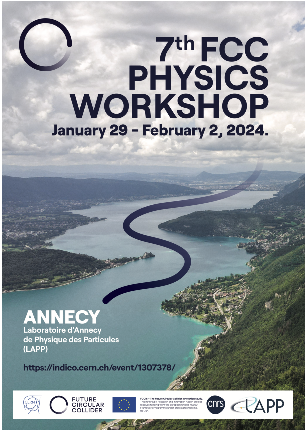


Fast simulation: status and use at FCC-ee

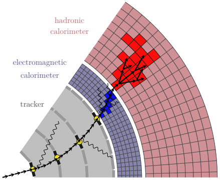
Anna Zaborowska

7th FCC Physics Workshop,
January 31st 2024

Disclaimer

I will describe only fast simulation understood as
a replacement for (certain part of) detailed (GEANT4) simulation.

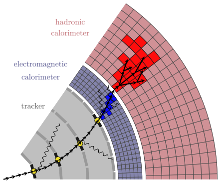
Simulation of particle passage: full vs fast



detailed / "full" simulation
→ Geant4

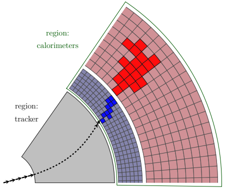
- detailed detector description (DD4hep⇒Geant4)
- definitions of particles and processes
- transport in e-m field

Simulation of particle passage: full vs fast



detailed / “full” simulation → GEANT4

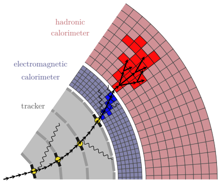
parameterisation / “fast” simulation → requires input



- detailed detector description (DD4hep⇒GEANT4)
- definitions of particles and processes
- transport in e-m field

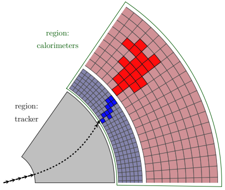
- where particles are parametrised
- which particles
- how/what happens

Simulation of particle passage: full vs fast



detailed / "full"
simulation
→ Geant4

parameterisation /
"fast" simulation
→ requires input

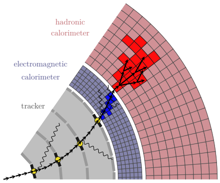


- detailed detector description (DD4hep⇒Geant4)
- definitions of particles and processes
- transport in e-m field

- where particles are parametrised
- which particles
- how/what happens

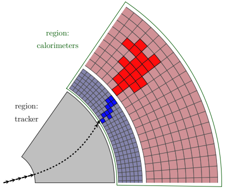
Defining both 'full' and 'fast' simulation within one framework offers great flexibility to seamlessly mix both types.

Simulation of particle passage: full vs fast



detailed / "full" simulation
→ GEANT4

parameterisation / "fast" simulation
→ requires input



- detailed detector description (DD4hep⇒GEANT4)
- definitions of particles and processes
- transport in e-m field

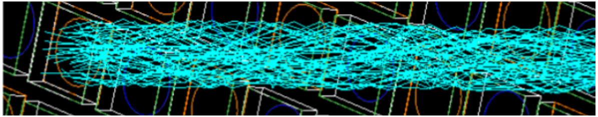
- where particles are parametrised
- which particles
- how/what happens

Defining both 'full' and 'fast' simulation within one framework offers great flexibility to seamlessly mix both types.

Given the recent decision to turn towards DDG4 as the full sim framework, the old examples from k4SimGeant4 and native Geant4 examples need to be adapted, but the main principles remain.

FCC-ee usecases

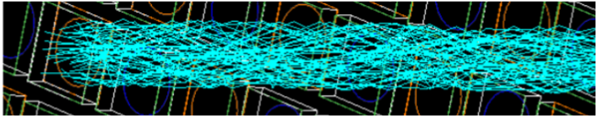
Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



Sanghyun Ko's slides
dual-readout git

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.

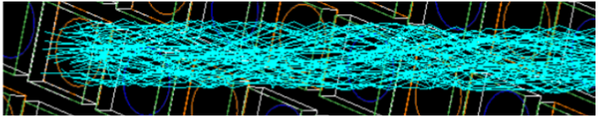


Sanghyun Ko's slides
dual-readout git

More potential use-cases that could play a role for FCC-ee simulations:

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



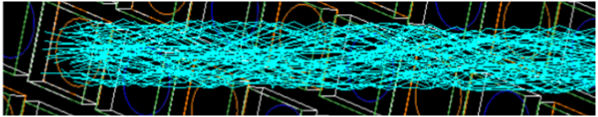
Sanghyun Ko's slides
dual-readout git

More potential use-cases that could play a role for FCC-ee simulations:

- Rough parameterisation of particles in the detectors that are not of primary importance (or not yet implemented)

