The Hitchhiker's Guide to DELPHES: EIC Edition

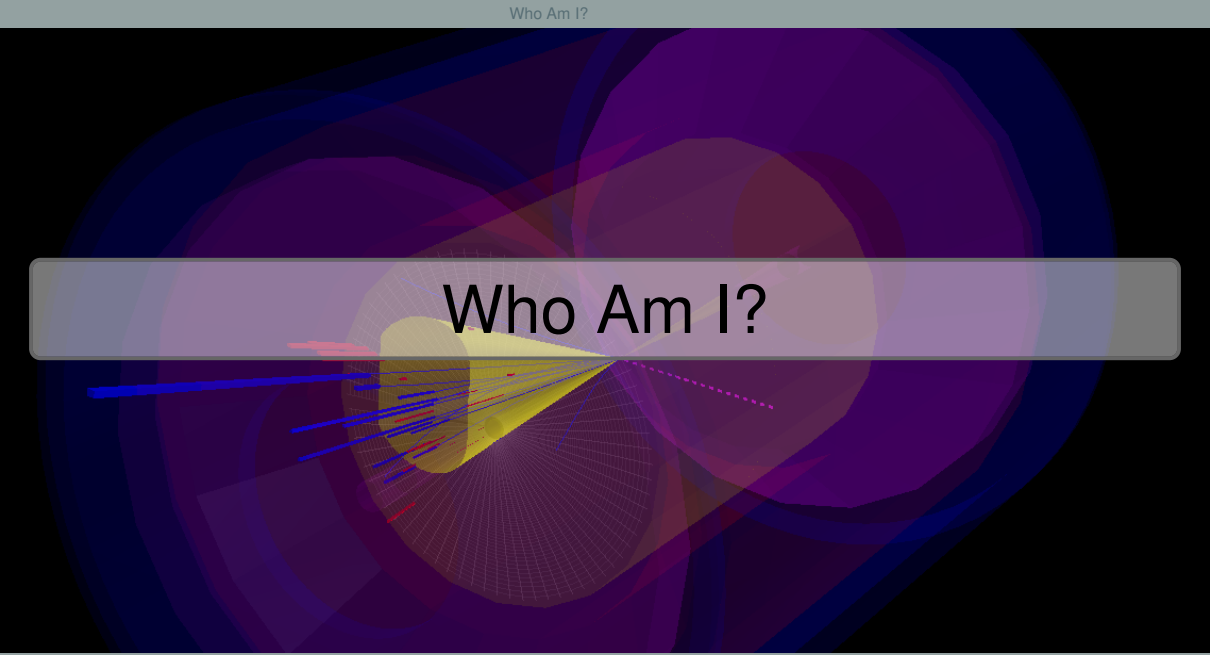
Stephen Sekula[1]

[1]Southern Methodist University
Dallas, TX, USA

July 31, 2020

SMU | Dedman College of Humanities & Sciences

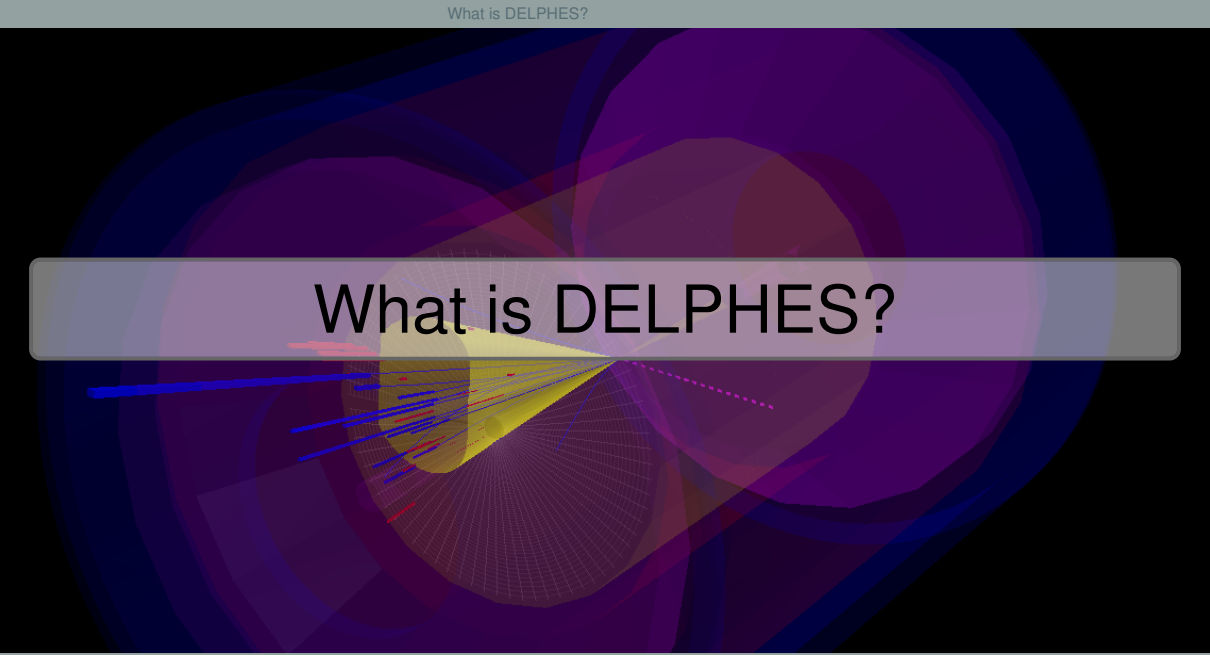
Outline



Who Am I?

Who Am I? (a brief introduction)

- I am an Associate Professor of Physics at SMU and a computational experimental physicist in practice. I am also about to become the Physics Department Chair.
- I earned my Ph.D. on the BaBar Experiment. Most notable accomplishment there was the discovery of the $\eta_b(1S)$.
- I've been active on the ATLAS Experiment since 2009. Most notable accomplishments so far have been the Run 2 b-quark-initiated jet (b-jet) triggers and the first observation of $H^0 \to b\bar{b}$.
- I am very interested in heavy flavor physics at the EIC, notably intrinsic strange and heavy flavor in the proton, neutron, etc. Generically, I am interested in heavy flavor as a tool in the EIC physics portfolio.
- Most notably: I am not an originator or developer of the DELPHES analysis framework; I am just a committed user these days.



