pfRICH Tutorial

pfRICH Tutorial

This is an introductory tutorial for the EPIC pfRICH software. Given the significant implementation overlap with the dRICH, many parts of this tutorial are applicable to both of these RICHes.

Some code relevant for the pfRICH is not yet merged. Until then, the combination of git branches we will use for today's tutorial is:

```
drich-dev: pfrich-support
     epic: pfrich-tutorial
  EDM4eic: irt-data-model
      irt: main
  juggler: 73-add-rich-irt-algorithm
```

N.B.: we will be migrating away from `juggler` relatively soon, to `EICrecon`; the reconstruction is performed by `irt`, which is independent of reconstruction framework, but currently framework bindings are only implemented in `juggler`.

The drich-dev repository contains detailed documentation how to run the EPIC software for the RICH detectors. The following tutorial is meant to be a quick-start guide.

Setup and Building

First of all, be in an `eic-shell` container (see software tutorials, first session).

Clone EPIC repositories. Note that these commands also `checkout` the branches specific to today's tutorial. If you are following this guide from a later date, these branches may no longer exist or have been merged to `main`; in that case, do not set `--branch`.

```
git clone git@github.com:eic/drich-dev.git --branch pfrich-support
cd drich-dev
git clone git@github.com:eic/epic.git --branch pfrich-tutorial
git clone git@github.com:eic/irt.git
git clone git@github.com:eic/EDM4eic.git --branch irt-data-model
git clone https://eicweb.phy.anl.gov/EIC/juggler.git --branch 73-add-rich-irt-algorithm
```

These git URLs are for SSH; if you do not have SSH, use the HTTPS URLs instead.

See what combination of branches you have:

```
./check_branches.sh
```

Set environment:

```
source environ.sh
```

Some important environment vars:

```
  DRICH_DEV        = path to drich-dev
  BUILD_NPROC      = number of parallel threads for building (might not auto-detect correctly for all machines)
  EIC_SHELL_PREFIX = primary installation directory for our builds
  DETECTOR_PATH    = detector geometry installation (XML files)
```

Customize these environment variables as needed, in particular `$EIC_SHELL_PREFIX`.

Build, using one of the following options:

```
rebuild_all.sh    # this will build everything, in order of dependence
build.sh REPO     # this will only build the repository REPO
cmake             # build it yourself, however you want
rebuild_all.sh clean  # rebuild everything from a clean slate
```

N.B.: Assuming this is the first time you are building, run `source environ.sh` a second time so that your detector build is used. Check that `DETECTOR_PATH` and any other environment variables involving the detector are set correctly.

Finally, build the `drich-dev` local code:

```
make
```

Geometry

Relevant files:

```
# compact files
epic/compact/pfrich.xml
epic/compact/drich.xml
epic/compact/definitions.xml    # global constants (positions, envelopes,
```

```
etc.)
epic/compact/optical_materials.xml  # material property tables
epic/compact/materials.xml          # general materials
epic/templates/epic.xml.jinja2      # template for generating top-level
compact files, run by 'cmake';
                                    # these are used by simulation and
reconstruction;

# geometry plugins
epic/src/PFRICH_geo.cpp
epic/src/DRICH_geo.cpp
```

One way to see the geometry is by `jsroot`:

```
run_dd_web_display.sh
run_dd_web_display.sh -p
run_dd_web_display.sh -d
run_dd_web_display.sh -e
```

Open resulting `geo/detector_geometry.root` in `jsroot`, either from the [CERN hosted version]() or from a self-hosted server.

Unfortunately, an event display is not supported on all machines (including the author's); this is a well known issue.

If you want to see the beam pipe, enable it in `epic/configurations/pfrich_only.yml` (similarly for the dRICH); these configuration files control the rendered XML files from the main `jinja` template.