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



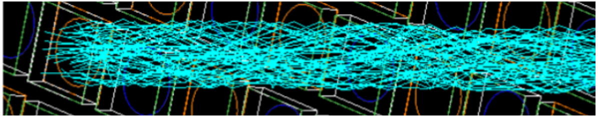
Sanghyun Ko's slides
dual-readout git

More potential use-cases that could play a role for FCC-ee simulations:

- Rough parameterisation of particles in the detectors that are not of primary importance (or not yet implemented)
- Speeding up calorimetry simulation taking advantage of active developments in the community

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



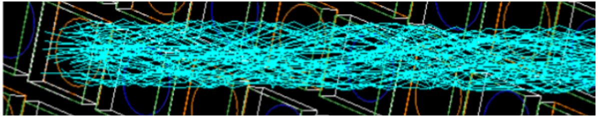
Sanghyun Ko's slides
dual-readout git

More potential use-cases that could play a role for FCC-ee simulations:

- Rough parameterisation of particles in the detectors that are not of primary importance (or not yet implemented)
- Speeding up calorimetry simulation taking advantage of active developments in the community
- ... (all other ideas that maybe you want to discuss!)

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



Sanghyun Ko's slides
dual-readout git

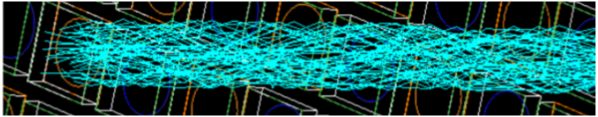
More potential use-cases that could play a role for FCC-ee simulations:

- Rough parameterisation of particles in the detectors that are not of primary importance (or not yet implemented)
- Speeding up calorimetry simulation taking advantage of active developments in the community
- ... (all other ideas that maybe you want to discuss!)

⇒ of course implementation/tuning and validation of any of fast simulation requires time and resources, but:

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



Sanghyun Ko's slides
dual-readout git

More potential use-cases that could play a role for FCC-ee simulations:

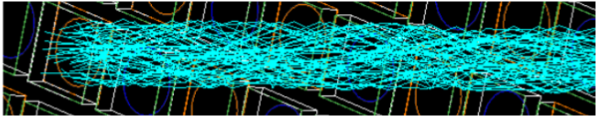
- Rough parameterisation of particles in the detectors that are not of primary importance (or not yet implemented)
- Speeding up calorimetry simulation taking advantage of active developments in the community
- ... (all other ideas that maybe you want to discuss!)

⇒ of course implementation/tuning and validation of any of fast simulation requires time and resources, but:

- it may considerably speed-up simulation (by orders of magnitude)
  - **less time needed to produce samples from final version of the detailed simulation**

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



Sanghyun Ko's slides
dual-readout git

More potential use-cases that could play a role for FCC-ee simulations:

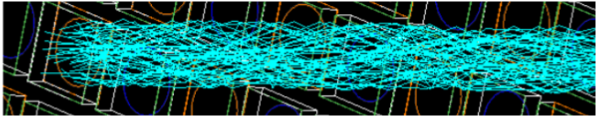
- Rough parameterisation of particles in the detectors that are not of primary importance (or not yet implemented)
- Speeding up calorimetry simulation taking advantage of active developments in the community
- ... (all other ideas that maybe you want to discuss!)

$\Rightarrow$ of course implementation/tuning and validation of any of fast simulation requires time and resources, but:

- it may considerably speed-up simulation (by orders of magnitude)
  - **less time needed to produce samples from final version of the detailed simulation**
- it can benefit from **tools that already exist**

FCC-ee usecases

Probably the only current use-case of fast simulation for FCC (except for examples) is optical photon transport defined for the dual readout calorimeter.



Sanghyun Ko's slides
dual-readout git

More potential use-cases that could play a role for FCC-ee simulations:

- Rough parameterisation of particles in the detectors that are not of primary importance (or not yet implemented)
- Speeding up calorimetry simulation taking advantage of active developments in the community
- ... (all other ideas that maybe you want to discuss!)

⇒ of course implementation/tuning and validation of any of fast simulation requires time and resources, but:

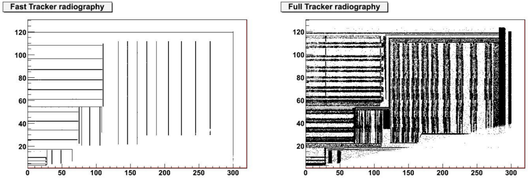
- it may considerably speed-up simulation (by orders of magnitude)
  - **less time needed to produce samples from final version of the detailed simulation**
- it can benefit from **tools that already exist**
- there is many active R&Ds in the community and there is **interest to collaborate**
  - but detector and physics specific knowledge to validate needs to come from the FCC-ee community

Faster simulation outside of FCC

Example methods:

- Simplification of detector geometry

J. Phys.: Conf. Ser. 513 022012

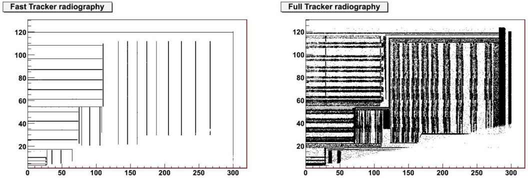


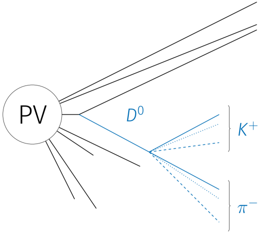
Faster simulation outside of FCC

Example methods:

- Simplification of detector geometry
- Reuse of (part of) the simulated event (backgound, pile-up, showers ...)