What is DELPHES?

What is DELPHES?

A framework for fast simulation of a generic collider experiment

"Delphes [1] is a C++ framework, performing a fast multipurpose detector response simulation. The simulation includes a tracking system, embedded into a magnetic field, calorimeters and a muon system. The framework is interfaced to standard file formats (e.g. Les Houches Event File or HepMC) and outputs observables such as isolated leptons, missing transverse energy and collection of jets which can be used for dedicated analyses. The simulation of the detector response takes into account the effect of magnetic field, the granularity of the calorimeters and sub-detector resolutions. Visualisation of the final state particles is also built-in using the corresponding ROOT library."

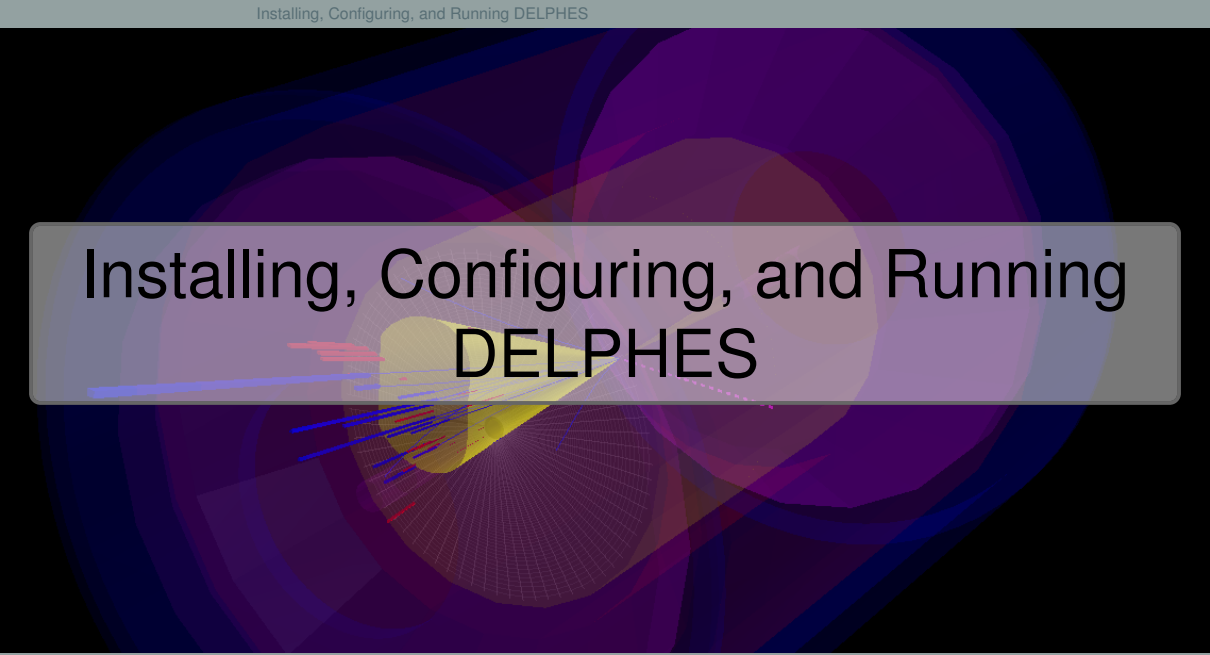
DELPHES Project Site

I first encountered DELPHES during the 2013 Snowmass process, when it was clear the community needed a collaboration-independent framework to project into the future without needing large collaboration approvals.

DELPHES: the basic ideas

- Generate a primary event and hadronization using standard tools like POWHEG, PYTHIA, etc. (choose your favorite) $\rightarrow$ export to LHEF or HepMC2 format.
- Use an appropriate DELPHES executable to read in your events as well as the detector parameterization.
- The flow of algorithms in the detector parameterization is controlled by the user, but a sensible order is usually:
  - Separate neutral and charged long-lived particles;
  - Apply a track efficiency and smearing procedure to emulate the tracker;
  - Run a calorimeter reconstruction on energy deposits, including particle flow;
  - Construct jets, missing energy, and other high-level objects;
  - Apply particle ID maps to emulate a dedicated PID system (or systems);
  - Save to disk in ROOT format.
  - Analyze and visualize.

Examples of all of this follow.



Installing, Configuring, and Running DELPHES

Installing DELPHES

Install from GitHub,
https://github.com/delphes/delphes

```
git clone https://github.com/delphes/delphes.git
# Make sure ROOT is installed before compiling
# If you want to compile with PYTHIA8.2:
# HAS_PYTHIA8=true
PYTHIA8=/path/to/pythia82/ ./configure
# Otherwise, just run:
# ./configure
make -j
make -j display
```

```
-rwxrwxr-x 1 ssekula ssekula 132041296 Jul 22 13:56 libDelphesNoFastJet.so
-rwxrwxr-x 1 ssekula ssekula 133566912 Jul 22 13:56 stdhep2pileup
-rwxrwxr-x 1 ssekula ssekula 133566928 Jul 22 13:56 root2pileup
-rwxrwxr-x 1 ssekula ssekula 133580744 Jul 22 13:56 root2lhco
-rwxrwxr-x 1 ssekula ssekula 133571152 Jul 22 13:56 pileup2root
-rwxrwxr-x 1 ssekula ssekula 134190744 Jul 22 13:56 libDelphes.so
-rwxrwxr-x 1 ssekula ssekula 133571920 Jul 22 13:56 lhco2root
-rwxrwxr-x 1 ssekula ssekula 133566904 Jul 22 13:56 hepmc2pileup
-rwxrwxr-x 1 ssekula ssekula 133571040 Jul 22 13:56 example1
-rwxrwxr-x 1 ssekula ssekula 133779792 Jul 22 13:56 DelphesValidation
-rwxrwxr-x 1 ssekula ssekula 133571184 Jul 22 13:56 DelphesSTDHEP
-rwxrwxr-x 1 ssekula ssekula 133571040 Jul 22 13:56 DelphesROOT
-rwxrwxr-x 1 ssekula ssekula 138795224 Jul 22 13:56 DelphesPythia8
-rwxrwxr-x 1 ssekula ssekula 133571184 Jul 22 13:56 DelphesLHEF
-rwxrwxr-x 1 ssekula ssekula 133571184 Jul 22 13:56 DelphesHepMC
-rwxrwxr-x 1 ssekula ssekula 133577208 Jul 22 13:56 CaloGrid
drwxrwxr-x 2 ssekula ssekula     41472 Jul 22 14:12 modules
```

This results in a number of libraries and executables being built. I will concentrate on what you do with the `DelphesXYZ` executable and the `libDelphesDisplay.so` library.

