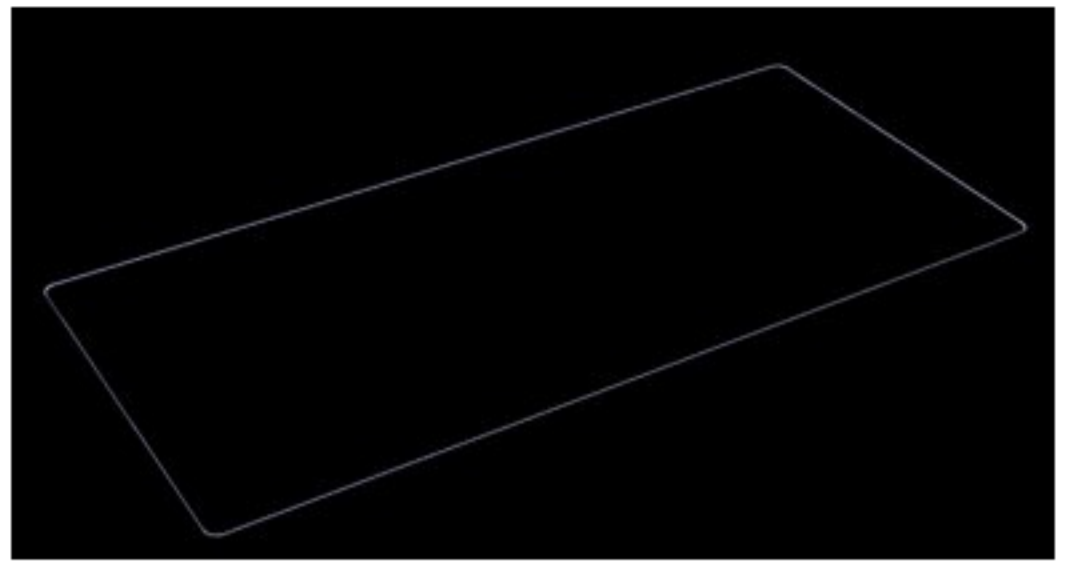
Access the sub-builder (Wires)
as defined by the configuration

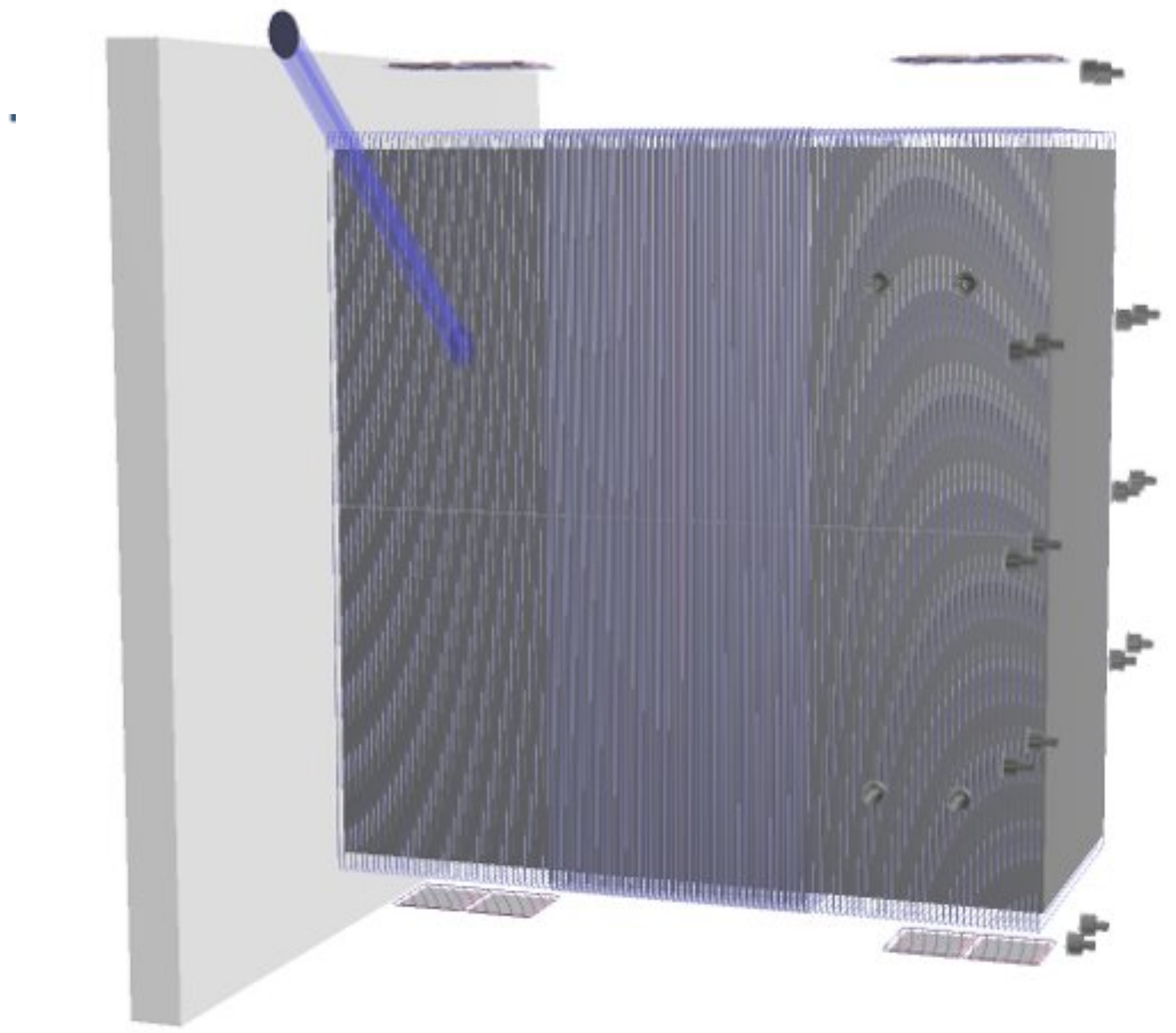


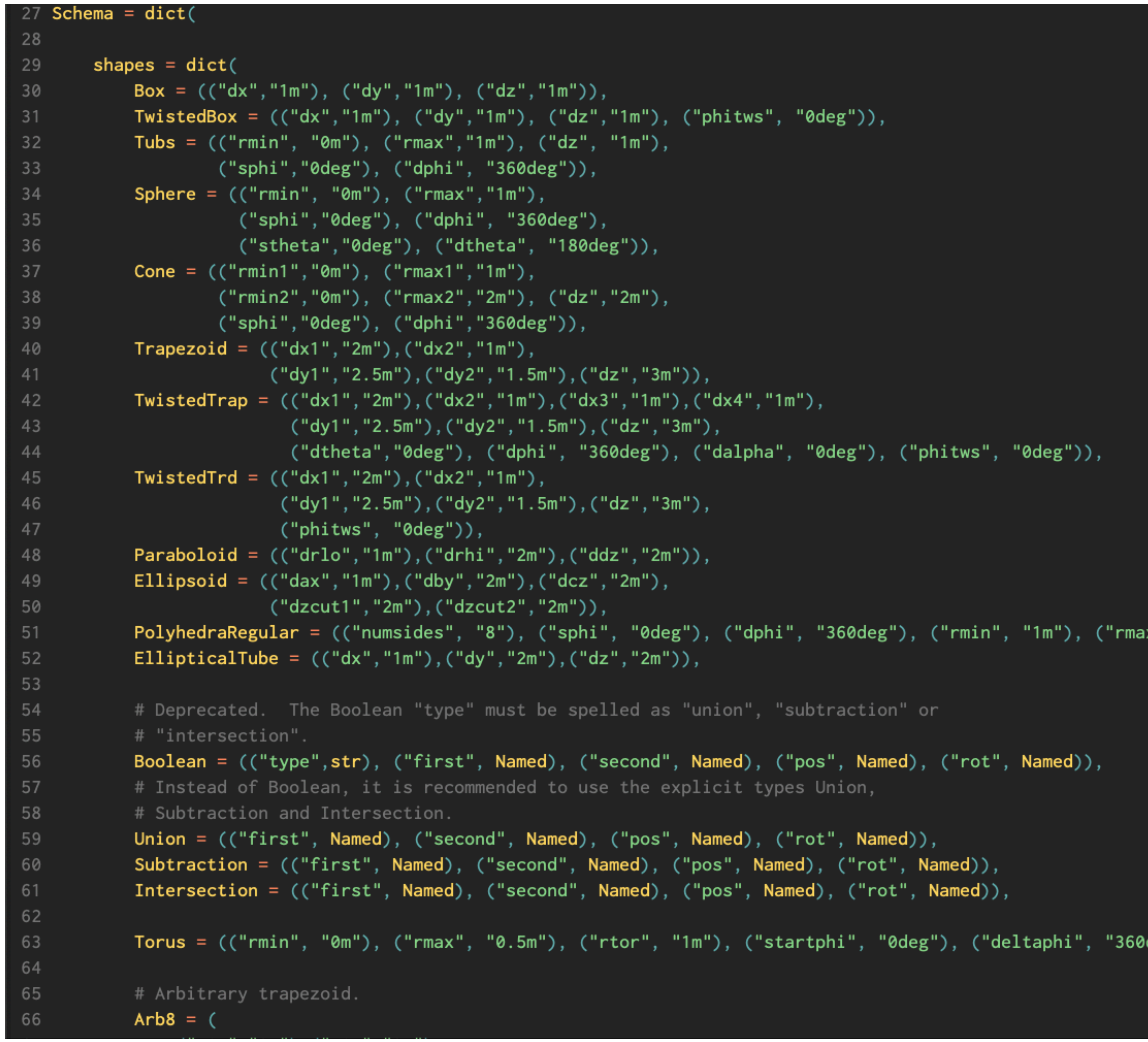
```
[TPC]
class = TPC.TPCBuilder
subbuilders = ["Wires"]
```

Shapes

- Shapes defined based on GDML schema
  - Can transparently add new ones as needed
  - Brett added new ones (CutTubs, ExtrudeMany) recently as well
    - For Field Cage Rings and Beam pipe volumes in PD-VD


One Slim Field Cage Ring

