LUXE Tracking with ACTs v40

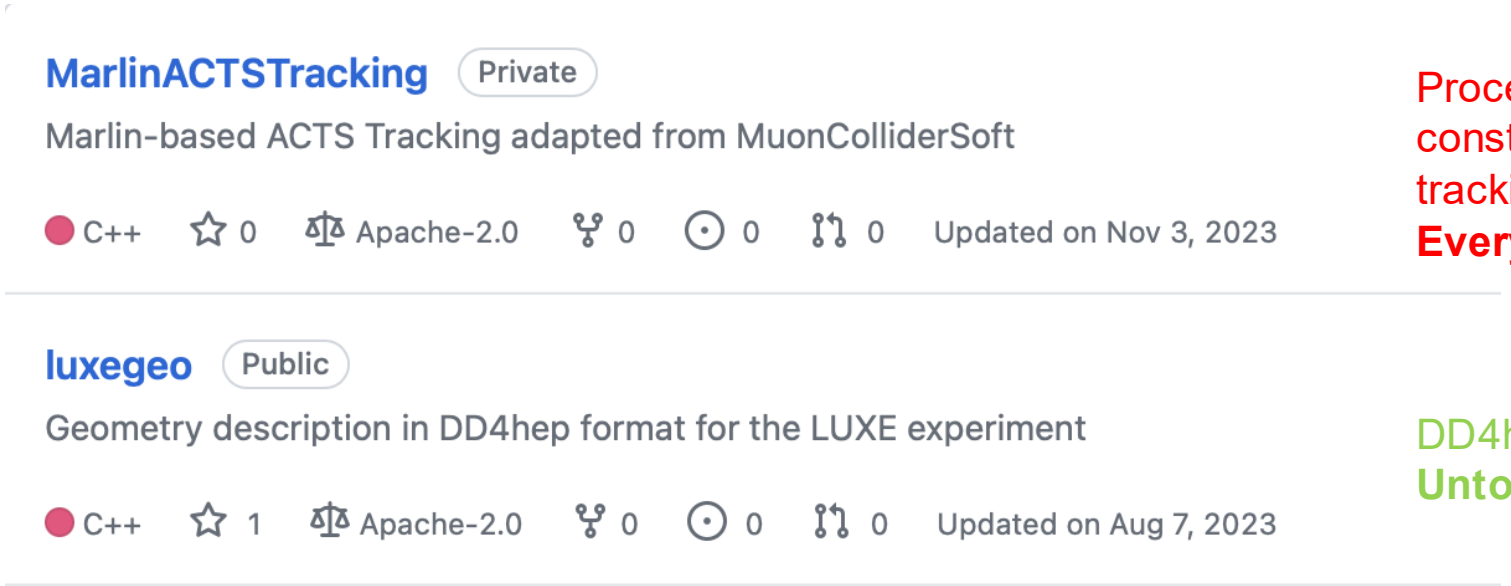
Yufeng Wang
Jul 25, 2025

HELMHOLTZ

DESY.