Configuring and Running DELPHES

Get the Delphes EIC Model, https://github.com/miguelignacio/delphes_EIC

```
git clone https://github.com/miguelignacio/delphes_EIC.git
ls delphes_EIC/delphes_card_EIC.tcl # Detector Configuration Card
ls delphes_EIC/CC_DIS.cmnd # Pythia8 Configuration File for CC DIS
```

For this example, I will assume we want to run a simulation of charged current deep-inelastic scattering (CC DIS) at leading order using PYTHIA8.2, then simulate a EIC-like detector response using DELPHES.
The all-in-one executable for this is `DelphesPythia8` (see appendix for important information about building Pythia 8.2 for the EIC!)

```
cd delphes/
./DelphesPythia8 ../delphes_EIC/delphes_card_EIC.tcl ../delphes_EIC/CC_DIS.cmnd
```

A Peek in the DELPHES Cards: Defining the Sequence of Algorithms

```
#####################################
# Order of execution of various modules
#####################################

set ExecutionPath {
  ParticlePropagator

  ChargedHadronTrackingEfficiency
  ElectronTrackingEfficiency
  MuonTrackingEfficiency

  ChargedHadronMomentumSmearing
  ElectronMomentumSmearing
  MuonMomentumSmearing

  TrackMerger
  TrackSmearing

  ECal
  HCal

  Calorimeter
  EFlowMerger
  EFlowFilter

  PhotonEfficiency
  PhotonIsolation

  ElectronFilter
  ElectronEfficiency
  ElectronIsolation

  ChargedHadronFilter
  MissingET

  NeutrinoFilter
  GenJetFinder
  GenMissingET

  FastJetFinder

  JetEnergyScale

  JetFlavorAssociation
  GenJetFlavorAssociation

  UniqueObjectFinder

  ScalarHT

  TrackCountingBTagging

  TreeWriter
}
```

- The first thing you do is specify the order of execution of “modules” in DELPHES.
- The first module is the `ParticlePropagator`, which is responsible for taking the ‘`stableParticles`” list from the input simulation and, for charged particles, creating a list of helical trajectories in space
- Here you can impose a solenoidal magnetic field

```
#################################
# Propagate particles in cylinder
#################################

module ParticlePropagator ParticlePropagator {
    set InputArray Delphes/stableParticles

    set OutputArray stableParticles
    set ChargedHadronOutputArray chargedHadrons
    set ElectronOutputArray electrons
    set MuonOutputArray muons


    #Values taken from EIC detector handbook v1.2
    # radius of the magnetic field coverage, in m
    set Radius 0.8
    # half-length of the magnetic field coverage, in m
    set HalfLength 1.00

    # magnetic field
    set Bz $PARAM_BZ
}
```

A Peek in the DELPHES Cards: Tracking Efficiency

```
####################################
# Common Tracking Efficiency Model
####################################

set CommonTrackingEfficiency {
    (pt <= 0.1)                                                    * (0.00) +
    (abs(eta) <= 1.5) * (pt > 0.1   && pt <= 1.0)                  * (0.97) +
    (abs(eta) <= 1.5) * (pt > 1.0)                                 * (0.99) +
    (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1   && pt <= 1.0)  * (0.96) +
    (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 1.0)               * (0.98) +
    (abs(eta) > 2.5 && abs(eta) <= 3.5) * (pt > 0.1   && pt <= 1.0)  * (0.95) +
    (abs(eta) > 2.5 && abs(eta) <= 3.5) * (pt > 1.0)               * (0.97) +
    (abs(eta) > 3.5)                                               * (0.00)
}


####################################
# Charged hadron tracking efficiency
####################################

module Efficiency ChargedHadronTrackingEfficiency {
  set InputArray ParticlePropagator/chargedHadrons
  set OutputArray chargedHadrons

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

  # tracking efficiency formula for charged hadrons
  set EfficiencyFormula $CommonTrackingEfficiency
}
```

- For each class of charged particle - hadrons, electrons, and muons - you can now impose a tracking efficiency model based on `pt` and `eta`.
- You define a formula string and pass it to a clone of the `Efficiency` module
- We've implemented a baseline EIC detector model, consistent with the handbook but improvising some, based on past experiments, to establish reasonable levels of efficiency (which the handbook doesn't over-specify, of course)
- I won't show them here, but there are also stages for tracking momentum resolution smearing and impact parameter resolution smearing. You again just define the formula.

A Peek in the DELPHES Cards: Calorimeter Layout and Performance

```
##############
#   ECAL
##############

module SimpleCalorimeter ECal {
  set ParticleInputArray ParticlePropagator/stableParticles
  set TrackInputArray TrackSmearing/tracks

  set TowerOutputArray ecalTowers
  set EFlowTrackOutputArray eflowTracks
  set EFlowTowerOutputArray eflowPhotons

  set IsEcal true
  set EnergyMin 0.1
  set EnergySignificanceMin 1.0

  set SmearTowerCenter true

  set pi [expr {acos(-1)}]

  # lists of the edges of each tower in eta and phi
  # each list starts with the lower edge of the first tower
  # the list ends with the higher edged of the last tower

  # Granularity is not discussed in EIC detector handbook.


  ##BARREL
 # assume 0.0174 x 0.020 resolution in phi,eta in the barrel |eta| < 1.0
  set PhiBins {}
  for {set i -180} {$i <= 180} {incr i} {
    add PhiBins [expr {$i * $pi/180.0}]
  }

 #deta=0.02 units for |eta| <=1.0
  for {set i -50} {$i < 50} {incr i} {
        set eta [expr {$i * 0.02}]
        add EtaPhiBins $eta $PhiBins
  }
```

- The `SimpleCalorimeter` module allows you to layout, in ϕ and η and longitudinal segmentation, a calorimeter system.
- ECal example is to the left, showing just the barrel layout. This is aligned with the EIC detector matrix/handbook.
- You can then use formulas to specify how particles partition their energy in this system.
- You can also specify an energy resolution formula, just like the tracking efficiency example but with η and energy.