J. Phys.: Conf. Ser. 513 022012





D. Muller, B. Siddi, CHEP2018

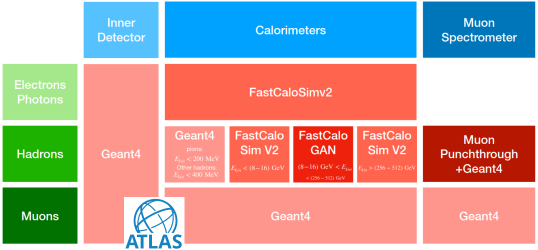
Faster simulation outside of FCC

Example methods:

- Simplification of detector geometry
- Reuse of (part of) the simulated event (backgound, pile-up, showers ...)
- Parametrisation (showers)
  - "classical" (formulae, histograms)

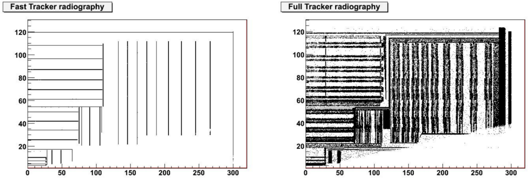
$$dE = dE(\bar{r}) = Ef(t)dt\ f(r)dr\ f(\varphi)d\varphi$$

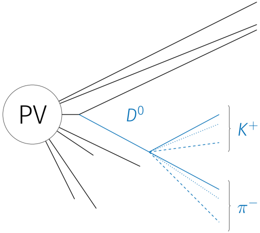
  - machine learning (ML)-based models



Atlfast3, Comput Softw Big Sci 6, 7 (2022)

J. Phys.: Conf. Ser. 513 022012



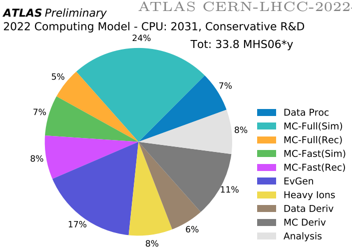


D. Muller, B. Siddi, CHEP2018

Fast shower simulation

At LHC typically most of the simulation time is spent in calorimeters.

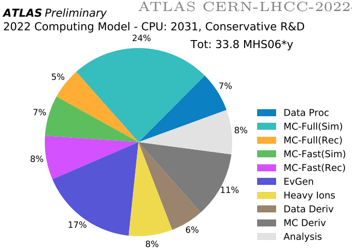
ATLAS CERN-LHCC-2022-005



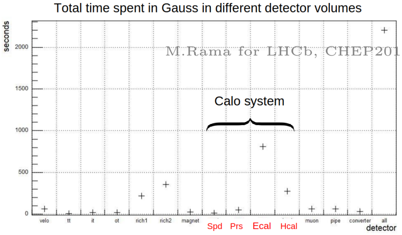
Fast shower simulation

At LHC typically most of the simulation time is spent in calorimeters.

ATLAS CERN-LHCC-2022-005



M.Rama for LHCb, CHEP2018



CPU time in calorimeter system: ~ **53**%

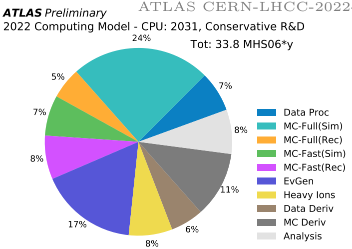
CPU time in RICH1+2: ~ **27**%

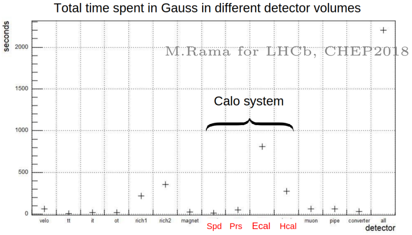
Fast shower simulation

At LHC typically most of the simulation time is spent in calorimeters.