Motivation

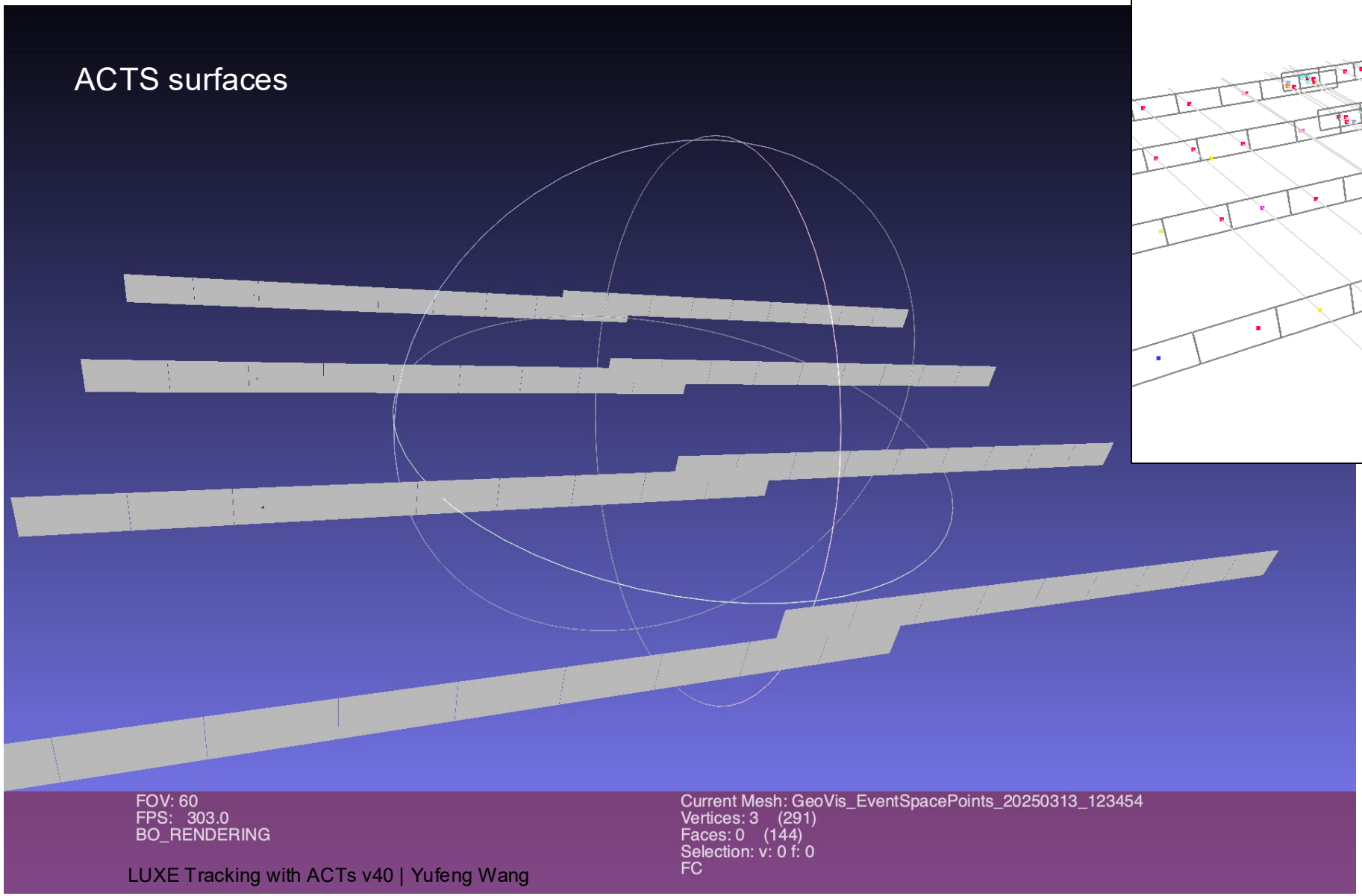
- Continue to update the current Marlin processor for LUXE tracking.
- Major change:
  - ACTs version: 13→32→40 (C++17)
  - DD4hepDetector in ActsExamples can now be used as a class
  - Binning→AxisDirection, better support telescope-type detector
    - But ACTs DD4hep plugin is not entirely updated accordingly, so I hacked it
  - Lots of other updates (new seeding, propagator… since v34)



Processor for ACTS geometry construction, hit conversion, seeding, tracking…
Everything's changed