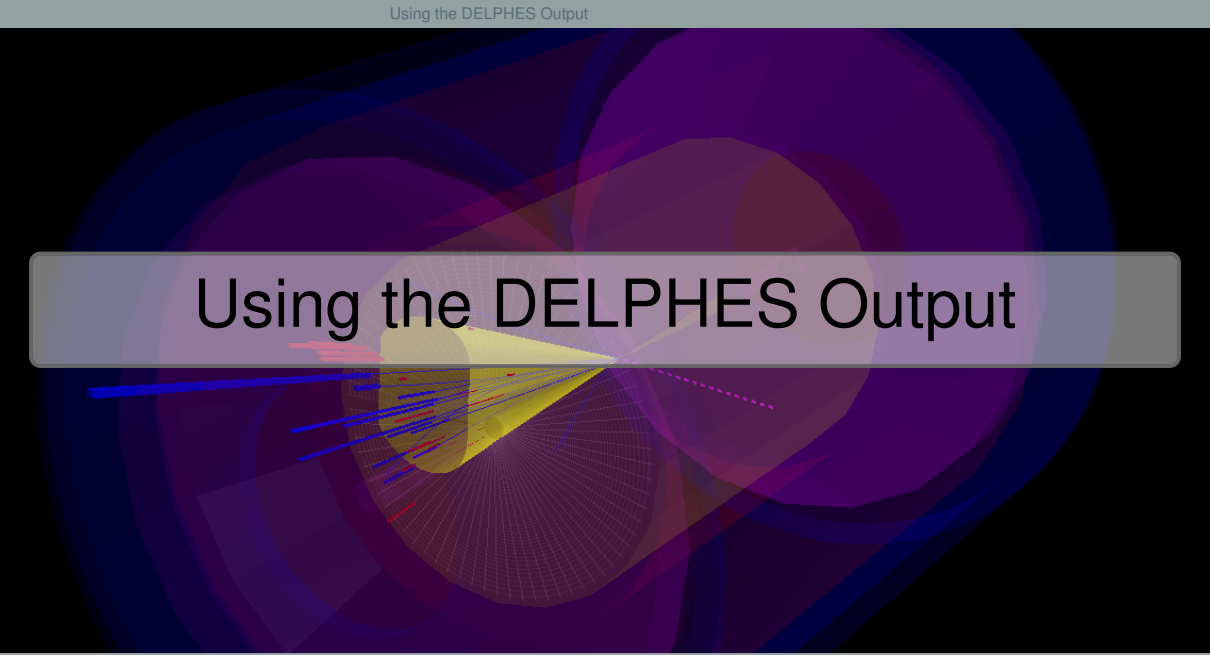
Energy Partitioning and Resolution

```
# energy fractions for e, gamma and pi0
add EnergyFraction {11} {1.0}
add EnergyFraction {22} {1.0}
add EnergyFraction {111} {1.0}
set ResolutionFormula {(eta <= -2.0 && eta>-4.0) * sqrt(energy
    ^2*0.01^2 + energy*0.02^2)+ \ #...
```

A Peek in the DELPHES Cards: Particle ID Maps

```
# mRICH

module IdentificationMap mRICHPID {
  set InputArray HCal/eflowTracks
  set OutputArray tracks

  # {PID in} {PID out} {formula}
  # make sure "PID in" and "PID out" have the same charge (e.g {-13} {211} or {-321} {211})
  # {211} {-13} is equivalent to {-211} {13} (and needs to be written once only...)



 # From EIC Yellow Report PID studies, the mRICH coverage is -3.5<eta<-1 and 1<eta<2


  # --- pions ---

  add EfficiencyFormula {-11} {-11} { (eta<-3.5 || (-1 < eta && eta < 1) || eta>2.0)*( 0.00 ) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (0.00 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 0.50) * (1.00) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (0.50 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 1.90) * (1.00) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (1.90 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.00) * (0.99) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.00 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.10) * (0.98) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.10 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.20) * (0.96) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.20 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.30) * (0.92) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.30 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.40) * (0.87) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.40 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.50) * (0.82) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.50 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.60) * (0.77) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.60 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.70) * (0.73) +
                                      ((-3.5 <= eta && eta <= -1.0) || (1.0 <= eta && eta <= 2.0)) * (2.70 <= (pt * cosh(eta)) && (pt * cosh(eta)) < 2.80) * (0.69) +
```

The `IdentificationMap` module lets you implement a PID efficiency map for specific particle types. The above except is adapted from information from the mRICH concept, and shows $e \to e$ ID efficiency vs. η and p_{T}. I wrote a Python script to auto-generate the formula from given info.



Using the DELPHES Output

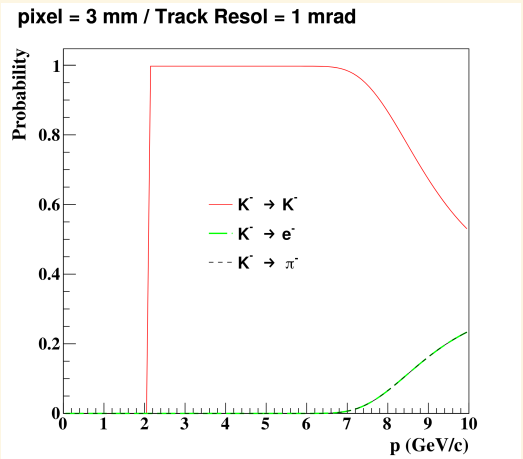
Delphes ROOT Structure

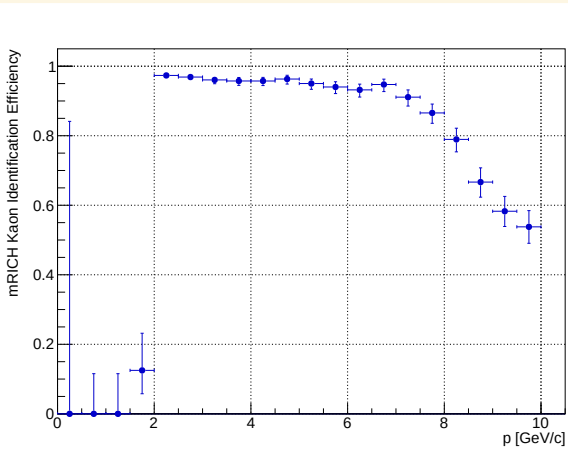
The output ROOT file contains a TTree, `Delphes`, with generator- and reconstruction-level information. The DELPHES Wiki has a Workbook with decent documentation, including a description of the tree contents.