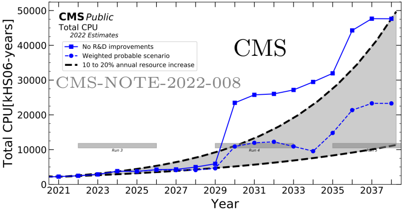
- Speed-up of simulation (generate more data within same CPU time) is vital for HL-LHC
  - to match available computing resources;
  - to provide sufficient amount of simulation data for comparison with the experimental data;

ATLAS CERN-LHCC-2022-005





CPU time in calorimeter system: ~ **53%**
CPU time in RICH1+2: ~ **27%**



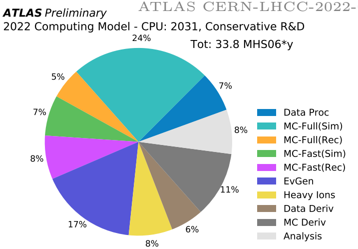
Fast shower simulation

At LHC typically most of the simulation time is spent in calorimeters.

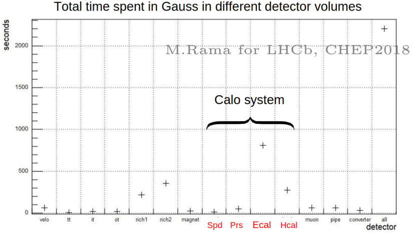
- Speed-up of simulation (generate more data within same CPU time) is vital for HL-LHC
  - to match available computing resources;
  - to provide sufficient amount of simulation data for comparison with the experimental data;

→ focus of many R&Ds is on fast shower simulation.

ATLAS CERN-LHCC-2022-005

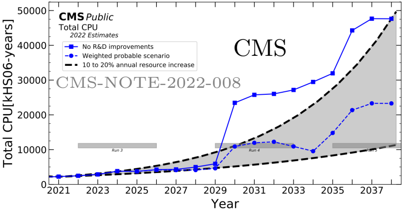


M.Rama for LHCb, CHEP2018

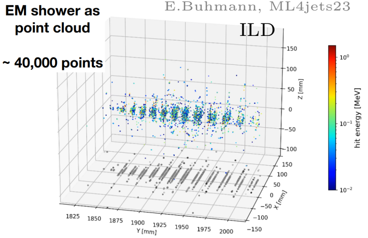


CPU time in calorimeter system: ~ **53%**
CPU time in RICH1+2: ~ **27%**

CMS-NOTE-2022-008



E.Buhmann, ML4jets23



Current activities in the community: Fast Shower Simulation Challenge

CaloChallenge: Goals

1. trigger new development
2. evaluate existing models
3. understand common issues

Current activities in the community: Fast Shower Simulation Challenge

CaloChallenge: Goals

1. trigger new development
2. evaluate existing models
3. understand common issues

Datasets

1. ATLAS γ and π (lowest granularity)
2. Simplistic cylinder e^-, low granularity
3. Simplistic cylinder e^-, high granularity

Datasets 2 and 3 were simulated with GEANT4 example Par04

Current activities in the community: Fast Shower Simulation Challenge

CaloChallenge: Goals

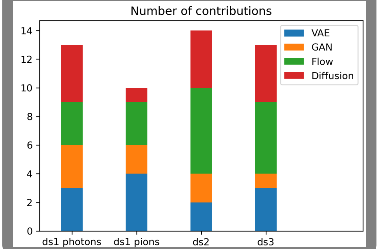
1. trigger new development
2. evaluate existing models
3. understand common issues

Datasets

1. ATLAS γ and π (lowest granularity)
2. Simplistic cylinder e^-, low granularity
3. Simplistic cylinder e^-, high granularity

Datasets 2 and 3 were simulated with GEANT4 example Par04

Contributions



Current activities in the community: Fast Shower Simulation Challenge

CaloChallenge: Goals

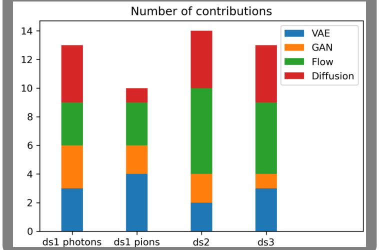
1. trigger new development
2. evaluate existing models
3. understand common issues

Datasets

1. ATLAS γ and π (lowest granularity)
2. Simplistic cylinder e^-, low granularity
3. Simplistic cylinder e^-, high granularity

Datasets 2 and 3 were simulated with GEANT4 example Par04

Contributions



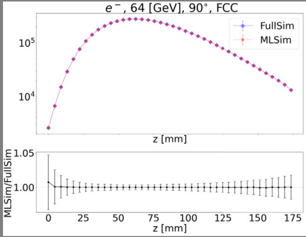
Current status

Finalisation of benchmark results (measured uniformly).
Finalisation of publication.