DD4hep geometry, ddsim
Untouched

Detector



2*9*4=72 DD4hep sensitive volumes/ACTS surfaces

Steer file

```
  <execute>
    <!-- ========== setup  ========== -->
    <processor name="MyAIDAProcessor"/>
    <processor name="EventNumber" />
    <processor name="Config" />

    <!-- ==========  Geometry initialization  ========== -->
    <processor name="InitDD4hep"/>

    <!-- ==========  Tracker Digitization  ========== -->
    <processor name="MyDDPlanarDigiProcessor"/>

    <!-- ==========  Tracking  ========== -->
   <processor name="MyCKFTracking"/>
   <processor name="MyTrackDeduper"/>
   <processor name="MyTrackTruth"/>

    <!-- ==========  Output  ========== -->
<!--    <processor name="MyACTSTuple" />-->
    <processor name="MyLCTuple" />
    <processor name="Output_REC"/>
  </execute>
```

Run with:
Marlin actsseedckf_steer.xml

Tracking with ACTS:

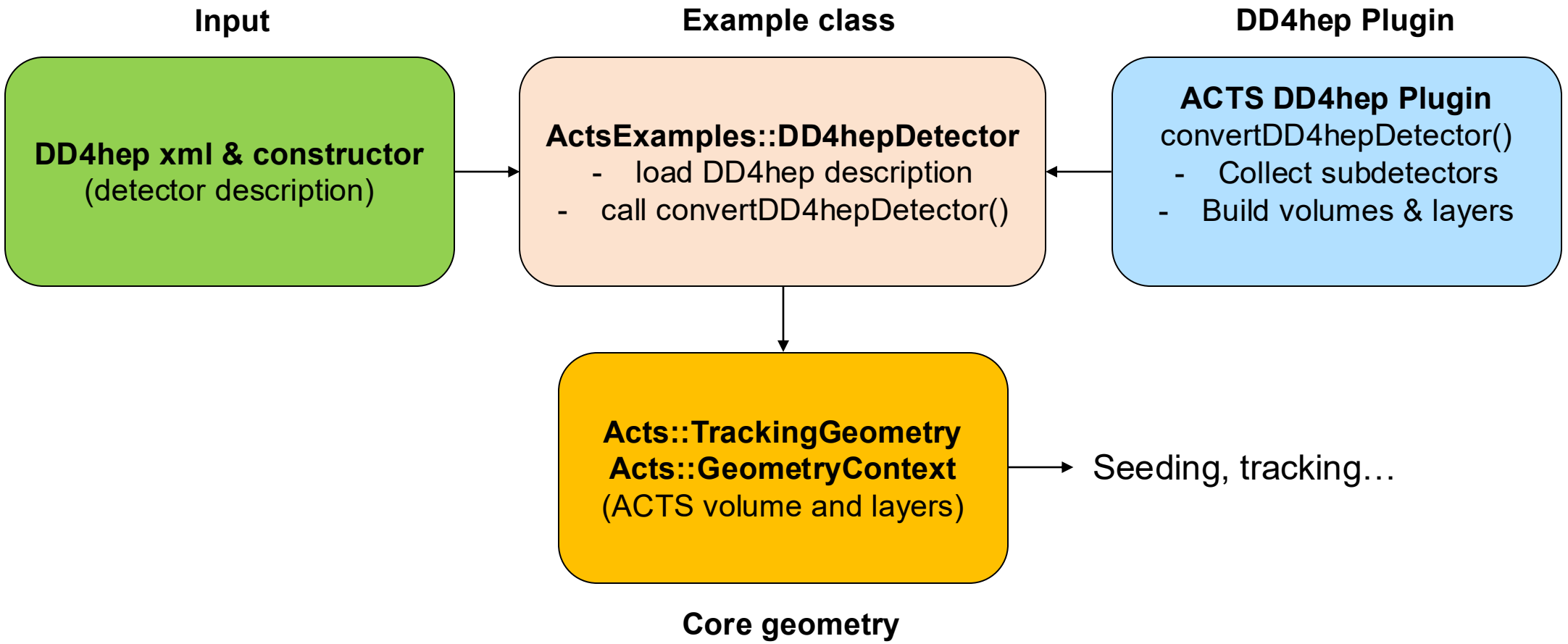
ACTS seeding & tracking with CKF/KF (performance to be validated)

Truth: same, but you already know the MC particle (Works fine)

Output as LCIO hits and track collection

Possible to extend to a complete LUXE software framework.