You can load the DELPHES library to help ROOT understand the objects in the tree, or you can just use them without special treatment (e.g. to access simple features like track p_T, etc.). You can run interactively or write ROOT macros, or full compiled analysis framework code, etc. to use the output. It's very flexible and allows us to get to physics quickly.

class Track	
PID	HEP ID number
Charge	track charge
P	track momentum
PT	track transverse momentum
Eta	track pseudorapidity
Phi	track azimuthal angle
CtgTheta	track cotangent of theta
EtaOuter	track pseudorapidity at the tracker edge
PhiOuter	track azimuthal angle at the tracker edge
T	track vertex position (t component)
X	track vertex position (x component)
Y	track vertex position (y component)
Z	track vertex position (z component)
TOuter	track position (t component) at the tracker edge
XOuter	track position (x component) at the tracker edge
YOuter	track position (y component) at the tracker edge
ZOuter	track position (z component) at the tracker edge
Xd	X coordinate of point of closest approach to vertex
Yd	Y coordinate of point of closest approach to vertex
Zd	Z coordinate of point of closest approach to vertex
L	track path length
D0	track transverse impact parameter
DZ	track longitudinal impact parameter
ErrorP	track momentum error
ErrorPT	track transverse momentum error
ErrorPhi	track azimuthal angle error
ErrorCtgTheta	track cotangent of theta error
ErrorT	time measurement error
ErrorD0	track transverse impact parameter error
ErrorDZ	track longitudinal impact parameter error
Particle	reference to generated particle
VertexIndex	reference to vertex
class Tower	
ET	calorimeter tower transverse energy
Eta	calorimeter tower pseudorapidity
Phi	calorimeter tower azimuthal angle
E	calorimeter tower energy
T	ecal deposit time, averaged by sqrt(EM energy) over all particles, not s
NTimeHits	number of hits contributing to time measurement

Example: From PID performance text file to DELPHES to Projected PID Performance





Left: plot from Murad Sarsour, also provided as performance text file. Right: DELPHES $K \to K$ PID efficiency from quick analysis of output ROOT file.

Visualizing Events

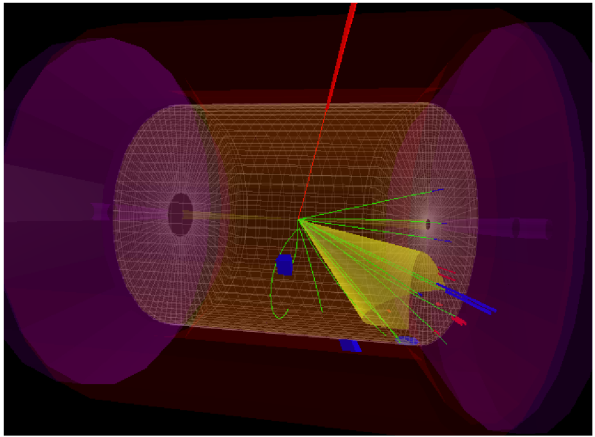
Built-In Event Display

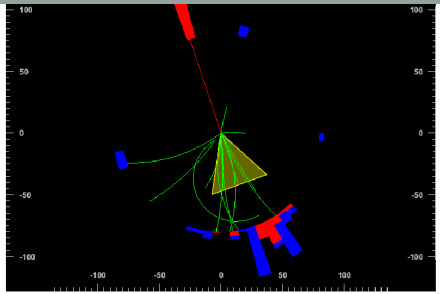
Running the Built-In Event Display

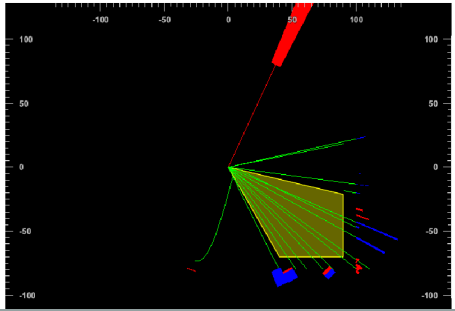
```
# Make sure the delphes/ folder is in PATH and LD_LIBRARY_PATH (or DYLD_LIBRARY_PATH)
cd delphes/
root -l ./examples/EventDisplay.C'("../delphes_EIC/delphes_card_EIC.tcl","out.root")'
```

The event displays on the following slides were provided by Miguel Arratia and showcase the power of this framework. We've used the event display already to study backgrounds and develop selection criteria in response to those, etc (e.g. light jets contaminating charm jets).