ML Models developed in CERN EP-SFT group

ML model: Variational Autoencoder

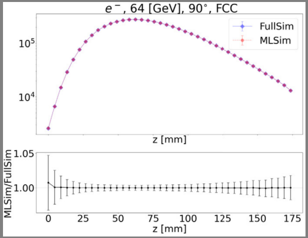
- Published with Geant4 releases in example called Par04
- Small and quick to train
- Reproduces well average shower variables (total energy, profiles and moments)
- But: blurry deposits (LHCb introduced additional sampling to fix it)



ML Models developed in CERN EP-SFT group

ML model: Variational Autoencoder

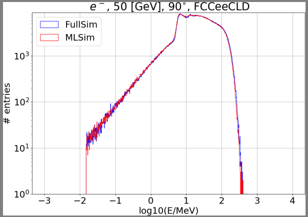
- Published with GEANT4 releases in example called Par04
- Small and quick to train
- Reproduces well average shower variables (total energy, profiles and moments)
- But: blurry deposits (LHCb introduced additional sampling to fix it)



ML transformer-based models

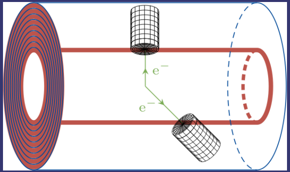
Focusing on well modelling cell-level variables as well as exploring generalisation power to multiple detectors

- Vector-quantized VAE + autoregressive:
- More promising diffusion model, CaloDiT (submitted to CaloChallenge in Dec 23)



Generic and reusable data representation

Par04: detector-independent scoring

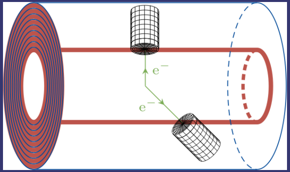


- Cylindrical scoring around the particle direction
- Same data structure at different angles
- Higher granularity than detector readout
- Cell size linked to R_M and X_0

- Implemented in standalone Geant4, in DD4hep (for FCC studies), and in Gaussino (for LHCb) → facilitates testing of new models

Generic and reusable data representation

Par04: detector-independent scoring



- Cylindrical scoring around the particle direction
- Same data structure at different angles
- Higher granularity than detector readout
- Cell size linked to R_M and X_0
- Implemented in standalone Geant4, in DD4hep (for FCC studies), and in Gaussino (for LHCb) $\rightarrow$ facilitates testing of new models

Shift of difficulty: From new ML model designs $\Rightarrow$ to the detector-specific placements of hits.

Of course cell-level data can also be used (and likely already exists), but then more modification of ML models is needed.

Training dataset for FCC

Produced dataset:

- single γ showers
- energy from 1 to 100 GeV
- 1 M showers
- proof-of concept on single $\eta = 0$ and $\phi = 0$
- energy scoring in virtual mesh

Training dataset for FCC

Produced dataset:

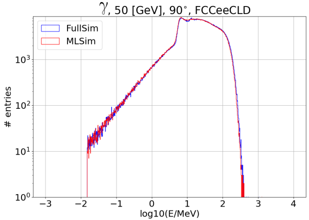
- single γ showers
- energy from 1 to 100 GeV
- 1 M showers
- proof-of concept on single $\eta = 0$ and $\phi = 0$
- energy scoring in virtual mesh

- Virtual mesh stored in EDM4hep and translated to HDF5 files (translation script)
- Allows to fetch any ML model from Par04 or CaloChallenge and retrain it on new dataset

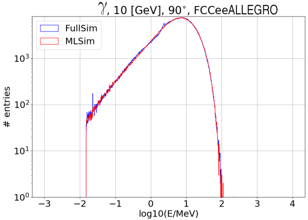
Training dataset for FCC

Produced dataset:

- single γ showers
- energy from 1 to 100 GeV
- 1 M showers
- proof-of concept on single $\eta = 0$ and $\phi = 0$
- energy scoring in virtual mesh



- Virtual mesh stored in EDM4hep and translated to HDF5 files (translation script)
- Allows to fetch any ML model from Par04 or CaloChallenge and retrain it on new dataset
- Example: ECals in FCCeeCLD and FCCeeALLEGRO, CaloDiT model

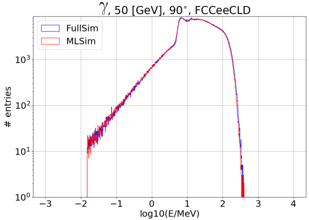


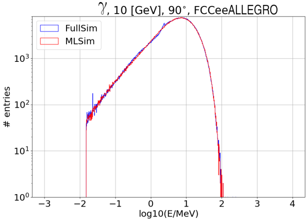
Training dataset for FCC

Produced dataset:

- single γ showers
- energy from 1 to 100 GeV
- 1 M showers
- proof-of concept on single $\eta = 0$ and $\phi = 0$
- energy scoring in virtual mesh

- Virtual mesh stored in EDM4hep and translated to HDF5 files (translation script)
- Allows to fetch any ML model from Par04 or CaloChallenge and retrain it on new dataset
- Example: ECals in FCCeeCLD and FCCeeALLEGRO, CaloDiT model





Simulation-level validation, but necessary to integrate back to FCCSW for digitisation and reconstruction. **Help needed to complete this!**

This represents just one ML model, but more could be tested (on the same dataset and with the same tools).