Geometry construction



Geometry construction

Previously:
TGeoManager → TGeoLayerBuilder → LayerArrayCreator → TrackingGeometryBuilder…

Now:

```
std::string geometryFile = argv[1];

// Configure DD4hepDetector
ActsExamples::DD4hepDetector::Config config;
config.xmlFileNames = {geometryFile};

// Build detector
ActsExamples::DD4hepDetector detector(config);

// Retrieve tracking geometry
auto trackingGeometry = detector.trackingGeometry();
if (!trackingGeometry) {
  std::cerr << "Failed to build tracking geometry from: " << geometryFile << "\n";
  return 1;
}
```

Also take cares of material & ID mapping (but interface is new)

Hacking DD4hep plugin

In cases where ACTS requires information that is not accessible or derivable from DD4hep data, it will look for additional parameters that mark up detector elements. One such example is the layer pattern. ACTS looks for detector elements which represent subdetectors, identified by `DetType::BARREL` and `ENDCAP`. To group sensors into layers correctly, it will search for the detector element corresponding to a layer **by name**. Which detector element names are interpreted as *layers* is steered by the `layer_pattern` parameter. It contains a regex that is evaluated against every detector element name encountered. It can be set from the XML using DD4hep's plugin mechanism and the provided `ParametersPlugin` like:

```
<detector id="ODD_PixelEndcapN_ID" name="PixelEndcapN"
          type="ODDPixelEndcap" readout="PixelEndcapReadout" vis="invisible">
  <type_flags type="DetType_TRACKER + DetType_ENDCAP"/>
  <!-- ... -->
</detector>
<!-- ... -->
<plugins>
  <plugin name="DD4hep_ParametersPlugin">
    <argument value="PixelEndcapN"/>
    <argument value="layer_pattern: str=PixelEndcapN\d|PixelEndplate"/>
  </plugin>
</plugins>
```

ACTS DD4hep Plugin only support barrel or compound (endcap+barrel) – a little awkward for telescope type detector.

need **type_flags** (DetType in DD4hep) to identify subdetectors.

type_flags is irrelevant to "type".

Hacking DD4hep plugin

```
<detectors>
  <detector name="Tracker" type="LUXETracker" readout="SiHits" insideTrackingVolume="true">
    <type_flags type="DetType_TRACKER + DetType_AUXILIARY"/>
```

Original version, type=LUXETracker (come with a constructor cxx file)
Has layer, stave (CarbonFiber), sensor (Si)

```
<detectors>
    <detector id="0" name="LUXETrackerEndcap" type="TrackerEndcap_o2_v06" readout="SiHits" reflect="false" >
        <type_flags type=" DetType_TRACKER + DetType_ENDCAP"/>
```

AUXILIARY

Endcap version, type=TrackerEndcap_o2_v06 (constructor can be found upstreams)
Has layer, ring, stave (CarbonFiber), module (Si)

Hacked ACTS DD4hep Plugin to support “DetType_AUXILIARY” as well (for a telescope simplification).

- No longer need “barrel” and “endcap” inside detectors
- A flat scan over “layer”, then wrap all the layers
- 1 <layer> → 1 <CylinderLayer> in ACTS, surfaces comes from sensitive vols

How-to…

Unfortunately I currently have only a local version,
Because you need:

- **latest Key4hep & DD4hep from cvmfs**
  - It has ACTs, but not the correct version, nor the right build options
- **So, your local ACTS v40**
  - But With DD4hep plugin hacked
- **MarlinTracking codes**
- **Luxgeo for simulation**

Alternatively, I could create a docker image (el9) with the “hacked” ACTS v40 installed.
(and contact the right people to maybe push the changes)

