Status and prospects of IRT 2.0 code adaptation to ePIC software stack

Alexander Kiselev

ePIC Collaboration Meeting, Frascati, 01/24/2025

Porting strategy

- Make use of a simplest possible pfRICH-like Cherenkov detector model rather than dealing with either pfRICH or dRICH from the very beginning
  - A large gas volume, aerogel layer, acrylic filter, large sensitive plane (single “sensor”)
- Code this geometry in a standalone pfRICH software suite, to start with, and make sure IRT 2.0 reconstruction makes sense
- Mimic this pseudo-detector (codename QRICH) in ePIC software environment
  - *epic*: dd4hep geometry description and optics file generation
  - *EICrecon*: a separate QRICH.cc plugin linked against IRT 2.0 library
  - *reconstruction_benchmarks*: something along the lines of a standalone evaluation script
- Once this works and produces results similar to the standalone case, add the required level of complexity matching ePIC pfRICH and dRICH description
  - Mirrors, sectors, segmented photosensors, etc

IRT 2.0 core & calibration logic [in a standalone mode]

- A typical R&D phase code: as much as possible happens on the fly, behind the scenes, and does not require much of book-keeping when changing e.g. a mirror or photosensor configuration
  - Optical setup encoded during GEANT geometry creation and becomes a part of the same ROOT file which contains the simulated data
  - Event tree consists of relatively complicated C++ class instance memory dumps (serialized and de-serialized by ROOT, therefore identical in GEANT simulation and event reconstruction, without any data model), with a full history of every optical photon propagation and relationships between radiators, charged particles and photons
- Un-detected photons (those which do not pass the QE normalization cut) are extensively used for any on-the-fly calibrations:
  - File-level: expected average emission point (Z-vtx) and effective average refractive index per radiator for *detected* optical photons as a function of η
  - Track-level: track parameters when crossing a particular radiator material (B-field bending), determination of “blackout zones” polluted by photons from parasitic sources, etc

Similar to IRT 1.0, these ingredients need to be emulated for use in ePIC

Input emulation

- Optical setup
  - Presently created in QRICH_geo.cpp (epic) and imported in QRICH.cc (EICrecon)
    - Bypasses the geometry service & requires *epic* code dependency on IRT libraries
    - Greatly simplifies the debugging and possible future extensions (IMHO)
    - This way, dRICH and pfRICH EICrecon codes can become virtually identical apart from QE parameterization stuff and such (a factory importing hits and an optical black box description)
  - *Chances are Chris's logic of populating volume attributes in the geometry description in DRICH_geo.cpp and parsing them back in DRICH.cc plugin when creating an optical configuration in the EICrecon code startup is flexible & robust enough to handle required multi-path optical configurations; let's put this topic on hold for now*
- Event tree (tracks and photons)
  - Will see; cumbersome, but in principle should be similar to IRT 1.0
- Calibrations
  - File-level calibrations will have to be created separately and imported as extra input files
  - Track parameterization to be provided by ACTS (cut'n'paste from DRICH sources)

Current status

- Standalone QRICH code is a no-brainer
  - Exists and is uploaded to https://github.com/eic/pfRICH/tree/main/simple
- QRICH_geo.cpp also exists and can be uploaded in a separate irt2.0 *epic* branch
  - Both dd4hep geometry and optical configuration
- EICrecon part work has just started though
  - Can be uploaded to github as a separate branch nonetheless

Was a pain to re-start this integration exercise one more time; will try to have at least a cut-down version working by the end of January

- pfRICH and dRICH optics configurations
  - Should be a relatively straightforward incremental addition of features and debugging …
  - … unless an optical ROOT file import in EICrecon is absolutely excluded

Other considerations

- Digitization code / data model may require adjustments
  - Duplication, one-to-many relationship, etc

- EDM4eic PID output implementation requires a separate discussion:
  - Should be event-level rather than track-level
  - For pfRICH may *not* want to disentangle imaging and timing information
  - May require inclusion of some proto-data rather than final PID quantities (allow variable efficiency cuts, subset of tracks and / or pdg hypotheses, etc)
  - Better combine information of all PID detectors (and e/π input usable for calorimetry?)

[Has probably all been discussed during Chandra’s talk already]

- Should probably be defined by the algorithm and user code implementation(s), later?