Simulation production

Also an example of fast simulation in DDG4

Custom classes needed to produce the virtual mesh:

- Fast simulation model that triggers the mesh measurement
  `DefineMeshModel`
- Event information that holds position and direction of virtual mesh
  `ScoreMeshEventInformation`
- Custom sensitive detector that scores relatively to the virtual mesh
  `ShowerMeshSD`
- Run and event actions (for control histograms)
  `ScoreMeshEventAction` and `ScoreMeshRunAction`

`ddFastSim` reimplements functionalities of Geant4 example Par04 to produce training shower dataset.

Simulation production

Also an example of fast simulation in DDG4

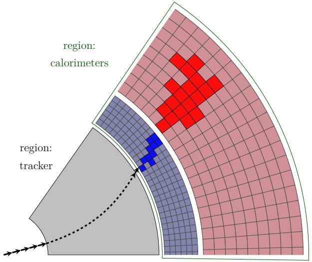
Custom classes needed to produce the virtual mesh:

- Fast simulation model that triggers the mesh measurement
  `DefineMeshModel`
- Event information that holds position and direction of virtual mesh
  `ScoreMeshEventInformation`
- Custom sensitive detector that scores relatively to the virtual mesh
  `ShowerMeshSD`
- Run and event actions (for control histograms)
  `ScoreMeshEventAction` and `ScoreMeshRunAction`

`ddFastSim` reimplements functionalities of Geant4 example Par04 to produce training shower dataset.

A second part necessary to complete ML fast shower simulation is the inference implementation. That is already implemented in `ddFastShowerML` and used for ILD studies. (not a topic of this talk)

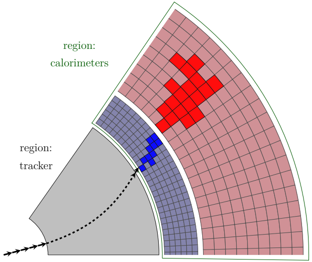
Fast simulation in DDG4



- where particles are parametrised
- which particles
- how/what happens

For fast simulation in GEANT4 one can consult the tutorial or `extended/parameterisations` examples.

Fast simulation in DDG4



- where particles are parametrised
- which particles
- how/what happens

For fast simulation in GEANT4 one can consult the tutorial or `extended/parameterisations` examples.

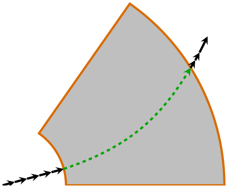
Implementation in DD4hep (DDG4) requires specific classes following DDG4 interface, but it follows overall a similar strategy.

Following slides will present how to do it based on `ddFastSim`.

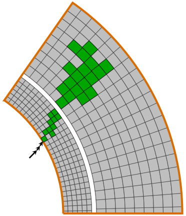
Parameterise: where?

Parameterisation/fast simulation may be attached to e.g.:

detector envelope
(single volume)



assembly of volumes
(non-physical volume)

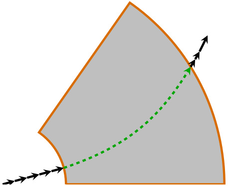


Fast simulation in Geant4 is attached to `G4Region` (associated to root G4LogicalVolume in either mass or parallel geometry).

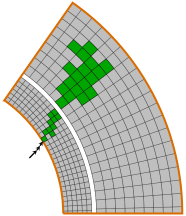
Parameterise: where?

Parameterisation/fast simulation may be attached to e.g.:

detector envelope
(single volume)



assembly of volumes
(non-physical volume)



Fast simulation in Geant4 is attached to `G4Region` (associated to root G4LogicalVolume in either mass or parallel geometry).

In DDG4 regions can be attached to volumes in XML and C++ factory, and then specified in the steering file.

Region

Par04_fullsim.xml

```
<regions>
  <region name="preECalBarrelRegion" eunit="MeV" lunit="mm" cut="0.001"
  ↪  threshold="0.001"/>
</regions>
```

```
    <detector name="preECalBarrel" type="SingleCylinder"
    ↪  region="preECalBarrelRegion">
```

Region

Par04_fullsim.xml

```
<regions>
  <region name="preECalBarrelRegion" eunit="MeV" lunit="mm" cut="0.001"
  ↪  threshold="0.001"/>
</regions>
```

```
    <detector name="preECalBarrel" type="SingleCylinder"
    ↪  region="preECalBarrelRegion">
```

SimpleCylinder_geo.cpp

```
  cylinderVol.setAttributes(lcdd, x_det.regionStr(), x_det.limitsStr(),
  ↪  x_det.visStr());
```

Region

Par04_fullsim.xml

```
<regions>
  <region name="preEC alBarrelRegion" eunit="MeV" lunit="mm" cut="0.001"
  ↪  threshold="0.001"/>
</regions>
```

```
    <detector name="preECalBarrel" type="SingleCylinder"
    ↪  region="preECalBarrelRegion">
```

SimpleCylinder_geo.cpp

```
  cylinderVol.setAttributes(lcdd, x_det.regionStr(), x_det.limitsStr(),
  ↪  x_det.visStr());
```

Par04_ddsim_steer.py

```
 model.RegionName = 'preECalBarrelRegion'
```

Parameterise: what?

