Detector simulations, part 2

Dataset creation for full simulation

Instead of using standalone GEANT simulations, the HEP experiments often create their own applications and simulation frameworks. The full GEANT simulation is used, but via a customized interface, different for each experiment as the code is written from scratch by the physicists participating the experiment. Therefore an average user does not have to worry about e.g. a correct detector geometry description, as the geometry is modelled by experts and included in the simulation framework.

CMS is no exception, it has its own programs written for end users. The first full simulation program CMSIM was written in Fortran, and it used Fortran based GEANT3 for simulating the particle-matter interactions. Later CMSIM was replaced with OO software ORCA, OSCAR and CMSSW, and GEANT3 was switched to GEANT4. CMSSW uses a OO framework, any Fortran code still living is hidden by C++ interfaces to the Fortran programs. We use CMS here as an example to get the general idea, how the full simulation works in practice.

CMSSW

The dataset production for CMSSW simulations can be done in steps. The steps for a full simulation study are

- Event generation
- Hits
- Digis
- High level object reconstruction
- Analysis

Event generation is done using the Monte Carlo event generators available. The most common event generators are included in the program as libraries, and the user can select the event generator, and what kind of events are to be simulated. The output is a root file containing the data collections user has selected.

Detector simulations, part 2

Framework

The CMSSW framework implements a software model wherein there is one executable, called cmsRun, and many plug-in modules which run algorithms. The same executable is used for both detector simulation and for creating the Monte Carlo data.

The CMSSW executable, cmsRun, is configured at run time by the user's job-specific configuration file. This file tells cmsRun which data to use, which modules to run, which parameter settings to use for each module, and in what order to run the modules. Required modules are dynamically loaded at the beginning of the job. The configuration file language is Python.

```
$ cmsRun myPythonConfigFile.py
```

The CMS Event Data Model (EDM) is centered around the concept of an Event as a C++ object container for all RAW and reconstructed data pertaining to a physics event. During processing, data are passed from one module to the next via the Event, and are accessed only through the Event. All objects in the Event may be individually or collectively stored in ROOT files, and are thus directly browsable in ROOT.

The CMS code can be found at GitHub
`https://github.com/cms-sw/cmssw`

The reference manual can be found at
`https://cmssdt.cern.ch/SDT/doxygen/`

Environment setup

On kale.grid, the environment needed for CMSSW is set by

```
setenv VO_CMS_SW_DIR /cvmfs/cms.cern.ch
source $VO_CMS_SW_DIR/cmsset_default.csh
setenv SCRAM_ARCH slc7_amd64_gcc630
```

Detector simulations, part 2

Dataset creation

To create a new dataset one first needs to create a working area. This is done with an auxiliary program called SCRAM (Source Configuration, Release, And Management). With command "scram list" you can see all the software and versions available via scram.

```
$ scram project CMSSW CMSSW_8_0_30
$ cd CMSSW_8_0_30/src/
```

set the runtime environment:

```
$ cmsenv
```

The source code (Gen-fragment) is in the git repo,

```
git cms-addpkg Configuration/Generator
```

Compile:

```
$ scram b -j 16
```

Next, using cmsDriver.py, create a configuration file for the generation of the events in which Higgs particle decays to two Z^0 bosons which in turn decay to four leptons:

```
cmsDriver.py Configuration/Generator/python/ \
H200ZZ4L_13TeV_TuneCUETP8M1_cfi.py \
-s GEN --conditions auto:run2_mc_GRun --datatier \
GEN-SIM --eventcontent RAWSIM --no_exec
```

This generates the events. For the detector simulation, change '-s GEN' to '-s GEN,SIM'.

The generated config file is the input for running CMSSW (cmsRun). The number of events to be simulated is controlled with the line

```
process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32(1))
```

Pile-up is selected by including a pile-up config file, in this case no pile-up

```
process.load(``SimGeneral.MixingModule.mixNoPU_cfi'')
```

The output file name is selected with PoolOutputModule. In PoolOutputModule one can also select the saved collections, e.g. if one wants only the generator level information, use drop:

```
process.FEVT = cms.OutPutModule(``PoolOutputModule'',
  cms.PSet(outputCommands=cms.untracked.vstring(
  ''drop *'',''keep *_generator_*_*''))
  fileName = cms.untracked.string(...)
```

Detector simulations, part 2

The wildcard * in the drop and keep commands replace only whole strings, not substrings

”keep *_TriggerResults_*_*”, Allowed
”keep *_Trigger*_*_*”, Won’t work.

