Delphes 3
Latest Developments

Michele Selvaggi
(on behalf of the Delphes collaboration)
CERN - UCL

github.com/delphes

CHEP 2016
11/10/2016
Detector simulation

- Full simulation (GEANT):
  - simulates particle-matter interaction (including showers, nuclear int., brehmstrahlung, photon conversions, etc ...) → 100 s /ev

- Experiment Fast simulation (ATLAS, CMS ...):
  - simplifies and makes faster simulation and reconstruction → 1 s /ev

- Parametric simulation (Delphes, PGS):
  - parameterize detector response, reconstruct complex objects
    B field propagation, Jets, Missing ET → 10 ms /ev

- Object smearing (Atom, Falcon, TurboSim):
  - from parton to detector object (lookup tables)
When Delphes?

- When to use FastSim?
  
  → test your model with detector simulation and you have limited resources
  → scan big parameter space (SUSY-like)
  → preliminary tests of new geometries/resolutions (future detectors)
  → educational purpose (bachelor/master thesis)

- When not to use FastSim?
  
  → very exotic topologies (HSCP, long-lived, ...) (NOT YET ...)

The Delphes project

- Delphes project started back in 2007 at UCL as a side project to allow quick phenomenological studies

- Since 2009, its development is community-based
  - ticketing system for improvement and bug-fixes
  → user proposed patches, can be forked from github and make pull-requests

- In 2013, DELPHES 3 was released (DELPHES 2 NOT SUPPORTED ANYMORE !!):
  - C++ modular software
  - Dependencies: gcc, tcl, ROOT
  - is shipped with FastJet

- Delphes is itself distributed by various tools: MadGraph, MadAnalysis, CheckMate
- Widely tested and used by the community (pheno, Snowmass, Recasting, FCC, CMS upgrades ...)

- Repository: github.com/delphes
- Original publication: JHEP 02 (2014) 057 [1307.6346]
What is Delphes?

- **Delphes** is a *modular framework* that simulates of the response of a multipurpose detector in a *parameterized* fashion.

**Includes:**
- pile-up
- charged particle propagation in magnetic field
- electromagnetic and hadronic *calorimeters*
- particle flow

**Provides:**
- leptons (electrons and muons)
- photons
- jets and missing transverse energy (particle flow)
- taus and b's
Run Delphes

- Install ROOT from root.cern.ch
- Clone Delphes from github or download from website
- Type in shell:
  ```
  ./configure
  make -j 4
  ```
- Run Delphes:
  ```
  ./DelphesSTDHEP [configuration_file] [output] [input]
  ./DelphesHepMC [configuration_file] [output] [input]
  ```
- Input formats: HepMC, StdHep, ProMC, LHE
- Output: browsable ROOT tree
New Features
Run Delphes with Pythia 8

- You can now run the full MC/reconstruction chain with one simple command by linking Delphes with Pythia8 (more info here).

- Set PYTHIA8 path variable and recompile Delphes:
  
  ```
  export PYTHIA8=[path_to_pythia8_installation]
  make HAS_PYTHIA8=true DelphesPythia8
  ```

- You can then directly either directly use Pythia8 matrix element, or use external LHE (also with matching available).

- In both case the input to Delphes will be a Pythia8 “cmd” file:
  
  ```
  ./DelphesPythia8 [detector_card] [pythia8_cmd] [output]
  ```

- Avoids storing huge intermediary event files (hepmc), all the parton/hadron-level information can accessed via the Particle branch in the output.

- If multiple weights were stored in LHE input, Delphes stores them in the Weights branch in a vector.
Particle-Flow

- Given **charged track** hitting given **calorimeter cell**:
  - are deposits more **compatible** with **charged** or **charged + neutrals** hypotheses?
  - how to assign momenta to all resulting components?

- We have **two** measurements \((E_{\text{trk}}, \sigma_{\text{trk}})\) and \((E_{\text{calo}}, \sigma_{\text{calo}})\)

- Call \(E_{\text{neutral}} = E_{\text{calo}} - E_{\text{trk}}\)

**Algorithm:**

\[
\text{If} \quad \frac{E_{\text{neutral}}}{\sqrt{\sigma_{\text{trk}}^2 + \sigma_{\text{calo}}^2}} > S:
\]

→ create **PF neutral particle** + **PF track**

Otherwise:

→ create **PF track** with (weighted) average energy \(w_i = 1/\sigma_i^2\)
Particle flow makes optimal use of Tracker and Calorimeter information
Timing

- Beamspot description with $f(z,t)$ customizable profile has been included
- Time information is propagated up calorimeters, and then smeared
- Vertexing in 3D/4D has been included

Lindsey Gray, Andrew Hart
Delphes in LHCb

- Integration of Delphes in LHCb framework has started in private branch
- Complete re-writing of ParticlePropagator module, accounting for:
  - asymmetric acceptance and non-rotational symmetry taken into account
  - account for $p_T$ kick in $B_y$ field

- A first implementation of a full chain analysis has been done
- Currently working on correct emulation of efficiency and resolution of the LHCb detector
Conclusions

- Delphes 3 has been out for two years now, with **major improvements**:
  - modularity
  - default cards giving results on par with published performance from LHC experiments
  - updated configurations for future e+e- and hh colliders
  - interfaced within MadGraph5/Py8, CheckMate/MadAnalysis

- Delphes 3 can be used right away for fast and realistic simulation for present and future collider studies
- Delphes is used both by experimentalist and theorists
- Continuous development (particle flow, vertexing, timing ...)
- Feel free to contribute!
Back Up
The modular system allows the user to configure and schedule modules via a configuration file (.tcl), add modules, change data flow, alter output information.

Modules communicate entirely via collections (vectors) of universal objects (TObjArray of Candidate four-vector like objects).

Any module can access TObjArrays produced by other modules using ImportArray method:

```
ImportArray("ModuleName/arrayName")
```
Delphes configuration file is based on tcl scripting language.

This is where the detector, data-flow, and output tree is configured.

Delphes provides tuned detector cards for some detectors:
- ATLAS, CMS, ILD, FCC (LHCb in progress).
- can find other tunes in CheckMate, MadAnalysis.

Order of execution of various modules is configured in the Execution Path:

```tcl
set ExecutionPath {
    ParticlePropagator
    TrackEfficiency
    ...
    Calorimeter
    ...
    TreeWriter
}
```
Configuration file

```
### Photon isolation
###

module PhotonIsolation {
    set CandidateInputArray PhotonEfficiency/photons
    set IsolationInputArray EFlowMerger/eflow

    set OutputArray photons
    set DeltaRMax 0.5
    set PTMin 0.5
    set PTRatioMax 0.12
}

### Electron efficiency
###

module ElectronEfficiency {
    set InputArray ElectronFilter/electrons
    set OutputArray electrons

    # set EfficiencyFormula {efficiency formula as a function of eta and pt}

    # efficiency formula for electrons
    set EfficiencyFormula {
        (pt <= 10.0) * (0.00) +
        (abs(eta) <= 1.5) * (pt > 10.0) * (0.95) +
        (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 10.0) * (0.85) +
        (abs(eta) > 2.5) * (0.00)
    }
```