- ☐ within selected volumes

  region attached to volumes;

Parameterise: what?

- ☐ within selected volumes

  region attached to volumes;

- ☑ for selected particle types

  Geant4FastPhysics attached to physics list and activated for particles (steering file);

Parameterise: what?

- ☐ within selected volumes

  region attached to volumes;

- ☑ for selected particle types

  Geant4FastPhysics attached to physics list and activated for particles (steering file);

- ☐ if trigger is issued

Parameterise: what?

- ☐ within selected volumes

  region attached to volumes;

- ☑ for selected particle types

  Geant4FastPhysics attached to physics list and activated for particles (steering file);

- ☐ if trigger is issued

  - check particle type or intrinsic information (from G4ParticleDefinition)

Parameterise: what?

- ☐ within selected volumes

  region attached to volumes;

- ☑ for selected particle types

  Geant4FastPhysics attached to physics list and activated for particles (steering file);

- ☐ if trigger is issued

  - check particle type or intrinsic information (from G4ParticleDefinition)
  - check dynamic conditions (from G4FastTrack)
    - energy, momentum, direction, ... (from G4Track)
    - local coordinates (from G4LogicalVolume)

Parameterise: what?

- ☐ within selected volumes

  region attached to volumes;

- ☑ for selected particle types

  Geant4FastPhysics attached to physics list and activated for particles (steering file);

- ☑ if trigger is issued

  implementation of dd4hep::sim::Geant4FastSimShowerModel class;

  - check particle type or intrinsic information (from G4ParticleDefinition)
  - check dynamic conditions (from G4FastTrack)
    - energy, momentum, direction, ... (from G4Track)
    - local coordinates (from G4LogicalVolume)

Parameterisation trigger needs to be set in implementation of **dd4hep::sim::Geant4FastSimShowerModel**,

Parameterise: what?

- ☑ within selected volumes

  region attached to volumes and linked to model in steering file;

- ☑ for selected particle types

  Geant4FastPhysics attached to physics list and activated for particles (steering file);

- ☑ if trigger is issued

  implementation of dd4hep::sim::Geant4FastSimShowerModel class;

  - check particle type or intrinsic information (from G4ParticleDefinition)
  - check dynamic conditions (from G4FastTrack)
    - energy, momentum, direction, ... (from G4Track)
    - local coordinates (from G4LogicalVolume)

Parameterisation trigger needs to be set in implementation of **dd4hep::sim::Geant4FastSimShowerModel**, which is linked to **region** in a steering file.

Physics and particles (1/2)

Par04_ddsim_steer.py

```
emParticles =  ["e-","e+","gamma"]
model.ApplicableParticles = emParticles
```

```
# Now build the physics list:
phys = kernel.physicsList()
ph = PhysicsList(kernel, str('Geant4FastPhysics/FastPhysicsList'))
ph.EnabledParticles = emParticles
```

Physics and particles (2/2)

DefineMeshModel.h

```
/// User callback to determine if the model is applicable for the particle type
/** Default implementation checks if the particle is registered in
↪  'ApplicableParticles'
 */
virtual bool check_applicability(const G4ParticleDefinition& particle)  override {
  // if( fastsimML.has_check_applicability )   return
  ↪  fastsimML.check_applicability(particle) ;
  /// this model can be used with all particles
  return true;
}
/// User callback to determine if the shower creation should be triggered
/** Default implementation checks if for all particles registered in 'Etrigger'
 *  the kinetic energy is bigger than the value.
 */
virtual bool check_trigger(const G4FastTrack& track)  override;
```

Parameterise: what happens?

Once particle is in a chosen volume, fulfils all conditions
– take over tracking within volume and decide what to do, e.g.:

- alter energy
- move to different position (e.g. exit from volume)
- create energy deposit(s)
- kill particle
- create secondaries

Parameterise: what happens?

Once particle is in a chosen volume, fulfils all conditions
– take over tracking within volume and decide what to do, e.g.:

- alter energy
- move to different position (e.g. exit from volume)
- create energy deposit(s)
- kill particle
- create secondaries

DefineMeshModel.cc

```
void dd4hep::sim::DefineMeshModel::modelShower(const G4FastTrack& aTrack,
↪  G4FastStep& aStep) {
```

Important! Declare model (action)

```
#include <DDG4/Factories.h>
DECLARE_GEANT4ACTION(DefineMeshModel)
```

Sensitive Detector (1/2)

What needs to be implemented is how fast simulation hit $(\bar{r}, E)$ should be processed.

