iLCSoft and TPC Reconstruction

BINP - CERN meeting

André Sailer & Plácido Fernández Declara

May 18, 2020

CERN

Table of Contents

TPC Geometry

Simulation

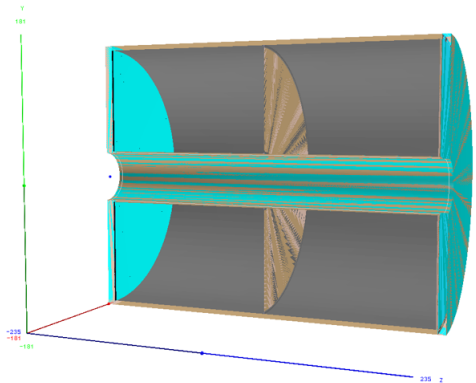
Integration into Aurora

- Integration of Geometry and Simulation
- Marlin Wrapper
- Event Data Model

TPC Geometry

TPC Geometry

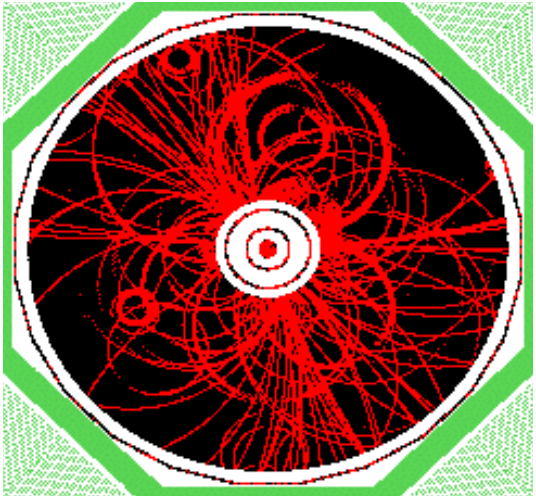
- DD4hep based TPC Driver in lcgeo
  - `https://github.com/iLCSoft/lcgeo/blob/master/detector/tracker/TPC10_geo.cpp`
- Describes
  - Inner and outer field cage at R_{min} and R_{max}, Cathode at $Z = 0$, end-plates at $Z = \pm Z_{\text{max}}$, gas volume split into layers
  - Some parameters taken from global constants
  - Field cage and cathode configuration example: `https://github.com/iLCSoft/lcgeo/blob/master/ILD/compact/ILD_common_v02/tpc10_01.xml`



Simulation

Simulation

- Sensitive Detector For TPC: TPCSDAction
  - `https://github.com/iLCSoft/lcgeo/blob/master/plugins/TPCSDAction.cpp`
- For general simulation studies: gaseous volume separated into layers to force hit at volume boundaries
- For occupancy studies limit step length or similar approach to simulate energy deposits



Integration into Aurora

Integration of Geometry and Simulation

- iLCSoft uses ddsim python program to configure and run Geant4 simulation for DD4hep
- Plugins to output LCIO or EDM4hep format
- How is the simulation done in Aurora?

```
from DDSim.DD4hepSimulation import DD4hepSimulation
from SystemOfUnits import mm, GeV, MeV, keV
SIM = DD4hepSimulation()
SIM.compactFile = "CLIC_o3_v06.xml"
SIM.runType = "batch"
SIM.numberOfEvents = 2
SIM.inputFile = "electrons.HEPEvt"
SIM.part.minimalKineticEnergy = 1*MeV
SIM.filter.filters ['edep3kev'] =
dict (name="EnergyDepositMinimumCut/3keV" ,
      parameter={"Cut" : 3.0*keV} )
```

```
$ ddsim
--action.calo                --filter.tracker              --part.keepAllParticles
--action.mapActions          -G                            --part.minimalKineticEnergy
--action.tracker             --gun.direction               --part.printEndTracking
--compactFile                --gun.energy                  --part.printStartTracking
--crossingAngleBoost         --gun.isotrop                 --part.saveProcesses
--dump                       --gun.multiplicity            --physics.decays
--dumpParameter              --gun.particle                --physics.list
--dumpSteeringFile           --gun.position                --physicsList
--enableDetailedShowerMode   -h                            --physics.rangecut
--enableGun                  --help                        --printLevel
--field.delta_chord          -I                            --random.file
--field.delta_intersection   --inputFiles                  --random.luxury
--field.delta_one_step       -M                            --random.replace_gRandom
--field.eps_max              --macroFile                   --random.seed
--field.eps_min              -N                            --random.type
--field.equation             --numberOfEvents              --runType
--field.largest_step         -O                            -S
--field.min_chord_step       --outputFile                  --skipNEvents
--field.stepper              --output.inputStage           --steeringFile
--filter.calo                --output.kernel               -v
--filter.filters             --output.part                 --vertexOffset
--filter.mapDetFilter        --output.random               --vertexSigma
```

GMP Wrapper

- The GMP Wrapper project aims to smoothly bring Marlin functionality to Gaudi framework.
  - First step is creating interfaces (wraps) around Marlin *Processors* using Gaudi *Algorithms*.
  - This keeps the current base source code working while using Gaudi framework.
  - Then different pieces of Marlin Processors can be ported progressively to replace functionality.
- Gaudi is used by other HEP experiments successfully.
- This process allows for cleanup, modernization and optimization while keeping the functionality.

GMP Wrapper dependencies

GMP Wrapper can be built against an iLCSoft installation + Gaudi. Main dependencies:

- **Gaudi**: to wrap Marlin processors and run the algorithms.
- **Marlin**: to run the underlying processors
- **LCIO**: Event Data Model input/output

Other dependencies:

- **ROOT**
- **Boost**

GMP Wrapper GitHub: `https://github.com/andresailer/GMP` (To be integrated in key4hep: `https://github.com/key4hep/`)

GMP Wrapper configuration and running

Configuring and running the wrapper is done as in Gaudi, through a Python file:

- An algorithm list is filled with wrapped Marlin Processors.
- Processors parameters are defined for each instance, defining the Marlin processor to load and list of parameters and values
  - Converter for Marlin XML configuration files exists

On algorithm initialization of a Marlin Processor, `MARLIN_DLL` environment variable is used to load the necessary libraries.

```
digiVxd = MarlinProcessorWrapper("VXDBarrelDigitiser")
digiVxd.OutputLevel = DEBUG
digiVxd.ProcessorType = "DDPlanarDigiProcessor"
digiVxd.Parameters = [
    "SubDetectorName", "Vertex", END_TAG,
    "IsStrip", "false", END_TAG,
    "ResolutionU", "0.003","0.003","0.003","0.003","0.003","0.003",END_TAG,
    "ResolutionV", "0.003","0.003","0.003","0.003","0.003","0.003",END_TAG,
    "SimTrackHitCollectionName", "VertexBarrelCollection", END_TAG,
    "SimTrkHitRelCollection", "VXDTrackerHitRelations", END_TAG,
    "TrackerHitCollectionName", "VXDTrackerHits", END_TAG,
    "Verbosity" , "DEBUG", END_TAG,]
algList.append(digiVxd)
```

Event Data Model

- iLCSoft reconstruction uses LCIO event data model (EDM) at the moment
- Short term: need to convert at some point to or from LCIO EDM
  - Simulate and write output in LCIO and use that as input for simulation, convert tracks and hits to EDM4hep (or SCT EDM)
  - Simulate to EDM4hep and convert to LCIO, convert Clupatra output back to EDM4hep
- Medium term: change processors to use EDM4hep directly
- Some LCIOConverters already exist:
  `https://github.com/ihep-sft-group/K4LCIOReader`

Thank you