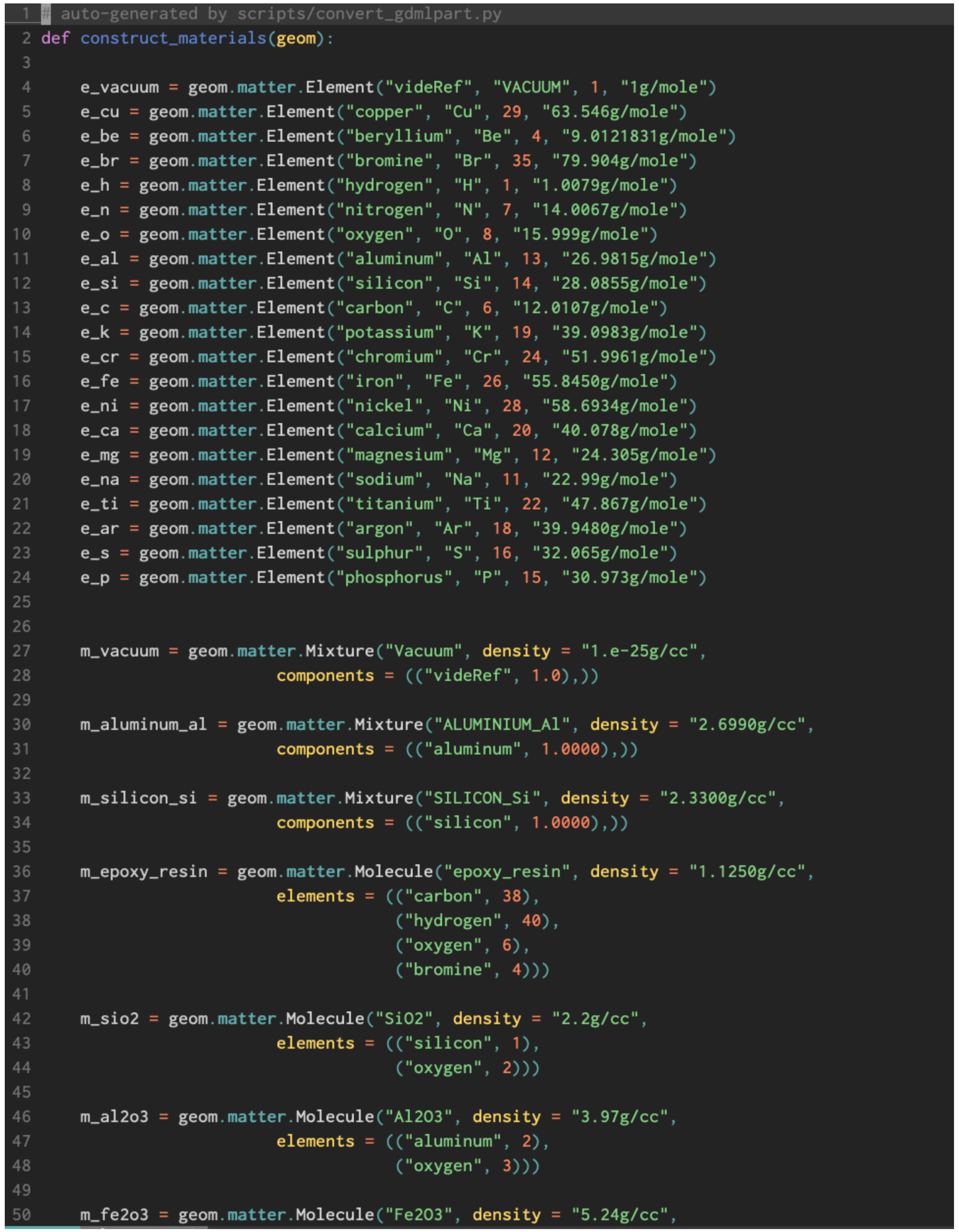




```
Schema = dict(

    shapes = dict(
        Box = (("dx","1m"), ("dy","1m"), ("dz","1m")),
        TwistedBox = (("dx","1m"), ("dy","1m"), ("dz","1m"), ("phitws", "0deg")),
        Tubs = (("rmin", "0m"), ("rmax","1m"), ("dz", "1m"),
                ("sphi","0deg"), ("dphi", "360deg")),
        Sphere = (("rmin", "0m"), ("rmax","1m"),
                  ("sphi","0deg"), ("dphi", "360deg"),
                  ("stheta","0deg"), ("dtheta", "180deg")),