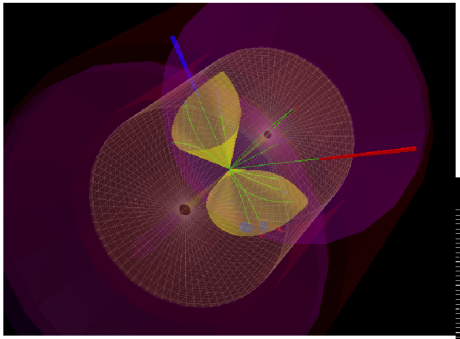
Single-jet high Q2 DIS

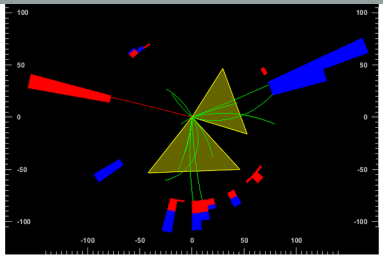


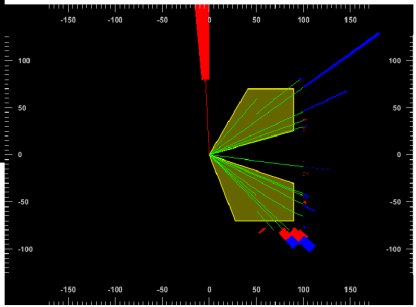




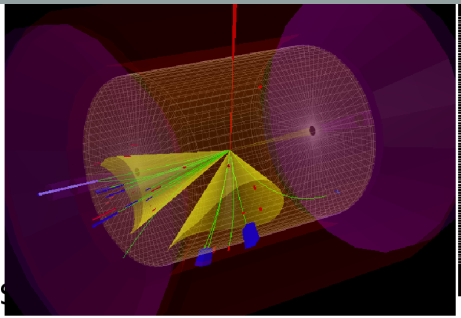
Dijet in high Q2 DIS

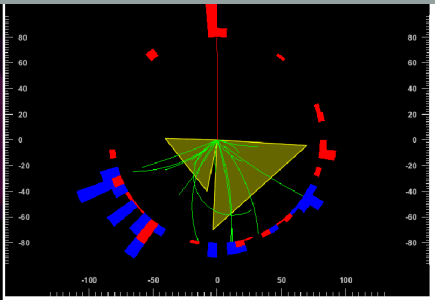


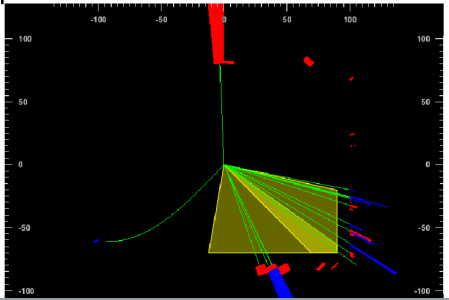


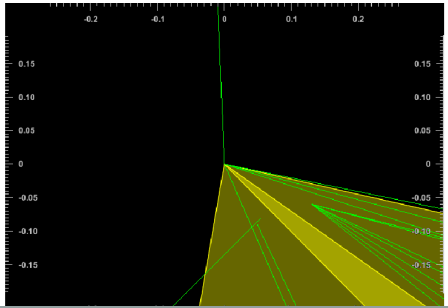


Neutral-current DIS
boson-gluon fusion
double-charm jet



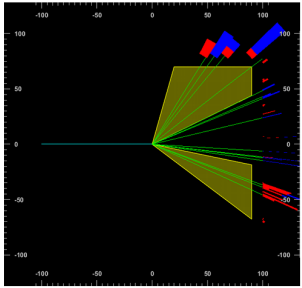


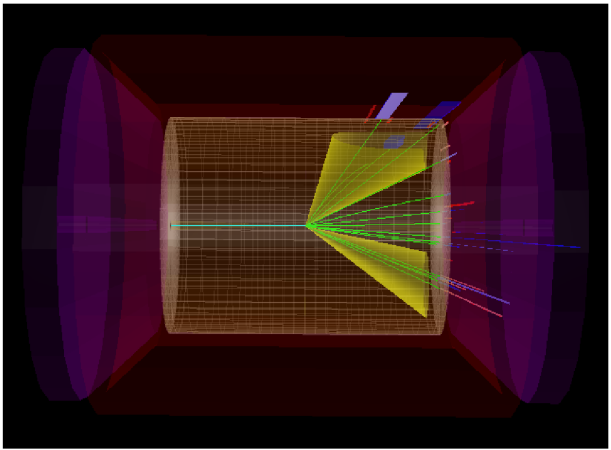


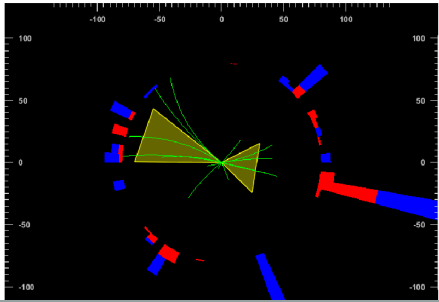


Dijet, low Q2 DIS (photoproduction) ,

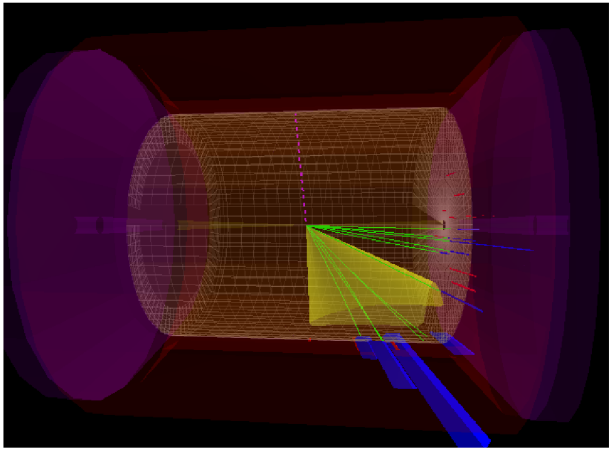
electron (cyan) gows down the beam pipe

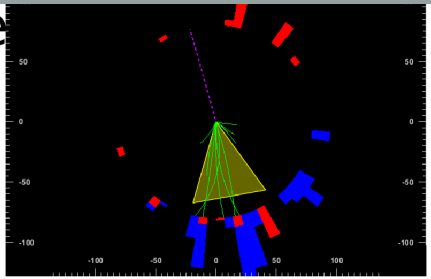