Dump the constants:

```
npdet_info dump $DETECTOR_PATH/epic.xml   # dump everything, and the
derivations
npdet_info dump $DETECTOR_PATH/epic.xml | grep -Ei '^pfrich'
```

Simulation

The primary tool for running DD4hep simulations is `ddsim`. Cherenkov physics requires a modified version of this, called `npsim`; this is found in the NPDet repository, and installed in `eic-shell`.

In `drich-dev`, we provide yet another wrapper: `simulate.py`:

```
simulate.py   # usage guide
simulate.py -t1 -d-1 -s -n 50   # throw 50 pions at the pfRICH
simulate.py -t1 -d1 -s -n 50    # throw 50 pions at the dRICH
```

Several tests are available, you are welcome to add your own.

HEPMC input files are also supported: `simulate.py -i [hepmc_file]`.

Let's open the output file:

```
root out/sim.root --web=off
new TBrowser
events->Draw("PFRICHHits.position.y:PFRICHHits.position.x")
```

Draw the hits, or the segmentation, using `drich-dev` executables:

```
bin/draw_hits
bin/draw_segmentation
```

About the TTree format in the output file: this is a `PODIO` tree. Yes it can be used as a `TTree`, but `PODIO` grants us much more power. See `src/example_podio_reader.cpp` for a demonstration how to read the data using the generated classes from the data model. Run this example as:

```
bin/example_podio_reader
```

The hits for the dRICH and pfRICH are of type `edm4hep::SimTrackerHit`. The truth particles are of type `edm4hep::MCParticle`. In some cases, we use the EIC-extended data model, in the `edm4eic` namespace, such as `edm4eic::CherenkovParticleID` for the PID. For more information, see:

- EDM4hep
- EDM4eic

Within each, a YAML file defines the full data model; this YAML file is used to generate an implementation, allowing for a language-independent specification. To see the generated C++ implementation:

```
ls /opt/software/linux*/gcc*/edm4hep*/include/edm4hep/
vim /opt/software/linux*/gcc*/edm4hep*/include/edm4hep/SimTrackerHit.h
ls $EIC_SHELL_PREFIX/include/edm4eic/
vim $EIC_SHELL_PREFIX/include/edm4eic/CherenkovParticleID.h
```

Reconstruction

Create the IRT aux file; this produces `libIRT` objects from the DD4hep geometry, using the `PFRICH_RECON_*` and `DRICH_RECON_*` constants as a way to carry over the most important geometry parameters. This conversion is handled by code in `src/irtgeo`, with the hope that this will be easily portable to any framework. To make the aux file:

```
bin/create_irt_auxfile
```

Relevant code for reconstruction:

```
irt/{include,src}                                # the IRT code itself
juggler/JugPID/src/components/IRTAlgorithm*    # binds IRT to Juggler and the simulation data
juggler/JugPID/tests/options/*.py                # configures IRTAlgorithm; also has quantum efficiency tables
```

Run the reconstruction, using `juggler`:

```
recon.sh            # for usage guide
recon.sh -p -j -t   # dry run
recon.sh -p -j      # real run
```

Check the output:

```
root out/rec.root --web=off
events->Draw("PFRICHPID_1.radiator:PFRICHPID_1.theta>>h(100,0,0.5,2,0,2)","","lego")
```

An example analysis

Run a momentum scan:

- run `simulate.py`, using the momentum scan tests
- run `recon.sh`, to run Juggler on the simulation output
- draw the plots
- the automation script is in `ruby`, requiring some extra depenencies:

```
scripts/install_ruby.sh   # build a local copy of ruby
source environ.sh
bundle install   # install gems (dependencies)
```

To run the momentum scan:

```
scripts/momentum_scan.rb
```

This script runs `simulate.py` followed by `recon.sh`, for a variety of particles and momenta.

See the output in the `out/momentum_scan*/` directory, in particular:

```
ls out/momentum_scan.pfrich/_*.png
```

Contributing

We use standard Github workflow everywhere (see software tutorials, first session).

- Join the EIC organization
- Join the EPIC devs team
- Open issues and pull requests
- For the dRICH, we have:
  - Project page
  - Mattermost Channel