iLCSoft TPC Geometry and reconstruction for Aurora

Task 3 status: CERN

Plácido Fernández Declara, André Sailer

September 28, 2020

CERN

TPC related tools in iLCSoft

- TPC geometry

  https://github.com/iLCSoft/lcgeo/blob/master/detector/tracker/TPC10_geo.cpp

- Geant4 sensitive detector

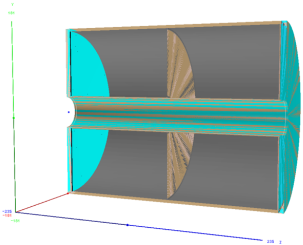
  https://github.com/iLCSoft/lcgeo/blob/master/plugins/TPCSDAction.cpp

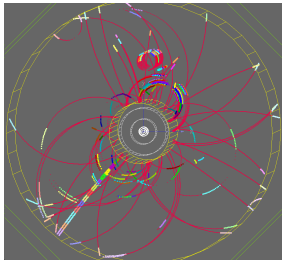
- Digitisation: parametrised resolutions

  https://github.com/iLCSoft/MarlinTrkProcessors/blob/master/source/Digitisers/src/DDTPCDigiProcessor.cc

- Pattern recognition and track reconstruction

  https://github.com/iLCSoft/Clupatra





Geometry detector file (I)

The different pieces build on each other.

- The gas volume is separated into cylinder surfaces, to force *Geant4* to make a step.
  - This in turn produces an energy deposit and hit
- The digitiser gets the hits produced by the simulation, and the geometry information that is contained in the TPC driver
- Reconstruction uses the surfaces that the TPC driver defines

To run simulation and reconstruction for SCTAU using *lcgeo* and *iLCSoft*, add an XML based on the *lcgeo TPC driver* to the SCT detector.

Geometry detector file (II)

Creating the geometry file:

- Based on `sctau_detector_geoinitialize.xml`[1]
- Missing materials were added in `material_mixture.xml`
- Needed constants were added from a mix of sources:
  - `lcgeo/ILD/compact/ILD_common_v01/basic_defs.xml`
  - `lcgeo/ILD/compact/ILD_common_v01/top_defs_common_v01.xml`
  - `lcgeo/ILD/compact/ILD_common_v01/envelope_defs.xml`
  - `lcgeo/ILD/compact/ILD_common_v02/top_defs_ILD_l5_v02.xml`
  - `DD4hep/DDDetectors/compact/detector_types.xml`

[1] `https://git.inp.nsk.su/sctau/aurora/-/blob/master/DetectorDescription/DetBase/xml/sctau_detector_geoinitialize.xml`

Geometry detector file (III)

- All detectors are added:
  - TPC is replaced by

```
<detectors>
  <include ref=".../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/BeamPipeGeo/beamPipeGeom_def.xml"/>
  <include ref=".../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/FARICHGeo/sctau_FarichPID.xml"/>
  <include ref=".../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/BarrelCrystalCaloGeo/BarrelCrystalCalo.xml"/>
  <include ref=".../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/EndcapCrystalCaloGeo/EndcapCrystalCalo.xml"/>
  <include ref=".../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/CoilGeo/CoilGeom_def.xml"/>
  <include ref=".../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/MuonSystemGeo/MuonSystem_def.xml"/>
  <include ref=".../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/DriftChamberGeo/DriftChamberGeom_def.xml"/>
  <include ref="my_tpc10_01.xml"/>
</detectors>
```

TPC (I)

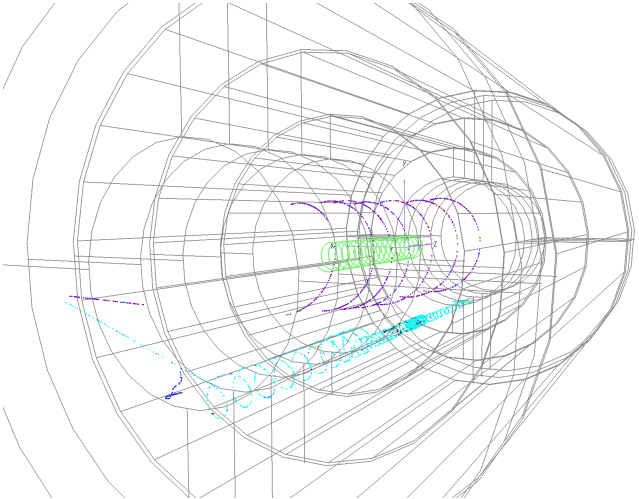
A TPC geometry file is generated [2]:

- It defines the readouts and limits
- The TPC detector is described
  - Dimensions are adapted to match SCTAU's TPC
  - The necessary constants are filled for the *innerWall*, the *outerWall* and the *readout*
- Different things need to be adapted:
  - The composition and thickness of the *innerWall* and *outerWall* is still defined as in ILD

[2] `https://git.inp.nsk.su/plfernan/geom_tpc_aurora`

TPC (II)