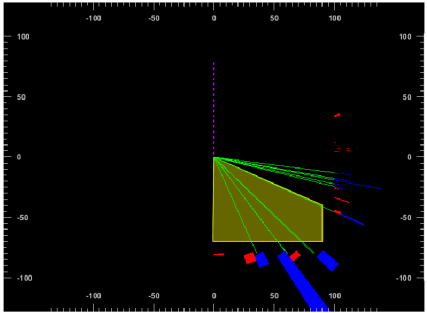




Single jet, Charged-curre

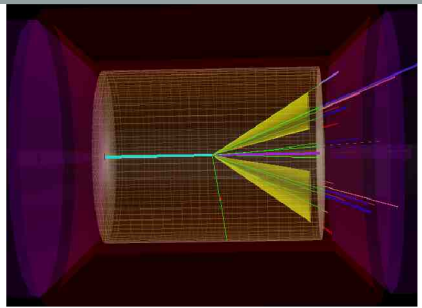


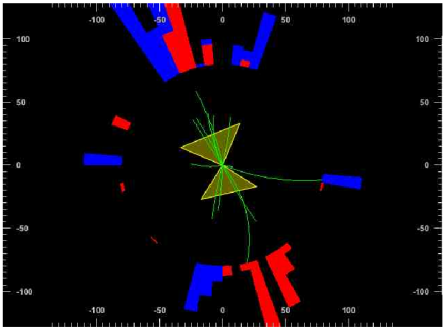


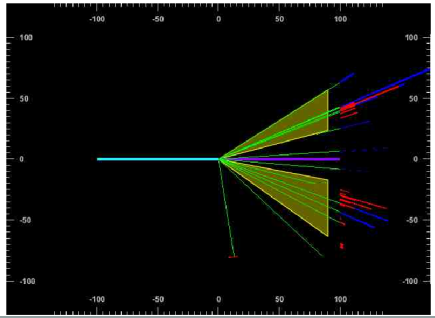


Hard diffractive photoproduction

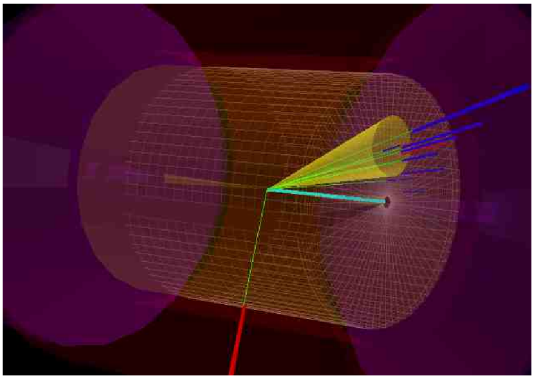
electron (cyan) proton (purple) escape down the beam pipe

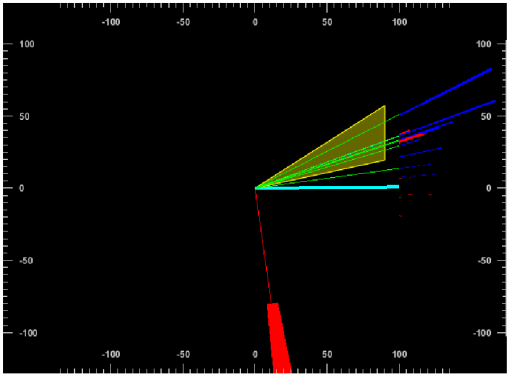






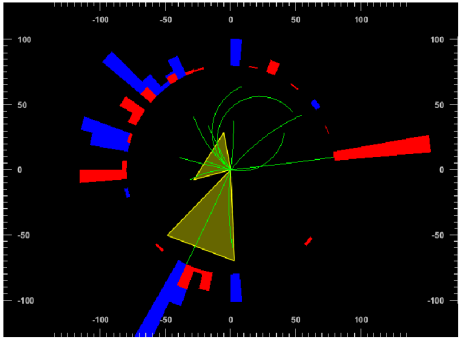
Hard diffractive DIS, (proton ~100 GeV in cyan escapes down the beam pipe)

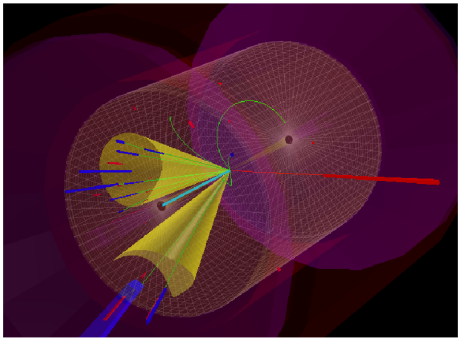




Hard diffractive DIS, high Q2.

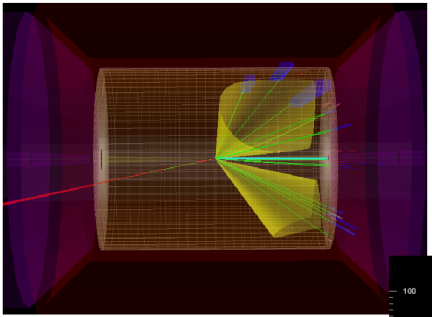
(proton in cyan escapes down the beam pipe)