        Cone = (("rmin1","0m"), ("rmax1","1m"),
                ("rmin2","0m"), ("rmax2","2m"), ("dz","2m"),
                ("sphi","0deg"), ("dphi","360deg")),
        Trapezoid = (("dx1","2m"),("dx2","1m"),
                     ("dy1","2.5m"),("dy2","1.5m"),("dz","3m")),
        TwistedTrap = (("dx1","2m"),("dx2","1m"),("dx3","1m"),("dx4","1m"),
                       ("dy1","2.5m"),("dy2","1.5m"),("dz","3m"),
                       ("dtheta","0deg"), ("dphi", "360deg"), ("dalpha", "0deg"), ("phitws", "0deg")),
        TwistedTrd = (("dx1","2m"),("dx2","1m"),
                      ("dy1","2.5m"),("dy2","1.5m"),("dz","3m"),
                      ("phitws", "0deg")),
        Paraboloid = (("drlo","1m"),("drhi","2m"),("ddz","2m")),
        Ellipsoid = (("dax","1m"),("dby","2m"),("dcz","2m"),
                     ("dzcut1","2m"),("dzcut2","2m")),
        PolyhedraRegular = (("numsides", "8"), ("sphi", "0deg"), ("dphi", "360deg"), ("rmin", "1m"), ("rma
        EllipticalTube = (("dx","1m"),("dy","2m"),("dz","2m")),