How-to…

source /cvmfs/sw.hsf.org/key4hep/setup.sh

Your local ACTS:

```
git clone --recursive https://github.com/acts-project/acts source
cd source
git checkout v40.0.0
cd ..
mkdir build
cd build
cmake -B . -S ../source -DCMAKE_BUILD_TYPE=Debug -DACTS_BUILD_FATRAS=on -
DACTS_BUILD_EXAMPLES=on -DACTS_BUILD_EXAMPLES_PYTHON_BINDINGS=on -
DACTS_BUILD_EXAMPLES_GEANT4=on -DACTS_BUILD_EXAMPLES_DD4HEP=on -
DACTS_BUILD_PLUGIN_GEANT4=ON -DACTS_BUILD_PLUGIN_TGEO=on -
DACTS_BUILD_PLUGIN_JSON=on -DACTS_BUILD_UNITTESTS=on -
DCMAKE_INSTALL_PREFIX=../install
cmake --build .
make install

export PYTHONPATH="[where/ACTs/install]/python:${PYTHONPATH}"
```

TGeo, UNITTESTS &
FATRAS are not necessary.

How-to…

Then, in MarlinTracking:

git clone git@github.com:RainWindWang/MarlinACTSTracking.git
git checkout -b porting_v40 origin/porting_v40 (or master branch for ACTs v13)
source [where/ACTs/install]/bin/this_acts.sh
mkdir build
cd build
export CMAKE_PREFIX_PATH=[where/ACTs/install]/:$CMAKE_PREFIX_PATH
cmake .. -DCMAKE_BUILD_TYPE=Debug -DCMAKE_INSTALL_PREFIX=../install
make install
cd ..
export MARLIN_DLL=${MARLIN_DLL}${PWD}'/install/lib/libMarlinACTSTracking.so:'

Then you're free to play around.

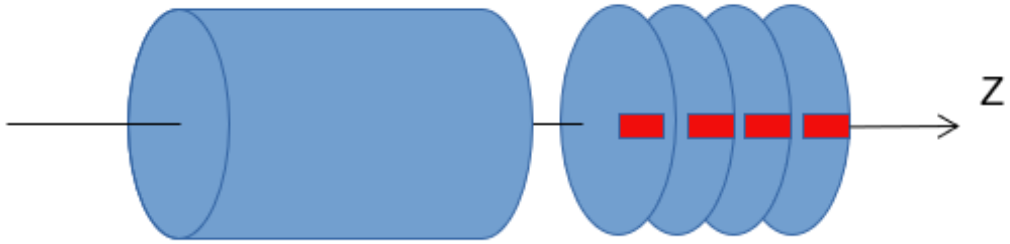
OR:

Everything is ready in /scripts/setup.sh once docker image is running (adding ACTS into it…)

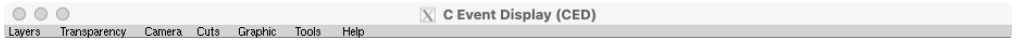
Back up

LUXE geometry and simulation

- Geometry
  - 4 “endcap” disk layer
    - each has 2 layers
      - each has 9 modules (1 overlap)
  - Parameters defined in LUXETrackerAsEndcap.xml, translated into LUXETrackerAsEndcap.root through TGeo in ACTS v13