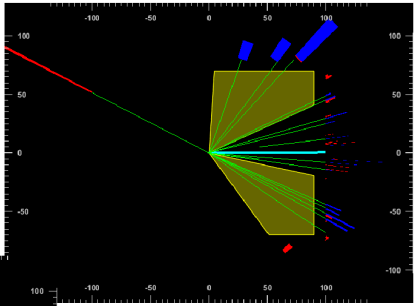


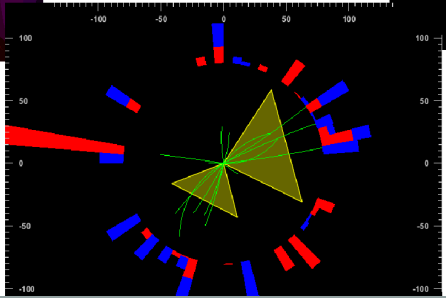


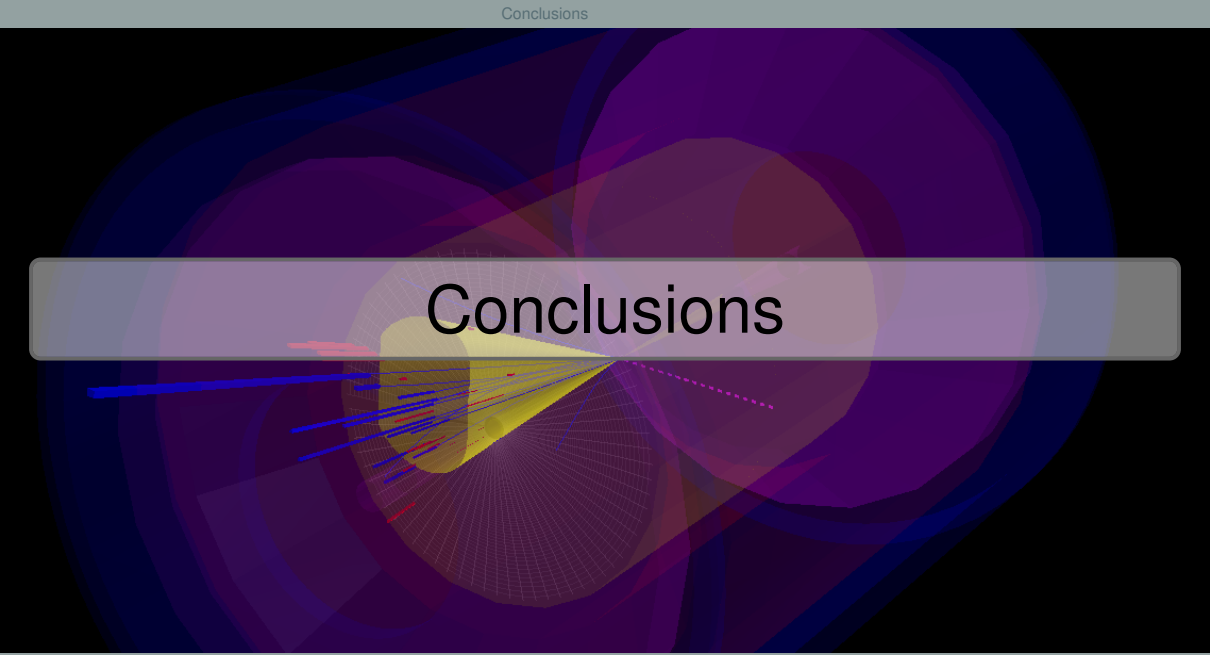
Hard diffractive DIS, low Q2,

(proton ~140 GeV in cyan escapes down the beam pipe)





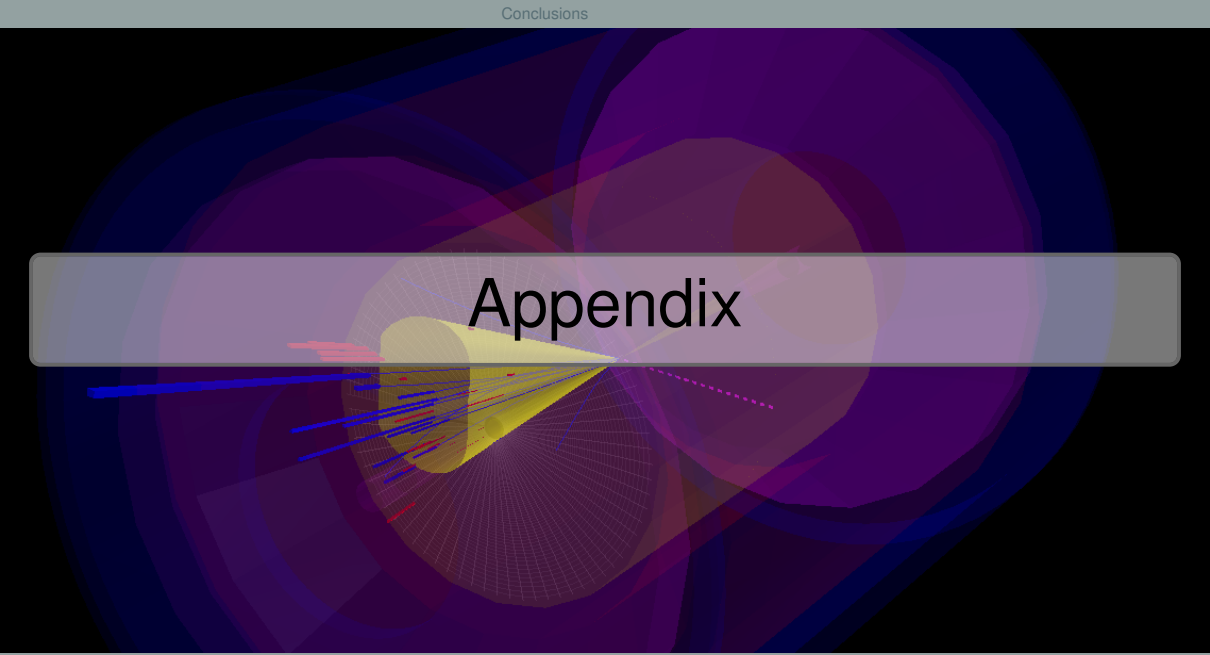




Conclusions

Conclusions

- DELPHES is an open-source framework for simulation of a generic particle detector.
  - An SMU student (Burleson) recently submitted a bug fix using the recommended GitHub pull request procedure, and it was accepted $\rightarrow$ we are happy to report they take the open-source, community development part very seriously.
- It's pretty easy to get up and running and we've already done a lot of the work of implementing a baseline EIC Detector
  - We've recently turned to trying out PID detector proposals in this framework, and you can see we're making positive progress $\rightarrow$ we're happy to study PID in the context of charm jet tagging and intrinsic proton strangeness, and the best way to facilitate that is to provide maps of PID hypothesis vs p and η, etc. for different configurations/choices.
- We've (Arratia, Furletova, Hobbs, Olness, S.S.) already generated 1 paper using this and we're planning on at least a few more (with a now larger group of charm jet enthusiasts), with a focus on material most relevant to the Yellow Report. (c.f. arXiv:2006.12520), available in Ref. [2]).



Appendix

Installing, Patching, and Compiling PYTHIA8.2

PYTHIA8.2 ships with a bug for DIS. You need to patch this to run any appreciable amount of simulation.

Installing PYTHIA8.2

```
# Download http://home.thep.lu.se/~torbjorn/pythia8/pythia8244.tgz
tar zxf pythia8244.tgz
cd pythia8244/
```

Patch PYTHIA8.2 for DIS

```
// Edit the file src/BeamRemnants.cc. Find the following line of code:
int iLepScat = isDIS ? (beamOther[0].iPos() + 2) : -1;
// Below that line of code, add this:
if (iLepScat > (event.size()-1)) {
 return false;
}
```

Then compile and use PYTHIA.

References I

[1] CP3 center of the Université catholique de Louvain, “DELPHES Fast Simulation,” 2020. https://cp3.irmp.ucl.ac.be/projects/delphes.

[2] M. Arratia, Y. Furletova, T. Hobbs, F. Olness, and S. J. Sekula, “Charm jets as a probe for strangeness at the future Electron-Ion Collider,” arXiv:2006.12520 [hep-ph].