        # Deprecated.  The Boolean "type" must be spelled as "union", "subtraction" or
        # "intersection".
        Boolean = (("type",str), ("first", Named), ("second", Named), ("pos", Named), ("rot", Named)),
        # Instead of Boolean, it is recommended to use the explicit types Union,
        # Subtraction and Intersection.
        Union = (("first", Named), ("second", Named), ("pos", Named), ("rot", Named)),
        Subtraction = (("first", Named), ("second", Named), ("pos", Named), ("rot", Named)),
        Intersection = (("first", Named), ("second", Named), ("pos", Named), ("rot", Named)),

        Torus = (("rmin", "0m"), ("rmax", "0.5m"), ("rtor", "1m"), ("startphi", "0deg"), ("deltaphi", "360

        # Arbitrary trapezoid.
        Arb8 = (
```

Materials

- GeGeDe implementation for various materials that go into the LV definitions
- Have a script that auto-fills this based on existing Materials file (xml parsing)



```
# auto-generated by scripts/convert_gdmlpart.py
def construct_materials(geom):

    e_vacuum = geom.matter.Element("videRef", "VACUUM", 1, "1g/mole")
    e_cu = geom.matter.Element("copper", "Cu", 29, "63.546g/mole")
    e_be = geom.matter.Element("beryllium", "Be", 4, "9.0121831g/mole")
    e_br = geom.matter.Element("bromine", "Br", 35, "79.904g/mole")
    e_h = geom.matter.Element("hydrogen", "H", 1, "1.0079g/mole")
    e_n = geom.matter.Element("nitrogen", "N", 7, "14.0067g/mole")
    e_o = geom.matter.Element("oxygen", "O", 8, "15.999g/mole")
    e_al = geom.matter.Element("aluminum", "Al", 13, "26.9815g/mole")
    e_si = geom.matter.Element("silicon", "Si", 14, "28.0855g/mole")
    e_c = geom.matter.Element("carbon", "C", 6, "12.0107g/mole")
    e_k = geom.matter.Element("potassium", "K", 19, "39.0983g/mole")
    e_cr = geom.matter.Element("chromium", "Cr", 24, "51.9961g/mole")
    e_fe = geom.matter.Element("iron", "Fe", 26, "55.8450g/mole")
    e_ni = geom.matter.Element("nickel", "Ni", 28, "58.6934g/mole")
    e_ca = geom.matter.Element("calcium", "Ca", 20, "40.078g/mole")
    e_mg = geom.matter.Element("magnesium", "Mg", 12, "24.305g/mole")
    e_na = geom.matter.Element("sodium", "Na", 11, "22.99g/mole")
    e_ti = geom.matter.Element("titanium", "Ti", 22, "47.867g/mole")
    e_ar = geom.matter.Element("argon", "Ar", 18, "39.9480g/mole")
    e_s = geom.matter.Element("sulphur", "S", 16, "32.065g/mole")
    e_p = geom.matter.Element("phosphorus", "P", 15, "30.973g/mole")


    m_vacuum = geom.matter.Mixture("Vacuum", density = "1.e-25g/cc",
                    components = (("videRef", 1.0),))

    m_aluminum_al = geom.matter.Mixture("ALUMINIUM_Al", density = "2.6990g/cc",
                    components = (("aluminum", 1.0000),))

    m_silicon_si = geom.matter.Mixture("SILICON_Si", density = "2.3300g/cc",
                    components = (("silicon", 1.0000),))

    m_epoxy_resin = geom.matter.Molecule("epoxy_resin", density = "1.1250g/cc",
                    elements = (("carbon", 38),
                                ("hydrogen", 40),
                                ("oxygen", 6),
                                ("bromine", 4)))

    m_sio2 = geom.matter.Molecule("SiO2", density = "2.2g/cc",
                    elements = (("silicon", 1),
                                ("oxygen", 2)))

    m_al2o3 = geom.matter.Molecule("Al2O3", density = "3.97g/cc",
                    elements = (("aluminum", 2),
                                ("oxygen", 3)))

    m_fe2o3 = geom.matter.Molecule("Fe2O3", density = "5.24g/cc",
```