- Hits can be visualized by exporting in SLCIO format
- ddsim --compactFile my_sctau_det_geo_2.xml -N 10 -G --outputFile=hits.slcio --part.userParticleHandler='' --gun.isotrop=true --gun.energy "100*MeV" --steeringFile=steering.py



lcgeo in Proxima cluster

- lcgeo [3] can be built to be used with `ddsim`.
  - It requires DD4hep with DDG4 plugin
  - LCIO, CLHEP and BOOST are the other dependencies to build everything
- CVMFS could be used to run and compile the different parts
- Spack now contains recipes for this stack of software
  - It compiles from scratch everything; only needs a compiler
  - Compilation against existing packages can be done, against CVMFS for example

[3] `https://github.com/iLCSoft/lcgeo`

Compile and use *lcgeo*

lcgeo for *ddsim* can be compiled with this script[4]

- LCIO and CLHEP are compiled
- LCIO is exported to `$LD_LIBRARY_PATH`
- BOOST is downloaded (can be picked from existing installation)
- DD4hep and plugins are compiled against the LCIO installation
- DD4hep is exported to `$LD_LIBRARY_PATH`, and `thisdd4hep.sh` sourced
- Existing DD4hep is removed from `$LD_LIBRARY_PATH`
- LCGEO is the compiled against the LCIO and DD4hep installations
- Finally LCGEO is exported to `$LD_LIBRARY_PATH` to be used by *ddsim*

[4] `https://git.inp.nsk.su/plfernan/lcgeo_build`

GMP Wrapper - Reconstruction

- The Gaudi-Marlin-Processors Wrapper project brings *Marlin* functionality to *Gaudi* framework, smoothly.
- It creates interfaces *(wraps)* around Marlin Processors, encapsulating them in Gaudi Algorithms.
- Current Marlin source code is kept intact, and it is just called on demand from the Gaudi Framework.

	Marlin	Gaudi
Language	C++	C++
Working unit	Processor	Algorithm
Config. language	XML	Python
Set-up function	init	initialize
Working function	process	execute
Wrap-up function	end	finalize
Transient Data Format	LCIO	anything

GMP Wrapper now

- Bugs were fixed, a manual (`README.md`) was included with instructions to compile, configure, run and test.
- Updated and modernization of the code base.
- Running examples are included as tests.
- A recipe to build it with Spack is also part of the *k4-spack* repo.
- It was included as part of Key4hep, moving there the repo[5].
- CI is now included with GitHub Actions, checking syntax (`clang-format`), and running two basic functionality tests.

[5] `https://github.com/key4hep/`

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
  - It will eventually disappear when only Gaudi Algorithms are used
- **LCIO**: Event Data Model input/output
  - Can be changed for a different one, i.e. EDM4hep

Other dependencies:

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

Or simply[6]:

- `spack install key4hep-stack`

[6] `https://key4hep.github.io/key4hep-doc/spack-build-instructions/README.html`

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.

GMP configuration example

```
MyTPCDigiProcessor = MarlinProcessorWrapper("MyTPCDigiProcessor")
MyTPCDigiProcessor.OutputLevel = INFO
MyTPCDigiProcessor.ProcessorType = "DDTPCDigiProcessor"
MyTPCDigiProcessor.Parameters = [
                                "DiffusionCoeffRPhi", "0.025", END_TAG,
                                "DiffusionCoeffZ", "0.08", END_TAG,
                                "DoubleHitResolutionRPhi", "2", END_TAG,
                                "DoubleHitResolutionZ", "5", END_TAG,
                                "HitSortingBinningRPhi", "2", END_TAG,
                                "HitSortingBinningZ", "5", END_TAG,
                                "MaxClusterSizeForMerge", "3", END_TAG,
                                "N_eff", "22", END_TAG,
                                # ...
                                ]
algList.append(MyTPCDigiProcessor)
```

Testing

Added testing with `ctest`:

- Simple test that runs some Marlin Processors: AidaProcessor -> InitDD4hep -> VXDBarrelDigitiser
- `muon.slcio` is used for input, without hits.
- Second test generates an input file with `ddsim`
- It runs a similar list of algorithms with actual hits
- Output checks for regex with `INFO Application Manager Terminated successfully`

```
ddsim \
    --steeringFile $ILCSOFT/ClicPerformance/HEAD/clicConfig/clic_steer.py \
    --inputFiles $ILCSOFT/ClicPerformance/HEAD/Tests/yyxyev_000.stdhep -N 4 \
    --compactFile $ILCSOFT/lcgeo/HEAD/CLIC/compact/CLIC_o3_v14/CLIC_o3_v14.xml \
    --outputFile $GMP_tests_DIR/inputFiles/testSimulation.slcio
```

SCTAU reconstruction

GMP Wrapper successfully computes the full CLIC reconstruction:

- The provided converter can translate to Python Gaudi steering file
- Algorithms for digitisers, reconstruction, pattern recognition, etc can be included into this sequence
- The converter add all algorithms to the list, and leaves the configurable ones commented
- It uses LCIO for the moment, but this can be adapted and will be changed in the future

Future directions

- We can simulate the geometry and export hits in SLCIO or other formats
- We would then run digitisers, pattern recognition and reconstruction
  - The Marlin file for reconstruction can be converted with the GMP Wrapper script
  - Adapters for Event Data Model
- Move from LCIO to EDM4HEP.
  - Converter available in K4LCIOReader [7]
- Replace wrapped Marlin Processors by actual Gaudi Algorithms.
- How to approach the transition?

[7] https://github.com/ihep-sft-group/K4LCIOReader/blob/master/src/K4LCIOConverter.cc