ShowerMeshSD.cc

```
struct Geant4ShowerMeshCalorimeter : public Geant4Calorimeter{
```

```
template <> bool
Geant4SensitiveAction<Geant4ShowerMeshCalorimeter>::processFastSim(const
↪  Geant4FastSimSpot* spot, G4TouchableHistory* /* hist */)
{
```

Sensitive Detector (1/2)

What needs to be implemented is how fast simulation hit $(\bar{r}, E)$ should be processed.

ShowerMeshSD.cc

```
struct Geant4ShowerMeshCalorimeter : public Geant4Calorimeter{
```

```
template <> bool
Geant4SensitiveAction<Geant4ShowerMeshCalorimeter>::processFastSim(const
↪  Geant4FastSimSpot* spot, G4TouchableHistory* /* hist */)
{
```

Again required:

```
#include "DDG4/Factories.h"
DECLARE_GEANT4SENSITIVE(Geant4ShowerMeshCalorimeterAction)
```

Sensitive Detector (1/2)

What needs to be implemented is how fast simulation hit $(\bar{r}, E)$ should be processed.

ShowerMeshSD.cc

```
struct Geant4ShowerMeshCalorimeter : public Geant4Calorimeter{
```

```
template <> bool
Geant4SensitiveAction<Geant4ShowerMeshCalorimeter>::processFastSim(const
↪  Geant4FastSimSpot* spot, G4TouchableHistory* /* hist */)
{
```

Again required:

```
#include "DDG4/Factories.h"
DECLARE_GEANT4SENSITIVE(Geant4ShowerMeshCalorimeterAction)
```

Par04_fullsim.xml

```
<detector id="1" name="EMCalBarrel" type="SandwichCylinders"
↪  readout="ECalBarrelCollection">
  <sensitive type="CalorimeterSD"/>
```

Sensitive detector (2/2)

Par04_ddsim_steer.py

```
##  set the default calorimeter action
SIM.action.calo = ('Geant4ShowerMeshCalorimeterAction', {'sizeOfZCells': meshCellSizeZ,
'sizeOfRhoCells': meshCellSizeRho, 'sizeOfPhiCells': meshCellSizePhi,
'numOfZCells': meshCellNumZ, 'numOfRhoCells': meshCellNumRho, })
```

Sensitive detector (2/2)

Par04_ddsim_steer.py

```
##  set the default calorimeter action
SIM.action.calo = ('Geant4ShowerMeshCalorimeterAction', {'sizeOfZCells': meshCellSizeZ,
'sizeOfRhoCells': meshCellSizeRho, 'sizeOfPhiCells': meshCellSizePhi,
'numOfZCells': meshCellNumZ, 'numOfRhoCells': meshCellNumRho, })
```

```
def fastsimSettings(kernel):
   from g4units import GeV, MeV  # DO NOT REMOVE OR MOVE!!!!! (EXCLAMATION MARK)
   from DDG4 import DetectorConstruction, Geant4, PhysicsList
   geant4 = Geant4(kernel)
   seq = geant4.detectorConstruction()
   # Create a model for fast simulation
   model = DetectorConstruction(kernel, "DefineMeshModel" )
   # Mandatory model parameters
   model.RegionName = 'preECalBarrelRegion'
   model.Enable = True
   emParticles =  ["e-","e+","gamma"]
   model.ApplicableParticles = emParticles
   model.enableUI()
   seq.adopt(model)
   # Now build the physics list:
   phys = kernel.physicsList()
   ph = PhysicsList(kernel, str('Geant4FastPhysics/FastPhysicsList'))
   ph.EnabledParticles = emParticles
   ph.BeVerbose = True
   ph.enableUI()
   phys.adopt(ph)
   phys.dump()
SIM.physics.setupUserPhysics( fastsimSettings)
```

Summary

Fast simulation can be used for FCC-ee detectors.

- Fast shower simulation with machine-learning models is already explored

Summary

Fast simulation can be used for FCC-ee detectors.

- Fast shower simulation with machine-learning models is already explored
- Needs a full chain validation (digitisation, reconstruction, physics benchmarks)
- **Collaboration needed!**

Summary

Fast simulation can be used for FCC-ee detectors.

- Fast shower simulation with machine-learning models is already explored
- Needs a full chain validation (digitisation, reconstruction, physics benchmarks)
- **Collaboration needed!**

Repository `ddFastSim` is an example of how to define fast simulation in DDG4:

- Requires a parameterisation region (xml and C++ factory),
- Physics list needs extending (steering file),
- The core part of parameterisation is defined in a model (class inherited from **`dd4hep::sim::Geant4FastSimShowerModel`**),
- If fast sim hits are created, sensitive detector needs to define how to process them.