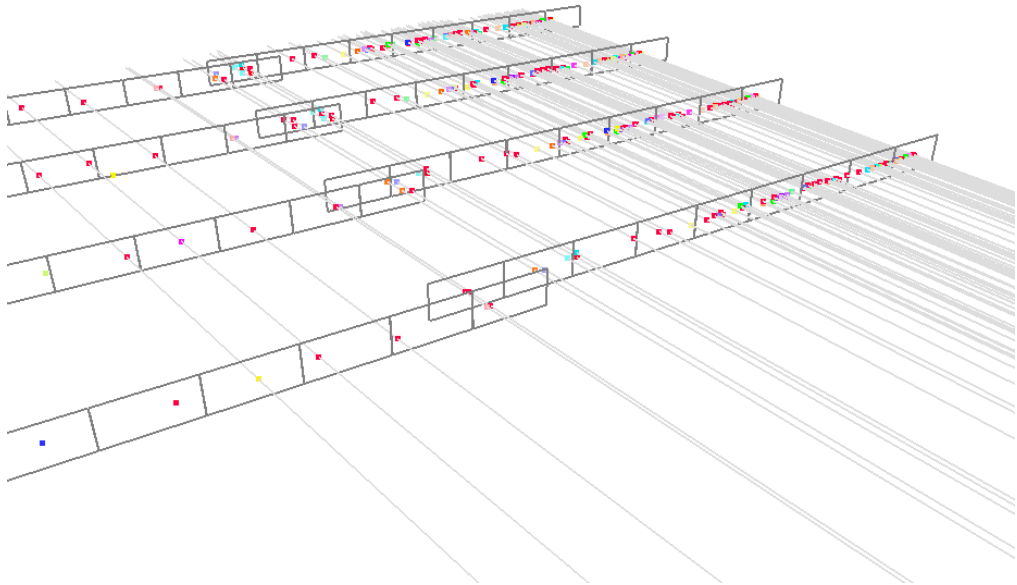


(↑ from Michal’s report)



- Simulation
  - (Irrelevant to the ACTS updates)

```
ddsim --compactFile LUXETracker.xml
--numberOfEvents 1
--enableGun
--gun.multiplicity 100
--gun.particle e+
--outputFile positrons_100_sim.slcio
```



Geometry building procedure (old)

ACTSTrackingProc.cxx
ACTSProcBase.cxx

trackinggeometry() is used to find **surface** in the main tracking code.
Then, **surface** (containing **geometryID**) is used to:

- Convert LCIO hits to ACTS hits
- Create sourcelinks between measurement and hit
- Get track parameters once seeds are found
- (hits & tracks back to LCIO collection)

---

trackingGeometry() is build through **buildDetector()** in the base code.

trackingGeometryBuilder::Config tgConfig → cylinderGeometryBuilder

- materialDecorator (_matFile.json)
- trackingVolumeBuilders
  - CylinderVolumeBuilder ← layerBuilder ← Acts::TGeoLayerBuilder ← **TGeoManager**

TGeoMenager reads from _tgeoFile (ROOT).

Geometry building procedure (new)

ACTSTrackingProc.cxx
ACTSProcBase.cxx

trackinggeometry() is used to find **surface** in the main tracking code.
Then, **surface** (containing **geometryID**) is used to:

- Convert LCIO hits to ACTS hits
- Create sourcelinks between measurement and hit
- Get track parameters once seeds are found
- (hits & tracks back to LCIO collection)

---

trackingGeometry() is build through **buildDetector()** in the base code.

trackingGeometryBuilder::Config tgConfig → cylinderGeometryBuilder

- materialDecorator (_matFile.json)
- trackingVolumeBuilders
  - CylinderVolumeBuilder ← layerBuilder ← Acts::TGeoLayerBuilder **← DD4hep Detector**

dd4hep::Detector& dd4hepDetector = dd4hep::Detector::getInstance();

- DD4hep description (xml)
- Detector constructor (cxx)

_trackingGeometry = geometryService->trackingGeometry(); **← DD4hepPlugin**

Surface visualization

```
// visualization global geometry
Acts::ObjVisualization3D geoVis;

Acts::ViewConfig sConfig;
sConfig.triangulate = true;
sConfig.color = {200, 200, 200}; // grey

trackingGeometry()->visitSurfaces([&](const Acts::Surface* surface) {
  if (surface) {
    Acts::GeometryView3D::drawSurface(
        geoVis,
        *surface,
        geometryContext(),
        Acts::Transform3::Identity(),
        sConfig
    );
  }
});
```

Traverse all ACTS surfaces

trackingGeometry & geometryContext built by cylinderGeometryBuilder

Saved an .obj file

- Cleaning up the codes to separate the geometry translation part
- visualize space points → still tiny. It's not a event display though, might not be necessary.