Let’s next browse the root file content. Open the root file and start the browser

```
$ root H200ZZ4L_13TeV_...._GEN.root
root [1] new TBrowser
```

double click ROOT Files
double click H200ZZ4L_13TeV_..._GEN.root
double click Events;1

Now you have all the collections listed. The collection names are formed from namespace+Class name _ label _ instance _ process name

Example: For MC info the namespace is edm, class name HepMCProduct, label generator, no instance and produced with process GEN, the name of the process in the example thus making edmHepMCProduct_generator__GEN

How is the data then accessed? One needs to write an analysis module (class), by inheriting edm::EDAnalyzer. In the analysis code the MC generator level information in edmHepMCProduct_generator__GEN is accessed as follows:
The class name was edm::HepMCProduct, and label ”generator”.

```
Handle<edm::HepMCProduct> mcEventHandle;
iEvent.getByLabel(”generator”,mcEventHandle);
if(mcEventHandle.isValid()){
    const HepMC::GenEvent* mcEvent =
        mcEventHandle->GetEvent();
    HepMC::GenEvent::particle_const_iterator i;
    for(i = mcEvent->particles_begin();...
    }
}
```

Let’s next modify the example Gen-fragment to produce SM Higgs bosons with one of the important SM Higgs production processes, the Vector Boson Fusion (VBF). The Higgs boson decays

Detector simulations, part 2

mainly to $\mathrm{b\bar{b}}$ pairs, but the b jet final state is difficult to extract from the QCD multijet background. The next most probable final state is $\mathrm{H} \to \tau\tau$, which is experimentally easier. The PYTHIA manual tells the correct PYTHIA switches for these kind of events, section Process selection – Higgs. The vector boson fusion is already switched on in the example datacard (processes HiggsSM:ff2Hff(t:ZZ) and HiggsSM:ff2Hff(t:WW)) so let's keep them and comment out the loop mediated process. The decays should now include only decays H→ $\tau\tau$. Since the SM Higgs KF code is 25 and tau KF code is 15, we change the config as follows

```
'25:onMode = off',

'25:onIfMatch = 15 15',
```

The Higgs decay final states can be listed by running stand-alone PYTHIA calling pythia.particleData.list(), as described in Lecture 8.

The example Gen-fragment looks now like this:

```
generator = cms.EDFilter(“Pythia8GeneratorFilter”,
  PythiaParameters = cms.PSet(
    parameterSets = cms.vstring(
      'pythia8CommonSettings',
      'pythia8CUEP8M1Settings',
      'processParameters'),
    processParameters = cms.vstring(
      'HiggsSM:ff2Hff(t:WW) = on',
      'HiggsSM:ff2Hff(t:ZZ) = on ',
      '25:m0 = 200',
      '25:onMode = off',
      '25:onIfMatch = 15 15'),
```

The process parameters define the hard interaction, but the definitions for the underlying event are needed, too. They are added by pythia8CommonSettings and (for the tune CUEP8M1, there are also other tunes available) pythia8CUEP8M1Settings.

Detector simulations, part 2

The CMSSW configuration files for event generation with different event generators are available in the git repository. There is also a link from the MCM page (passwd protected) to each dataset and Gen-fragment

`https://cms-pdmv.cern.ch/mcm`

This allows one to make a semi-private production: private event generation with official datacards, and it provides working examples which can serve as a starting point for creating new Gen-fragments.

It is also possible to create LHE files separately, with stand-alone tools, and then use those LHE files to feed official CMS dataset production. If you need to do this, contact your physics analysis group or physics object group generator contact person for further instructions.

Analysis example

For CMSSW analysis one needs data, and an analysis program. Let's assume we have a dataset available containing electrons as in the dataset generation example. The analysis program (module) needs to be written. Since we are now interested in electrons, let's start from an Electron Analysis example, found under RecoEgamma/Examples

```
$ scram p CMSSW CMSSW_8_0_30
$ cd CMSSW_8_0_30/src
$ cmsenv
$ git cms-addpkg RecoEgamma/Examples
$ cd RecoEgamma/Examples
$ scram b -j 16
```

We select the example GsfElectronMCAnalyzer. The source code is located in plugins/. First let's try to run the analyzer without modifications.

```
$ cd test
```

Try grepping to find the config file for module

Detector simulations, part 2

name GsfElectronMCAnalyzer. The file is GsfElectronsMCAnalyzer_cfg.py. Edit the file to contain the correct input (assuming the produced root file is moved to $PWD),

```
process.source = cms.source("PoolSource",
 fileNames = cms.untracked.vstring (
   "file:H200ZZ4L_13TeV_TuneCUETP8M1_ \
     cfi_py_GEN.root"
 )
)
```

The prefix 'file:' is needed when accessing local root files. If the file is read from CASTOR, prefix 'rfio:' is needed. If no prefix is specified, the default location for datasets /store.. is used.

Let's select `maxEvents = 2`,
and run the program:

```
$ cmsRun GsfElectronsMCAnalyzer_cfg.py
```

Since the input file contains only generator level information, the example is not expected to work without commenting out the reco-object code. Let's leave only the MC particle loop, and modify the code to print the MC electron pt, eta and phi. Edit file plugins/GsfElectronMCAnalyzer.cc member function GsfElectronMCAnalyzer::analyze

One can see there a loop over MC particles (GenParticleCollection is used here), where the print should be added. The reco::GenParticleCollection contains the same information as the edm::HepMCProduct (standard HepMC event record), but the former is recommended as the code will work with all data tiers, like AOD, from which the edm::HepMCProduct is dropped.

Recompile and rerun

```
$ scram b
$ cmsRun GsfElectronsMCAnalyzer_cfg.py
```

More about reco::GenParticle methods can be found in the CMSSW reference manual.